CONTENTS

1.0 ALLGEMEINES

1.1 REGISTRIERUNG

Die Firma APPARAT, Entwickler des NEWDOS/80, verlangt das Ausfuellen der Registrierungskarte. Hilfe und Aenderungen im Betriebssystem werden nur den Benutzern gewaehrt, die bei APPARAT erfasst sind. Wenn Sie Rueckfragen haben, geben sie immer Ihren Namen, Adresse und die Seriennummer Ihres NEWDOS/80 an.

1.2 URHEBERRECHTE

In diesem Handbuch werden mehrfach urheberrechtlich geschuetzte Produkte genannt. Es handelt sich hierbei jedoch keinenfalls um einen Vergleich mit diesen Produkten, sondern nur um eine Referenz. Ein technisches Handbuch dieser Art kommt ohne eine solche Referenz praktisch nicht aus. Wir hoffen nicht, dass damit die Urheberrechte dieser Firmen damit verletzt wurden.

Folgende Produkte werden in diesem Handbuch erwaehnt:

1. TRS-80, eingetragenes Warenzeichen der Fa. Radio Shack Inc.
2. TRSDOS, eingetragenes Warenzeichen der Fa. Radio Shack Inc.
3. VTOS, eingetragenes Warenzeichen, Firma unbekannt
4. LDOS, eingetragenes Warenzeichen der Fa. Lobo Drives Int.
5. DOUBLER, eingetragenes Warenzeichen der Fa. Percom Data Comp.
6. SCRIPSIT, eingetragenes Warenzeichen der Fa. Radio Shack Inc.

1.3 NEWDOS/80 VERSION 2

NEWDOS/80 ist ein Floppy-Betriebssystem, das fuer den TRS80 Modell 1 mit Level II - ROM, mindestens 32k RAM und mindestens einer Floppy entwickelt wurde. NEWDOS/80 Version 2 loest alle vorangegangenen Versionen des NEWDOS ab. NEWDOS/80 ist groesstenteils abwaertskompatibel mit anderen NEWDOS-Versionen und dem TRSDOS 2.3 des TRS80 Herstellers Tandy.

Der DOS und Disk-BASIC Teil von NEWDOS/80 wurde neu geschrieben. Die fruehere Anforderung an Benutzer aelterer NEWDOS- Versionen, sich zusaetzlich in jedem Falle ein TRSDOS zu kaufen, entfaellt fuer NEWDOS/80. Es ist aber auf jeden Fall wichtig, dass NEWDOS/80 Benutzer sich ein TRSDOS-Handbuch kaufen.

Dieses hier vorliegende NEWDOS/80-Handbuch baut auf das TRSDOS-Handbuch auf. Teile, die dort schon beschrieben wurden, werden hier nicht mehr extra erlaeutert. Es wird ebenfalls vorausgesetzt, das Benutzer des EDTASM bereits eine gekaufte Version vom Editor/Assembler von Radio Shack besitzen.

NEWDOS/80 Version 2 ersetzt die Version 1, die im Juni 1980 erschienen ist, und das NEWDOS/21 vom Maerz 1979. NEWDOS/80 Version 2 wurde sowohl fuer den TRS80 Modell I als auch fuer das Modell III. Diese Disketten werden jedoch getrennt verkauft, d. h. wenn ein Anwender beide Rechner besitzt muss er auch 2 Disketten mit dem NEWDOS/80 kaufen. Der TRS80 muss mit mindestens 32k Byte RAM und mindestens 1 Floppylaufwerk mit 35 (oder 40 bei Modell III) Spuren ausgeruestet sein. Dieses Laufwerk muss auf Position 0 sein. NEWDOS/80 Version 2 fuer Modell I wird auf einer Floppy mit 35 Spuren und einfacher Schreibdichte geliefert, NEWDOS/80 Version 2 fuer das Modell III auf einer Floppy mit 40 Spuren und doppelter Schreibdichte. Der Anwender muss das entsprechende Laufwerk haben.

1.4 KOPIE DER SYSTEMDISKETTE

NEWDOS/80 hat ebenso wie vorhergehende Versionen des NEWDOS kein BACKUP-Kommando. Fuer diese Funktion muss der Benutzer das COPY (siehe "COPY 5" auf Seite 14) einsetzen.

Das NEWDOS/80 ist ein sehr komplexes Betriebssystem. Aus diesem Grunde wird dem Benutzer empfohlen, sich zuerst dieses Handbuch durchzulesen um mindestens ueber die wichtigsten Unterschiede informiert zu sein, bevor er das erste Mal die Diskette in Betrieb nimmt.

Benutzen Sie moeglichst nie die Orginal-Diskette mit dem NEWDOS/80, ausser um davon Kopien zu machen. Testen Sie auch die Aenderungen zuerst an einer anderen Diskette aus, bevor Sie diese auf der Orginal-Diskette machen. Legen Sie die Orginal-Diskette nicht in das Handbuch, bewahren Sie diese an einem sicheren Ort auf. Verlorene oder zerstoerte Disketten koennen nicht ausgetauscht werden.

Wenn Sie das System hochgefahren haben, geben Sie folgendes Kommando ein:

```
LIB
```

Sie erhalten damit die Liste aller DOS-Kommandos. Diese sind in Kapitel 2 ausfuehrlich erklaert.

Geben Sie dann folgendes Kommando ein:

```
DIR,0,S,I
```

Sie erhalten damit eine Liste aller NEWDOS/80-Files. Die Funktion dieser Files, ausser NWD80V2/ILF und NWD80V2/XLF, wird in Kapitel 5 erklaert.

Mit dem Kommando

```
SYSTEM,0
```

erhalten Sie eine Liste aller Systemoptionen, die Sie ueber das Kommando SYSTEM veraendern koennen (siehe dazu "SYSTEM" auf Seite 46). Vergleichen Sie die Liste mit der Beschreibung und entscheiden Sie selbst, ob Sie daran etwas aendern muessen. Aendern Sie es aber nur dann sofort auf der Orginal-Diskette, wenn es absolut notwendig ist. Warten Sie lieber damit, bis Sie eine Kopie der Diskette haben. Wenn Sie die Systemoptionen aber gleich aendern wollen, muessen Sie zuerst den Schreibschutz entfernen. Die eingegebene Aenderung wird beim naechsten Reset aktiv.

Die Systemoptionen einer neuen Systemdiskette sind:

```
AA=N,AB=N,AC=Y,AD=Y,AE=Y,AF=Y,AG=N,AI=N,AJ=Y,AL=4/4H,
AM=10/AH,AN=0/0H,AO=0/0H,AP=0/0H,AQ=Y,AR=N,AS=Y,AT=Y,
AU=Y,AV=30/1EH,AW=2/2H,AX=90/5AH,AY=Y,AZ=N,BA=N,BC=Y,
BD=Y,BE=Y,BF=N,BG=N,BH=N,BI=0/0H,BJ=1/1H,BK=Y,BM=Y,BN=N
```

Geben Sie folgendes Kommando ein:

```
PDRIVE,0
```

NEWDOS/80 kann mit verschiedenen Floppy-Laufwerken und Interfaces arbeiten. Alle Werte der Laufwerke muessen ueber das DOS- Kommando PDRIVE (siehe "PDRIVE" auf Seite 34) eingegeben werden. Auch diese Werte werden beim Reset eingelesen.

Die PDRIVE-Parameter einer neuen Systemdiskette sind (ohne Gewaehr):

```
0*   TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
1*   TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
2*   TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
3*   TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
4    TI=CM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,DDSL=17,DDGA=2
5    TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDGA=2
6    TI=CK,TD=E,TC=39,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
7    TI=A,TD=C,TC=80,SPT=20,TSR=2;GPL=2,DDSL=17,DDGA=2
8    TI=C,TD=E,TC=40,SPT=18,TSR=3,GPL=2,DDSL=17,DDGA=2
9    TI=C,TD=G,TC=80,SPT=36,TSR=3,GPL=8,DDSL=17,DDGA=2
```

Wie auch schon im vorhergehenden Abschnitt gilt auch hier: aendern Sie diese Parameter nur dann, wenn es absolut notwendig ist.

Dem Benutzer von NEWDOS/80 ist es freigestellt, eine beliebige Anzahl von Kopien von der Orginal-Diskette fuer seinen eigenen Gebrauch zu machen. Es ist aber ausdruecklich verboten, Kopien von NEWDOS/80 oder auch nur einzelner Programme der Systemdiskette an andere weiterzugeben.

Machen sie moeglichst eine oder mehrere Kopien der Orginal-Systemdiskette, ohne vorher etwas zu veraendern.

Mit COPY 5 (siehe "COPY 5" auf Seite 14) koennen sie ein BACKUP der Systemdiskette machen. Benutzen Sie eines der folgenden Beispiele.

1. `COPY,0,0,,FMT,USD`

   Kopie einer Systemdiskette mit einem Laufwerk, beide Disketten haben die gleiche PDRIVE-Angaben.

2. `COPY,0,1,,FMT,USD`

   Kopie einer Systemdiskette mit mehreren Laufwerken, beide Disketten haben die gleiche PDRIVE-Angaben.

3. `COPY,0,0,,FMT,USD,CBF,DPDN=4`

   Kopie einer Systemdiskette mit einem Laufwerk, beide Disketten haben unterschiedliche PDRIVE-Angaben. Aendern Sie vorher die Angaben fuer Drive 4 auf Ihre passenden Werte ab und machen Sie ein Reset oder geben Sie bei PDRIVE den Parameter A mit an, um die neuen Angaben ins System zu uebernehmen.

Lesen Sie sich zuerst die Beschreibung des Kommandos COPY durch, machen sie sich dann einige Kopien des Systems. Auf diesen Kopien koennen sie sich dann die speziellen Systemeigenschaften ihres Systems mit den Kommandos SYSTEM und PDRIVE generieren.

1.5 AENDERUNGEN

Bevor sie mit dem NEWDOS/80 bereits bestehende Anwenderprogramme wie z. B. EDIT/CMD oder SCRIPSIT laufen lassen, kontrollieren sie in "Die Aenderungen im NEWDOS/80" auf Seite 170, ob Aenderungen zu diesen Programmen noetig sind.

Machen sie diese Aenderungen auf allen Kopien von NEWDOS/80, aber erst dann, wenn sie mindestens 2-3 lauffaehige Kopien von der Master-Diskette gemacht haben (siehe dazu Kapitel "Fehlerrueckmeldung und Durchfuehrung von Aenderungen" auf Seite 155)

1.6 BENUTZERHINWEISE

Haben Sie genuegend Backup-Kopien von der Master-Diskette gemacht, alle Aenderungen eingefuegt, die System- und Laufwerkparameter eingegeben, koennen Sie mit NEWDOS/80 anfangen zu arbeiten.

Bei einem Reset wird das DOS neu initialisiert, 'DOS READY' erscheint auf dem Bildschirm. Der Benutzer kann jetzt ein DOS-Kommando eingeben oder den Name eines lauffaehigen Programms. Wird ein Programmname eingegeben ohne /TYP wird als Defaultwert /CMD angenommen. Z. B.

BASIC - laden und starten des Programmes BASIC/CMD
SCRIPSIT/LC - laden und starten des Programmes SCRIPSIT

Wenn das Kommando oder das aufgerufene Programm einen oder mehrere Uebergabeparameter verlangt, so sind diese durch Komma oder Leerzeichen vom Programmnamen oder Kommando zu trennen. Z. B.

```
BASIC,6,60000
DIR 1 I A P
```

1.7 UNVERTRAEGLICHKEIT

NEWDOS/80 Version 2 fuer Modell I und NEWDOS/80 Version 2 fuer Modell III sind groesstenteils abwaertskompatibel mit NEWDOS/80 Version 1, NEWDOS21 und TRSDOS 2.3 fuer Modell I. Es ist aber trotzdem notwendig, von verschiedenen Programmen verschieden Versionen fuer verschiedene Betriebssysteme zu machen. NEWDOS/80 Version 2 fuer Modell III ist unterschiedlicher zum TRSDOS fuer Modell III als es NEWDOS/80 Version 1

fuer das Modell I zum TRSDOS war. In der Vergangenheit hat das TRSDOS relativ wenig geboten, wurde aber inzwischen durch diverse Verbesserungen erweitert, die allerdings, aus welchen Gruenden auch immer, die Betriebssysteme unvertraeglicher machten. Bei Tandy Inc. ist man dabei einen Weg gegangen, bei Apparat Inc. einen anderen. NEWDOS/80 kann und soll auch nicht nur ein Duplikat sein vom TRSDOS. Wer dies moechte, sollte sich besser ein TRSDOS kaufen.

Folgende Punkte sind gegenueber den anderen DOS-Arten zu aendern:

1.7.1 Interrupt-Routinen

Aktivieren und deaktivieren von Programmen, die ueber den 25 ms-Interrupt aufgerufen werden.

1.7.2 Aufruf von DEBUG

Aktivieren und deaktivieren von Debug. Frueher wurde dies ueber das Veraendern der Adresse 4315H und Bit 7 der Adresse 430FH gemacht. Jetzt wird es nur noch ueber Bit 7 von der Adresse 4369H oder das DOS-Kommando DEBUG (siehe "DEBUG" auf Seite 23) aufgerufen bzw. aktiviert. Keine dieser Aenderungen hat Auswirkungen auf den Aufruf ueber '123'

1.7.3 BREAK-Taste

Aktivieren und deaktivieren der Break-Taste. Frueher wurde dies ueber das Veraendern der Adresse 4313H gemacht, jetzt nur noch ueber Bit 4 der Adresse 4369H oder ueber das DOS- Kommando BREAK (siehe "BREAK" auf Seite 10 und "4419H - DOS-CALL" auf Seite 56).

1.7.4 NEXT und EOF-Handling

NEXT und EOF-Werte im FCB (siehe "FCB" auf Seite 86 und "4419H - DOS-CALL" auf Seite 56).

2.0 D O S - BEFEHLE

2.1 ALLGEMEINE VEREINBARUNGEN UND INFORMATIONEN

Alle DOS-Kommandos schliessen mit 'ENTER' ab. In der folgenden Beschreibung wird dies nicht mehr angegeben.

DOS-Kommandos haben eine maximale Laenge von 80 Zeichen zuzueglich der 'ENTER'-Taste.

Alle angegebenen Parameter in eckiger Klammer <> sind optionale Parameter. Dieser Parameter kann weggelassen werden wenn er nicht gebraucht wird, oder, wenn er benutzt wird, sind die Klammern wegzulassen.

Beispiel: <,INV>
ist einzugeben als ,INV

Alle Kommas in einem DOS-Kommando koennen durch die entsprechende Anzahl Leerzeichen (Blanks) ersetzt werden.

Alle Zahlen ohne angehaengtes 'H' werden als Dezimalzahlen interpretiert, alle Zahlen, die mit 'H' abschliessen als hexadezimaler Wert.

Beispiel: 4000H und 16384 haben den gleichen Wert

An den Stellen, wo ein Filename angegeben wird, steht in dieser Beschreibung 'FNAME' bzw. 'FNAME1', 'FNAME2' etc. der komplette Name setzt sich zusammen aus

NAME<./TYP><.PASSWORT><:DISK>

Der Name setzt sich zusammen aus 1-8 Charactern, der erste muss ein Buchstabe sein, alle weiteren entweder Buchstaben oder Zahlen.

Der 'TYP' kennzeichnet den Filetyp (z. B. CMD, BAS, OBJ, CIM, TXT, DOC, COM etc.). Ein Typ muss nicht unbedingt angegeben werden, wird er aber angegeben, muss er aus 1-3 alphanumerischen Zeichen bestehen, wobei das erste Zeichen ein Buchstabe sein muss, die weiteren Zeichen entweder Buchstaben oder Ziffern. Manche Files muessen einen Zusatz (Typ) haben (z. B. /CMD oder /JCL).

Fuer das Passwort gelten die gleichen Vereinbarungen wie fuer den Namen, es kann, aber es muss nicht unbedingt eingegeben werden. Das Passwort gilt dann, wenn eingegeben, sowohl fuer Zugriffe auf den File als auch fuer Veraenderungen.

:DISK bezeichnet die Floppy, auf der dieser File entweder generiert oder angesprochen werden soll.

Beispiele fuer Filenamen

- XFILE000/JCL.PASSWORT:0
- XFILE100:1
- XFILE200.PASSW200

Das NEWDOS/80 akzeptiert gleichermassen Gross- als auch Kleinschreibung fuer Kommandos und sonstige Eingaben.

Jedes der hier beschriebenen DOS-Kommandos ist in seiner Grundform und den moeglichen Parametern angegeben. Parameter, die wahlweise angegeben werden koennen, stehen in <> Klammern.

Manche Kommandos gehen in dieser Dokumentation ueber mehrere Zeilen. Der Benutzer sollte dieses aber als ein zusammenhaengendes Kommando betrachten und auch am Stueck (max. Eingabelaenge beachten !) eingeben.

Sofern nicht anders angegeben sind die Kommandos sowohl in DOS als auch in MINI-DOS zulaessig.

2.2 APPEND

APPEND,FNAME1,<TO,>FNAME2

Dieses Kommando haengt den File mit dem Name 'FNAME1' an den File mit dem Name 'FNAME2' an. Das Ende von 'FNAME2' wird aus dem Directory-Eintrag 'FPDE' entnommen. Es kann also Probleme geben, wenn 'FNAME2' keinen speziellen EOF-Character hat wie z. B. BASIC-Files oder Assembler-Source-Files.

APPEND kann nicht im MINI-DOS benutzt werden.

Beispiele:

1. APPEND,XXX:1,YYY/DAT:0 - Der Inhalt von dem File XXX auf der Floppy 1 wird an das Ende des Files YYY/DAT auf Laufwerk 0 angehaengt.

2. APPEND AAA TO BBB - Der Inhalt des Files AAA wird an das Ende des Files BBB angehaengt. Das DOS beginnt mit der Suche nach beiden Files auf dem Laufwerk 0.

2.3 ATTRIB

ATTRIB,FNAME<,INV><,VIS><,PROT=xxxx>
<,ACC=Passwort1><,UPD=Passwort2>
<,ASE=e><,ASC=c><,UDF=u>

Die Parameter eines Files im Directory werden veraendert

INV Der File 'FNAME' wird beim normalen DIR-Kommando nicht mehr angezeigt, nur noch bei DIR,I. Ebenso wird dieser File bei PROT=LOCK oder UNLOCK ausgenommen.

VIS Nach diesem Kommando werden sog. unsichtbare Files bei 'DIR' wieder angezeigt.

PROT=xxxx Mit diesem Kommandoteil kann der Zugriff auf einen File eingeschraenkt werden. Dies gilt jedoch nur, wenn beim Generieren dieses Files ein Passwort benutzt wurde. Folgende Moeglichkeiten sind erlaubt:

- **LOCK** Level 7 (angezeigt im Directory mit p=7) - Keine Zugriffsmoeglichkeit auf den File ausser durch das Betriebssystem.
- **EXEC** Level 6 (angezeigt im Directory mit p=6) - Lauffaehiges Programm, das nur im RUN-Mode betrieben werden kann. In BASIC nur aufrufbar mit 'RUN...'

READ Level 5 (angezeigt im Directory mit p=5) - Dieser File kann nur von der Floppy geladen oder ausgefuehrt werden, Zurueckschreiben ist nicht erlaubt.

WRITE Level 4 (angezeigt im Directory mit p=4) - Wie bei READ, nur dass ein Zurueckschreiben erlaubt ist.

RENAME oder

NAME Level 2 (angezeigt im Directory mit p=2) - Wie bei WRITE, der Filename darf aber mit dem RENAME- Kommando umbenannt werden.

KILL Level 1 (angezeigt im Directory mit p=1) - Wie bei NAME/RENAME, der File kann aber auch geloescht werden.

FULL Jede Operation ist erlaubt, der File also ungeschuetzt.

ACC=Passwort1 Passwort1 wird definiert als Zugriffspasswort fuer diesen File. Wenn keine Eingabe gemacht wird, nimmt das System als zulaessiges Passwort 8 Leerzeichen. Fuer das Passwort gilt die gleiche Vereinbarung wie fuer den Filenamen (8 Character, A-Z und 0-9, erstes Zeichen muss ein Buchstabe sein).

UPD=Passwort2 Passwort2 ist das Update-Passwort. Mit diesem Passwort hat man den vollen Zugriff auf diesen File, unabhaengig davon, was bei PROT=xxxx eingegeben wurde.

ASE=e wobei e entweder Y oder N ist. Dieser Parameter wurde hinzugefuegt um dem DOS zu erlauben, automatisch Platz fuer einen File zu reservieren (bei ASE=Y) oder um eine weitere Platzbelegung zu unterbinden (bei ASE=N). Der Defaultwert ist ASE=Y.

ASC=c wobei c entweder Y oder N ist. Dieser Parameter wurde hinzugefuegt um dem DOS zu erlauben, automatisch Platz am Ende eines Files freizugeben (bei ASC=Y) beim CLOSE oder den einmal reservierten Platz zu halten (bei ASC=N). Der Defaultwert ist ASC=Y.

UDF=u wobei u entweder Y oder N ist. Dieser Parameter wurde hinzugefuegt um dem DOS zu erlauben, den File als veraendert zu kennzeichnen, wenn ein Sektor des Files veraendert wird (UDF=Y). Bei UDF=N wird das Update-Kennzeichen geloescht.

Beispiele:

1. ATTRIB,XXX/CMD:1,UPD=ZXCVB,ACC=NMLKJ,PROT=EXEC

   Dem File XXX/CMD auf dem Laufwerk 1 werden die folgenden Attribute zugewiesen:

   - Veraenderungspasswort wird ZXCVB
   - Zugriffspasswort wird NMLKJ
   - Der Zugriff wird auf das Ausfuehren beschraenkt, der File kann nicht gelesen oder beschrieben werden.

   Da bei diesem Filenamen kein Passwort angegeben wurde, kann man annehmen, dass entweder noch kein Zugriffspasswort hatte, oder dass die Kontrolle der Passwoerter mit der Systemoption AA=N ausgeschaltet war.

2. ATTRIB YYY/DAT.OZBV INV ASE=N ASC=N UDF=N

Dieses Kommando ueberprueft zuerst, ob der File YYY/DAT das Veraenderungspasswort OZBV hat, wenn ja, bekommt der File das Attribut unsichtbar (INV), die Kennzeichen fuer zusaetzlichen reservieren und ueberfluessigen Platz loeschen werden auf nicht erlaubt gesetzt, das Kennzeichen fuer veraenderten File wird geloescht.

2.4 AUTO

AUTO,DOS-Kommando

Mit diesem Kommando kann der Benutzer ein DOS-Kommando eingeben. Dieses Kommando wird in den letzten 32 Bytes des Directory abgespeichert und bei jedem Reset automatisch ausgefuehrt.

Wenn die System-Option AB=N und BC=Y eingegeben ist kann die Ausfuehrung dieses Kommandos durch Druecken der ENTER-Taste beim Reset unterdrueckt werden. (AB=N entspr. System nicht im RUN-Modus)

Besonders nuetzlich kann dieses Kommando dann sein, wenn das System im RUN-Mode ist (AB=Y) und anstelle eines einzelnen Kommandos ein Chainfile mit einer ganzen Kommandofolge aufgerufen wird.

Beispiele fuer AUTO:

1. AUTO BASIC RUN"XXX/BAS"

   bewirkt, dass beim naechsten Einschalten oder Reset das BASIC aufgerufen wird und das BASIC-Programm XXX/BAS gestartet wird.

2. AUTO DO RSACTION

   bewirkt, dass beim naechsten Einschalten oder Reset die naechste Eingabe vom File RSACTION/JCL geholt wird, der wiederum mehrere DOS-Kommandos enthalten kann.

3. AUTO

   bewirkt, dass ab dem naechsten Einschalten oder Reset sofort die Meldung DOS-READY erscheint und das naechste DOS-Kommando von der Tastatur aus eingegeben werden kann.

2.5 BASIC2

Der Rechner verlaesst das Betriebssystem NEWDOS/80 und fuehrt nur noch das normale BASIC im ROM aus.

2.6 BLINK

Der Cursor wird auf blinken bzw. nicht blinken gestellt.

BLINK<,x>

Bei Eingabe von BLINK oder BLINK,Y beginnt der Cursor zu blinken, bei BLINK,N wird das Blinken abgestellt. Mit der SYSTEM-Option BH kann der Defaultwert beim Einschalten gesetzt werden.

2.7 BOOT

Es erfolgt ein Reset, die Floppy wird hochgefahren und das Betriebssystem neu geladen, gegebenenfalls das unter 'AUTO' abgespeicherte Kommando ausgefuert.

2.8 BREAK

BREAK<,x>

Wenn x gleich 'Y' ist hat die Break-Taste ihre normale Funktion.

Im NEWDOS/80 kann die Break-Taste auch direkt ueber die Adresse 4312H (bei Modell III Adresse 4478H) beeinflusst werden. Ein Abspeichern von C3H in diese Adresse deaktiviert die BREAK-Taste. Die alte Adresse von Version 1 (Bit 4 von 4369H) hat keine Wirkung mehr, ist aber bei Modell III verboten (System-Buffer).

2.9 CHAIN

CHAIN,FNAME<,Abschnitt>

Umschalten von Tastatureingabe auf Fileeingabe

Dieses Kommando hat die gleiche Funktion wie das Kommando DO. Es versetzt das DOS in den sogenannten Chaining- Mode, falls es nicht schon im Chaining-Mode aufgerufen wurde. Im angegebenen File muss eine Kommandofolge stehen, jedes Kommando muss mit 0DH (Return) abschliessen. NEWDOS/80 eroeffnet diesen File und bleibt entweder auf der ersten Zeile positioniert stehen (falls kein Abschnitt angegeben wurde) oder sucht in dem File den eingegebenen Abschnittsnamen und positioniert sich dort.

Sobald nun NEWDOS/80, BASIC oder ein anderes Programm von der Tastatur ueber die normale Eingaberoutine 05D9H im ROM eine Eingabe erwarten, wird die naechste Zeile vom File FNAME gelesen und anstelle der Tastatureingabe in den Eingabebuffer gestellt.

Dem Benutzer von NEWDOS steht auf der Orginal-Diskette das BASIC-Programm CHAINTST/BAS zur Verfuegung. Darin ist der Aufbau eines Chainfiles ueber ein BASIC-Programm beschrieben.

Wenn bei FNAME kein Typ angegeben wurde, nimmt das System automatisch den Typ /JCL an.

Das Kommando CHAIN kann entweder ueber DOS oder ueber BASIC mit CMD"S=...." aufgerufen werden. Wurde das Chain-Kommando eingegeben, werden die folgenden Kommandos oder Antworten aus diesem File ausgefuehrt. Danach kehrt das System in den normalen Status zurueck.

Das Kommando CHAIN kann nicht im MINI-DOS aufgerufen werden.

Der Benutzer bzw. Ersteller eines Chain-Files ist selbst dafuer verantwortlich, dass es zu keinen Fehlern oder Konflikten mit dem System oder Anwenderprogramm kommt.

Beispiele fuer DO:

1. CHAIN,XXX:0

   Das Chaining beginnt mit der ersten Zeile des Files XXX/JCL auf der Floppy 0.

2. DO,YYY/CHN:1,QQQ

   Das Chaining beginnt mit der ersten Zeile des Files YYY/CHN nach der Sektionskennung QQQ.

Unter "CHAINING" auf Seite 71 stehen die weiteren Erlaeuterungen fuer die Chain-Files.

2.10 CHNON

CHNON<,x>

Das Kommando CHNON wird waehrend des Chaining benutzt. Ein Fehler tritt auf, wenn gerade kein Chain-File eroeffnet ist. Das Kommando CHNON sollte nicht das letzte Kommando in einem Chain-File ohne Abschnitte sein, ebenso sollte es nicht das letzte Kommando vor einem Abschnittsnamen sein, da es dann bedeutungslos ist.

CHNON,N Die momentane Position innerhalb eines Chain-Files wird gespeichert und das System kehrt vom Chaining-Modus in den aufrufenden Level zurueck. Wurde der Chain-File vom DOS aus aufgerufen, wird DOS READY angezeigt. Es kann dann wieder von der Tastatur aus eingegeben werden.

CHNON,Y Die Eingabe wird wieder auf den Chain-File umgeschaltet, der an der gespeicherten Stelle weitermacht. Wenn CHNON,Y im DOS eingegeben wird, wird der momentane DOS-Level wieder verlassen.

CHNON,D Die Eingabe wird wieder auf den Chain-File umgeschaltet, der an der gespeicherten Stelle weitermacht. Wenn CHNON,D im DOS eingegeben wird, wird der momentane DOS-Level beibehalten und die im Chain-File folgenden Kommandos abgearbeitet, bis bis entweder CHNON,N, CHNON,Y, ein Abschnittsname oder das Ende des Chain-Files kommt.

2.11 CLEAR

CLEAR<,START=adr1><,END=adr2><,MEM=adr3>

Das CLEAR-Kommando hat folgende Funktionen:

1. Ruft die Funktion ROUTE,CLEAR auf

2. Loescht alle Interrupt-Routinen, die ueber den DOS-Call 4410H (Modell I) bzw. 447BH (Modell III) in der Interrupt-Warteschlange stehen. Ebenso wird die Anzeige der Uhrzeit abgeschaltet.

3. Loescht alle *name Programme, die ueber den DOS-Call 4461H in eine Eingabe-Warteschlange gestellt wurden. Wenn der Spooler aktiv ist, ist dies nicht gerade die beste Loesung fuer einen Abbruch. Der Spooler sollte durch normale Eingabe vor dem CLEAR-Kommando beendet werden.

4. Der Wert HIMEM wird auf den angegebenen Wert 'adr3' gesetzt, wenn 'adr3' nicht angegeben wurde, auf den hoechstmoeglichen Wert.

5. Das Memory wird geloescht vom Wert 'adr1' an (falls angegeben) bis zur Adresse HIMEM, oder, falls nicht angegeben, von 5200H bis HIMEM. Wenn 'adr1' eingegeben wird, muss es groesser sein als 5200H, 'adr2' muss kleiner sein als HIMEM.

Beispiele:

1. CLEAR START=6000H MEM=0DFFFH

   Alle Programme werden aus der Warteschlange herausgenommen, alle Timer geloescht, das Memory wird von 6000H bis 0DFFFH auf 0 gesetzt.

2. CLEAR

   Alle Programme werden aus der Warteschlange herausgenommen, alle Timer geloescht, der Wert HIMEM wird auf den groessten moeglichen Wert gesetzt, das Memory wird von 5200H bis HIMEM auf 0 gesetzt.

2.12 CLOCK

CLOCK<,x>

Wenn der Parameter x als 'Y' eingegeben wird, oder wenn kein Parameter eingegeben wird, wird die aktuelle Uhrzeit in Zeile 1, Spalte 53-60 angezeigt.

Die Anzeige wird abgeschaltet, wenn 'N' eingegeben wird.

Es wird darauf hingewiesen, dass die Uhr permanent Zeit verliert. Die Uhr wird nur mit einem alle 25 ms wiederkehrenden Interrupt angestossen und berechnet sich die Sekunden, Minuten und Stunden selbst. Sobald also die Interrupts abgeschaltet werden, z. B. bei Floppy-Ein- oder Ausgabe, gehen diese Interrupts verloren, die Uhr verliert damit Takte. Dies ist auch dann der Fall, wenn die Anzeige abgeschaltet wird.

2.13 CLS

Bei der Eingabe von CLS wird einfach nur das Display geloescht und der Bildschirm in den 64-Character-Mode gesetzt. Bei Modell III bleibt die 1. Zeile erhalten.

2.14 COPY

Dieses Kommando kann benutzt werden zum Kopieren einzelner Files, mehrerer Files oder ganzer Disketten.

1. COPY,FNAME1<,TO>,FNAME2<,SPDN=DI3><,DPDN=DI4>

2. COPY,$FNAME1<,TO>,FNAME2<,SPDN=DI3><,DPDN=DI4>

3. COPY,<:>DI1,FNAME1<TO,>,FNAME2<,SPDN=DI3><,DPDN=DI4>

4. COPY,<:>DI1,$FNAME1<TO,>,FNAME2<,SPDN=DI3><,DPDN=DI4>

5. COPY,<:>DI1<=tc1><,TO>,<:>DI2<=tc2>,MM/TT/YY<,Y><,N>
   <,NDMW><,FMT><SPDN=DI3><,DPDN=DI4><,NFMT>
   <,SPW=passwort1><,NDPW=passwort3><,DDND>
   <,ODN=name1><,KDN><,KDD><,NDN=name2><,SN=name3>
   <,USD><,BDU><,UBB>

6. COPY,<:>DI1<,to>,<:>DI2<=tc2>,MM/TT/JJ,CBF<,Y><,N>
   <,USR><,/EXT><,UPD><,ILF=FNAME3><,XLF=FNAME4>
   <,CFWO><,NDMW><,FMT><SPDN=DI3><,DPDN=DI4><,NFMT>
   <,SPW=passwort1><,ODPW=passwort2><,NDPW=passwort3>
   <,DDND><,ODN=name1><,KDN><,KDD><,NDN=name2>
   <,SN=name3><,USD><,UBB><,DDSL=1n1><,DDGA=gc1>

Das COPY-Kommando wurde gegenueber der Version 1 wesentlich veraendert, es ist also von jedem Anwender, egal ob alt oder neu, gut durchzulesen.

Benutzer von Modell III TRSDOS und Modell I TRSDOS 2.3B sollten sich die Aenderung #030 durchlesen.

COPY kann nicht vom MINI-DOS aufgerufen werden.

FNAME1 ist der Name des zu lesenden Files, FNAME2 ist der Name des 'Nach'-Files, der Kopie. DI1 und DI2 sind die Diskettennummern (:1, :2 oder 1, 2). DI1 und DI2 koennen gleich sein. Das 'TO' kann weggelassen werden, die Kommas koennen durch Leerzeichen ersetzt werden.

FNAME1 mit dem vorgesetzten '$' bedeutet, dass sowohl der 'VON'- als auch der 'NACH'-File auf der Diskette 0 liegen (bei Systemen mit nur einer Floppy) und diese Floppies entweder unterschiedliche oder keine Betriebssysteme haben.

Bei den Copies vom Typ 2, 3, 4, 5 und 6 muessen verschiedene Fragen beantwortet werden und gegebenenfalls verschiedene Floppies benutzt werden. Beantworten Sie die Fragen auf dem Bildschirm korrekt. Bei den bisher an den Hersteller gemeldeten Fehlern handelte es sich fast immer um Anwenderfehler.

1. Bei beendetem COPY muss die Systemdiskette in Drive 0 eingelegt werden. Benutzen Sie immer die gleiche Version von NEWDOS/80 wie beim Aufruf.

2. Wenn das COPY-Programm die Source-Diskette auf einen Drive anfordert, stecken Sie die Diskette mit dem Programm, das Sie kopieren wollen, in den entsprechenden Drive.

3. Wenn das COPY-Programm die Destination-Diskette auf einen Drive anfordert, stecken Sie die Diskette, auf die das Programm kopiert werden soll, in die ensprechende Drive.

Die zusaetzlichen Parameter fuer alle COPY's sind.

SPDN=DI3 Source PDrive Number. Dieser Parameter teilt dem System mit, dass anstelle der normal benutzten PDRIVE-Eingabe fuer den 'VON'-File die PDRIVE-Angabe der Floppy DI3 benutzt werden soll. DI3 muss einen Wert zwischen 0 und 9 haben.

DPDN=DI4 Destination PDrive Number. Dieser Parameter teilt dem System mit, dass anstelle der normal benutzten PDRIVE-Eingabe fuer den 'NACH'-File die PDRIVE-Angabe der Floppy DI4 benutzt werden soll. DI4 muss einen Wert zwischen 0 und 9 haben.

2.14.1 COPY 1

Normales Kopieren eines Files mit dem Namen 'FNAME1' in einen File mit dem Namen 'FNAME2' (duplizieren). Die entsprechenden Floppies muessen bereits im Geraet sein. Der Inhalt des Files 'FNAME1' wird nicht veraendert. Wenn beide Filenamen gleich sind, kann der erste Teil des zweiten Filenamens weggelassen werden, der zweite Teil muss allerdings mit / oder . oder : beginnen.

Beispiel:

```
COPY,FNAME1/DAT:0,TO,FNAME1/DAT:1
```

kann geschrieben werden als

```
COPY,FNAME1/DAT:0,TO,:1
```

Da das Schluesselwort 'TO' weggelassen werden kann und die Kommas durch ein Leerzeichen ersetzt werden koennen, kann dieses Kommando auch folgendermassen geschrieben werden:

```
COPY FNAME1/DAT:0 :1
```

2.14.2 COPY 2

Die gleichen Regeln fuer das COPY 1 gelten auch fuer den Teil COPY 2. Mit dem $-Zeichen wird nur dem Betriebssystem mitgeteilt, dass es zu einem Konflikt kommen kann, da z. B. die Diskette im Drive 0 kein Betriebssystem enthaelt. Es duerfen also waehrend des COPY keine System-Zugriffe erfolgen und nach dem COPY muss erst abgewartet werden, bis die System-Diskette eingelegt ist.

2.14.3 COPY 3

Das COPY 3 ist aehnlich wie das COPY 1, mit der Ausnahme, dass der Benutzer nur eine Floppy zum Kopieren zur Verfuegung hat und die 'NACH'-Diskette eine andere ist als die 'VON'-Diskette. Wenn Drive 0 angegeben ist, muessen sowohl die 'VON'- als auch die 'NACH'-Diskette das gleiche Betriebssystem haben (gleiche Version des NEWDOS/80). Andernfalls ist das COPY 4 zu benutzen.

2.14.4 COPY 4

COPY 4 ist aehnlich wie das COPY 2, mit dem Unterschied, dass die benutzten Floppies entweder unterschiedliche Versionen des NEWDOS/80 oder grundsaetzlich verschiedene Betriebssysteme haben duerfen. Sinnvollerweise sollte dieses COPY nur dann benutzt werden, wenn sowohl die 'VON'- als auch die 'NACH'-Diskette in Drive 0 sind. Andernfalls verliert man unnoetig Zeit fuer den Diskettenwechsel.

COPY 2 und 4 erlauben es dem Benutzer, Ihre Programme auf eine Diskette zu spielen, die nicht das gleiche Betriebssystem haben. Die Diskette muss nur ein Directory haben, die Daten-Diskette den File BOOT/SYS, der allerdings auch unterschiedlich sein darf von der Systemdiskette.

2.14.5 COPY 5

COPY 5 ist ein komlettes Diskettencopy, da das NEWDOS/80 kein Backup-Modul hat. Die urspruengliche Version des COPY wurde erheblich erweitert, man sollte daher die verschiedenen Moeglichkeiten genau studieren.

Die Hardwareangaben fuer die benutzten Diskettenlaufwerke werden der Systemfloppy entnommen, wo sie ueber das Kommando 'PDRIVE' festgelegt und veraendert werden koennen.

Beide Drives, sowohl 'VON' als auch 'NACH', muessen die gleiche Anzahl von Granules per Spur und Sektoren je Granule haben. Andernfalls muss das COPY 6 benutzt werden.

Die 'NACH'-Diskette kann mehr Spuren haben als die 'VON'-Diskette, die zusaetzliche Anzahl freier Granules wird in das Directory eingetragen (nicht bei Option 'BDU').

Folgende Optionen sind bei COPY 5 definiert:

=tc1 Das System benutzt den dezimalen Wert von tc1 als Anzahl der Spuren der 'VON'-Diskette. Wenn dieser Parameter nicht angegeben ist, wird der vorgegebene Wert von 'PDRIVE' benutzt. Der Wert von tc1 muss zwischen 18 und 96 liegen.

=tc2 Das System benutzt den dezimalen Wert von tc2 als Anzahl der Spuren der 'VON'-Diskette, ohne diesen Parameter ebenfalls den Defaultwert von 'PDRIVE'. Der Wert von tc2 muss ebenfalls zwischen 18 und 96 liegen.

Y Das System kopiert in jedem Falle, egal was vorher auf der 'NACH'-Diskette war. 'Y' kann nicht in Verbindung mit N, ODN, ODPW, DDND, KDN oder KDD benutzt werden.

N Die 'NACH'-Diskette darf keine Daten enthalten. Sobald das COPY-Programm Daten findet, bricht es ab. 'N' kann nicht in Verbinding mit Y, ODN, ODPW, DDND, KDN oder KDD benutzt werden.

NDMW (No Diskette Mount Waits) das COPY-Programm nimmt an, dass sich alle benoetigten Disketten bereits in den Laufwerken befinden und keine Rueckfragen noetig sind. Wenn ein Fehler auftritt wird das COPY abgebrochen.

FMT Format. Die 'NACH'-Diskette wird vor dem Kopieren formatiert. Es kann nur entweder der Parameter FMT oder NFMT benutzt werden, nicht beide gleichzeitig. Wird weder FMT, NFMT noch NDMW angegeben, fragt das System den Operator 'FORMAT DISKETTE? (Y OR N)'. Wird weder FMT noch NFMT angegeben, dafuer aber der Parameter NDMW, wird automatisch ein Format durchgefuehrt.

NFMT (No ForMaT) die 'NACH'-Diskette wird vor dem COPY nicht formatiert, muss also eine bereits formatierte Floppy sein.

SPW=Passwort1 (Source PassWord) Wenn die Benutzung der Passwoerter freigegeben wurde (SYSTEM AA=Y und AR=N), muss das Passwort der 'VON'-Diskette eingegeben werden. Trifft dieses Passwort nicht zu, erfolgt ein Abbruch der Copyfunktion.

NDPW=passwort3 (New Destination PassWord) Die 'NACH'-Diskette erhaelt das neue Passwort 'passwort3'. Kann nicht in Verbindung mit der Option BDU benutzt werden.

DDNT (Display Destination Name and daTe) Name und Datum der 'NACH'-Diskette werden angezeigt. Der Benutzer hat die Moeglichkeit, das COPY danach abzubrechen, wenn z. B. eine falsche Diskette benutzt wurde. Kann nicht in Verbindung mit Y, N oder NDMW benutzt werden.

ODN=Name1 (Old Destination Name) Das COPY erfolgt nur, wenn der eingegebene Name 'Name1' mit dem Namen der Diskette uebereinstimmt. Kann nicht in Verbindung mit den Optionen Y, N oder NDMW benutzt werden.

KDN (Keep Destination Name) die 'NACH'-Diskette behaelt ihren alten Namen, der Name der 'VON'-Diskette wird nicht mitkopiert. Nicht in Verbindung mit Y, N, BDU und NDN.

KDD (Keep Destination Date) Die 'NACH'-Diskette behaelt ihr altes Datum, das Datum der 'VON'-Diskette wird nicht mitkopiert. Nicht in Verbindung mit Y, N, BDU und NDN.

NDN=Name2 (New Destination Name) Der Name der 'VON'-Diskette wird nicht mitkopiert, die 'NACH'-Diskette hat nach dem Kopieren den Namen Name2. Nicht in Verbindung mit BDU und KDN.

USD (Use Source Date) Anstelle des im Kommando angegebenen Datums wird das Datum der 'VON'-Diskette kopiert. Nicht in Verbindung mit KDD oder BDU.

SN=Name3 (Source diskette Name) Falls die 'VON'-Diskette nicht den Namen 'Name3' hat besteht die Moeglichkeit, das COPY abzubrechen.

BDU (Bypass Directory Update) Das Directory wird nicht kopiert. Kann dann benutzt werden, wenn zum Beispiel die 'VON'-Diskette kein Directory hat (reine Datendiskette) oder wenn dieses Directory zerstoert ist, die Daten aber erhalten werden sollen. Ebenso, wenn das Directory nicht dem erforderlichen Standard entspricht. Nicht in Verbindung mit KDN, NDN, NDPW und USD.

UBB (Use Big Buffer) Mit diesem Parameter hat das NEWDOS/80 Version 1 den ganzen zur Verfuegung stehenden Buffer bis HIMEM benutzt. Diese Angabe wird jedoch in Version 2 nicht mehr benoetigt, da das System hier in jedem Falle den ganzen Buffer von 7000H aufwaerts benutzt. Dieser Parameter wird also ignoriert.

2.14.6 COPY 6

COPY 6 unterscheidet sich nur darin von COPY 5, dass die Option CBF (Copy By File) eingebaut wurde. Diskette 1 und 2 koennen unterschiedlich sein (DI1-DI2). In folgenden Punkten unterscheidet sich COPY 5 von COPY 6:

Wenn die Option 'NFMT' angegeben wird, duerfen weder Y, N, KDN, KDD, NDN, BDU, USD, NDPW, DDST, DDGA noch tc2 auftreten. ODPW kann angegeben werden, System-Files werden nicht kopiert, wenn sie nicht im Directory der 'NACH'-Diskette stehen.

Wird 'NFMT' nicht angegeben, wird die 'NACH'-Diskette normal formatiert wie mit dem Kommando 'FORMAT'. DIR/SYS (Directory) und BOOT/SYS (Bootstrap-Loader) werden auch angelegt. Dann erfolgt der Uebertrag des Directorys. Dies ist deshalb notwendig, weil die System-Files immer den gleichen Platz im Directoy belegen muessen.

CBF (COPY by File) Anstelle eines COPY von ganzen Spuren werden einzelne Files kopiert.

USR Es werden nur Benutzer-Files kopiert, System- und unsichtbare Files sind ausgeschlossen.

/EXT Es werden nur Files kopiert, deren Zusatz /EXT ist. Der Zusatz hat 0 bis 3 Character (nach dem '/'). So

koennen z. B. alle Files mit /CMD, /BAS oder /X kopiert werden.

ILF=FNAME3 Include List File. FNAME3 ist ein File, der die zu kopierenden Filenamen enthaelt. Dieser File muss nicht auf der 'VON'-Diskette liegen. Es ist kein Fehler, wenn ein File aus der Liste nicht auf der Floppy gefunden wird. Innerhalb der Liste muessen die Files komplett mit Namen und Zusatz stehen und mit einem Return (0DH) abschliessen.

XLF=FNAME4 Exclusion List File. FNAME4 ist ein File, der die nicht zu kopierenden Filenamen enthaelt. Er hat die gleiche Struktur wie der ILF-File. Es ist ebenfalls kein Fehler, wenn ein File, der in der Liste steht, nicht auf der Floppy ist. XLF wird nur benutzt, wenn ILF nicht als Parameter aufgefuehrt ist.

CFWO (Check File With Operator) Die ganze Diskette wird, ein File nach dem anderen, kopiert. Vor dem COPY eines Files wird der Filename am Bildschirm angezeigt, durch Eingabe von 'Y' wird dieser File kopiert, durch Eingabe von 'N' wird dieser File uebersprungen. Durch die Eingabe von 'Q' kann der Kopiervorgang abgebrochen werden. Zusammen mit dem Parameter 'NFMT' werden System-Files (.../SYS) nicht kopiert, ausser sie stehen bereits im Directory. Wird CFWO nicht eingegeben, werden alle Source-Files kopiert und die System-Files, die bereits im Directory stehen. Ausgenommen sind DIR/SYS und BOOT/SYS.

ODPW=Passwort2 (Old Destination PassWord) Wenn gleichzeitig der Parameter NFMT eingegeben wird, die Passwoerter aktiv sind und der Systemparameter AR=N eingegeben ist, muss das richtige Passwort fuer die 'NACH'-Diskette angegeben werden. Bei falschem Passwort erfolgt ein Abbruch der Funktion.

DDSL=lnl (Destination diskette Directory Starting Lump) Adresse des 'Lumps', auf dem das Directory beginnen soll. Ist DDSL nicht angegeben, wird der Wert genommen, der beim Kommando PDRIVE vereinbart wurde.

DDGA=gcl (Destination diskette Directory Granule Allocation) Die Formatierungsroutine legt das Directory mit der in GCl angegebenen Laenge (Granules) an. Der Wert GCl kann zwischen 2 und 6 liegen. Wird DDGA nicht angegeben, gegeben, wird automatisch die Laenge aus der Angabe von PDRIVE geholt. Kann nicht in Verbindung mit NFMT angegeben werden.

Wenn waehrend des Format 6 beim COPY die Diskette voll ist, wird die Meldung "DISKETTE FULL - name/ext" ausgegeben. Dieser File ist dann zwar noch auf der Diskette vorhanden, der EOF-Wert wurde jedoch auf 0 gesetzt.

Ein COPY einer einzelnen Diskette kann nicht als DOS-CALL ausgefuehrt werden (z. B. von BASIC aus), da das COPY das Memory unter 7000H belegt und dadurch zu viele Diskettenwechsel noetig waeren.

Beim COPY oder FORMAT kann, wenn der Parameter NDMW nicht angegeben wurde, jederzeit diese Funktion unterbrochen werden, indem man den rechten Pfeil ('Right Arrow') drueckt. Danach kann man mit 'ENTER' entweder fortfahren oder mit dem Pfeil nach oben ('Up Arrow') die Funktion abbrechen. Mit 'Up Arrow' kann jederzeit abgebrochen werden, sind Sie aber in jedem Falle vorsichtig, da in diesem Moment der Status der Diskette nicht bekannt ist, speziell beim Formatieren.

Mit dem COPY-Kommando kann man von einer Standard-Diskette mit 5" und 40 Spuren, doppelte Aufzeichnungsdichte und einseitig

beschrieben, beim Modell III Files in das NEWDOS/80-System kopieren.

Dies gilt ebenfalls 1t. Aenderung #030 fuer Floppies mit 35 Spuren, einseitig beschrieben mit einfacher Schreibdichte fuer Modell I TRSDOS 2.3B.

Dabei gelten folgende Einschraenkungen:

- NEWDOS/80 kann keine TRSDOS-Disketten von Modell III formatieren. Ist jedoch eine Diskette einmal formatiert, kann sie jederzeit im NEWDOS/80 mit dem COPY Format 5 und den Optionen NFMT und BDU kopiert werden.
- Der Benutzer hat darauf zu achten, dass die Systemparameter PDRIVE bei der TRSDOS-Diskette richtig eingegeben sind (siehe Beschreibung des Kommandos "PDRIVE" auf Seite 34 und Beispiele 1 und 3).
- Ein File muss nicht vor dem Kopieren von einer TRSDOS-Diskette angelegt werden, NEWDOS/80 reserviert automatisch den Platz und macht den Eintrag im Directory.
- Die Formate 1, 2, 3, 4 oder 6 des COPY koennen benutzt werden, um Files von oder auf eine TRSDOS-Diskette zu kopieren. FMT darf jedoch nicht angegeben werden. Wenn Format 6 des COPY benutzt wird und die 'NACH' oder die 'VON'-Diskette ist eine TRSDOS-Diskette, koennen keine Systemfiles (FPDE Byte 1 Bit 6=1) kopiert werden.
- Files koennen jederzeit von TRSDOS auf NEWDOS/80 kopiert werden, was jedoch nicht heisst, dass sie auch benutzbar sind.

Beispiele fuer COPY:

1. COPY XXX:1 YYY:1

   Bei diesem COPY Format 1 wird der File XXX von der Diskette 1 auf den File YYY auf der gleichen Diskette kopiert.

2. COPY,AAA,BBB:2

   Bei diesem COPY Format 1 wird der File AAA von der Diskette 0 an gesucht. Wenn er gefunden ist, wird er kopiert auf den File BBB auf der Floppy 2.

3. COPY SUPERZAP/CMD:0 :3

   Bei diesem COPY Format 1 wird der File SUPERZAP/CMD von der Diskette in Laufwerk 0 auf die Diskette in Laufwerk 3 kopiert. Da sowohl beide Namen als auch ihre Zusaetze gleich sind, kann man bei der Zielangabe den zweiten Namen weglassen.

4. COPY XXX:1 2 SPDN=9

   Bei diesem COPY Format 1 bewirkt der Parameter SPDN=9, dass fuer die Zeit des Kopierens angenommen wird, dass die Floppy in Laufwerk 1 die Characteristik von der Floppy in Laufwerk 9 hat. Wenn wir annehmen, dass es sich um ein Modell III handelt (siehe Beispiel "PDRIVE" auf Seite 34), wird dieses COPY den File XXX vom Laufwerk 1 mit einer TRSDOS-Diskette auf die NEWDOS/80-Diskette in Laufwerk 2 kopieren.

5. COPY $XXX:1,YYY:2

   Bei diesem COPY Format 2 ist die 'NACH'-Diskette nicht die gleiche wie die 'VON'-Diskette (unterschiedliche

Systeme). DOS fragt beim Kopieren jeweils nach der System-Diskette und der 'VON'-Diskette.

6. COPY,$XXX:1 YYY:2

Bei diesem COPY Format 2 ist die 'NACH'-Diskette nicht die gleiche wie die 'VON'-Diskette (unterschiedliche Systeme). DOS fragt beim Kopieren jeweils nach der System-Diskette und der 'VON'-Diskette.

7. COPY 1 XXX YYY/DAT

Bei diesem COPY Format 3 ist die 'NACH'-Diskette nicht die gleiche wie die 'VON'-Diskette (unterschiedliche Systeme), aber beide Floppies werden in Drive 1 benutzt. DOS fragt beim Kopieren jeweils nach der 'VON'-Diskette und der 'NACH'-Diskette, so wie sie gebraucht werden. Beachten Sie, dass kein Filename bei Format 3 und 4 eine Drivenummer enthalten darf.

8. COPY 0 XXX/DAT /DAT

Bei diesem COPY Format 3 wird der File XXX/DAT auf eine andere Floppy in einen File mit dem gleichen Namen kopiert. Beide Disketten werden in Drive 0 benutzt. DOS fragt beim Kopieren jeweils nach der 'VON'-Diskette und der 'NACH'-Diskette, so wie sie gebraucht werden. Wenn Drive 0 benutzt wird und dieses Format 3 von COPY anstelle von Format 4, muessen beide Floppies ein gleiches NEWDOS/80-System haben. Es muss das gleiche NEWDOS/80-System sein wie auf der Floppy, als das COPY aufgerufen wurde.

9. COPY 0 $XXX/DAT /DAT

Bei diesem COPY Format 4 wird der File XXX/DAT auf eine andere Floppy in einen File mit dem gleichen Namen kopiert (wie schon im vorangegangenen Beispiel). Beide Disketten werden in Drive 0 benutzt. Der Unterschied ist jedoch, dass die Floppies kein gleiches Betriebssystem haben. Deshalb fragt DOS jeweils nach der System-, 'VON'- und 'NACH'-Diskette, so wie sie benoetigt werden.

10. COPY 0 $XXX XXX SPDN=9

Bei diesem COPY Format 4 wird der File XXX/DAT auf eine andere Floppy in einen File mit dem gleichen Namen kopiert (wie schon im vorangegangenen Beispiel). Beide Disketten werden in Drive 0 benutzt. Der Unterschied ist jedoch der, dass Drive 0 waehrend des COPY zwei verschiedene PDRIVE-Werte annimmt. Die Standardangabe von Drive 0 ist gueltig beim Aufruf von COPY und beim Schreiben des 2. Files, und die PDRIVE-Angabe fuer Drive 9 werden beim Lesen des 'VON'-Files herangezogen. Beim Aufruf wurde der 2. Name nicht abgekuerzt, da es nichts abzukuerzen gab.

11. COPY 0 1 06/01/80 FMT

Dieses COPY Format 5 ist die einfachste Art, eine ganze Diskette zu kopieren. Damit wird die Floppy in Drive 0 auf eine Floppy in Drive 1 kopiert. Haben die Floppies unterschiedliche Anzahl von Spuren, so muss die 'VON'-Diskette weniger haben als die 'NACH'-Diskette. Dem Operator wird mitgeteilt, wann eine Floppy in welches Laufwerk zu schieben ist, sowie eventuelle Fehler. Der Default-Parameter 'Y' ist aktiv, d. h., dass es egal ist, ob die 'NACH'-Diskette Daten enthaelt oder nicht. Vor dem Kopieren wird die 'NACH'-Diskette formatiert und sie erhaelt dann den Namen und das Passwort der 'VON'-Diskette. Als Datum wird der 06/01/80 eingetragen. Hat die 'NACH'-Diskette mehr Spuren als die 'VON'-Diskette, so wird sie trotzdem voll formatiert und das Directory richtig angelegt, so dass sie nach dem Kopieren sofort normal zu benutzen ist.

12. COPY 0 1,,NFMT

Dieses COPY Format 5 ist ein anderes Beispiel fuer das Kopieren von ganzen Floppies. Der einzige Unterschied zwischen diesem Beispiel und dem Beispiel davor ist der, dass die 'NACH'-Diskette nicht formatiert wird und das aktuelle Datum (DATE) wird auf der Floppy eingetragen.

13. COPY,0,0,06/01/80,NFMT,USD,KDN,ODN=NAME1,SN=NAME2

Dieses COPY Format 5 ist ein weiteres Beispiel fuer das Kopieren von ganzen Floppies. Der Unterschied zum vorangegangenen Beispiel ist folgender:

- Beide Floppies ('VON' und 'NACH') kommen in Drive 0
- Eine Meldung wird gemacht, wenn die 'VON'-Diskette nicht den angegebenen Namen hat
- Eine Meldung wird gemacht, wenn die 'NACH'-Diskette nicht den angegebenen Namen hat
- Die 'NACH'-Diskette behaelt ihren Namen
- Die 'NACH'-Diskette erhaelt ihr Datum von der 'VON'-Diskette

Da die volle Floppy in einem Laufwerk kopiert wird, muessen oefter die Floppies getauscht werden. Aus diesem Grunde kann auch dieses COPY nicht als DOS-CALL aufgerufen werden (z. B. von BASIC aus).

14. COPY 0,1,,FMT,CBF

Dieses COPY in Format 6 ist eines der einfachsten Beispiele ueber das Kopieren mehrerer Files. Die 'NACH'-Diskette befindet sich im Laufwerk 1 und wird zuerst formatiert und erhaelt ihren Namen und das Datum sowie das Passwort von der 'VON'-Diskette (in Laufwerk 0). Danach werden alle Files der 'VON'-Diskette, ausser BOOT/SYS und DIR/SYS, auf die 'NACH'-Diskette kopiert.

15. COPY 0,1,,NFMT,CBF

Dieses COPY in Format 6 ist ein weiteres einfaches Beispiel fuer das Kopieren mehrerer Files. Die 'NACH'-Diskette befindet sich ebenfalls in Laufwerk 1. Alle Files, die noch nicht existieren auf der 'NACH'-Diskette, werden von der 'VON'-Diskette kopiert. Die 'NACH'-Diskette wird vorher nicht formatiert, sie behaelt sowohl ihren Namen, ihr Passwort und ihr Datum.

16. COPY 0,1,,NFMT,CBF,USR

Dieses COPY in Format 6 arbeitet genau wie das vorangehende Beispiel mit dem Unterschied allerdings, dass keine System-Files und keine Files mit dem Attribut 'INVISIBLE' kopiert werden.

17. COPY,0,1,,NFMT,CBF,USR,UPD

Dieses COPY in Format 6 arbeitet genau wie die vorangehenden Beispiele, nur dass keine System-Files und keine Files mit dem Attribut 'INVISIBLE' kopiert werden, sowie keine Files, deren Update-Flag nicht gesetzt ist. Es werden also nur Files kopiert, die seit dem letzten Kopieren veraendert wurden (ausser BOOT/SYS und DIR/SYS).

18. COPY,2,3=60,06/01/80,FMT,NDMW,CBF,DDSL=29,DDGA=4

Dieses COPY in Format 6 wartet nicht, bis eine Diskette im Laufwerk ist, sondern beginnt gleich mit dem Format. Die 'NACH'-Diskette wird sofort mit 60 Spuren formatiert, das Directory beginnt auf 'Lump' 29 und ist 4 Granules lang.

Alle Files der 'VON'-Diskette ausser BOOT/SYS und DIR/SYS werden kopiert.

19. COPY 2 3 06/01/80,CBF,CFWO,NFMT

Dieses COPY in Format 6 formatiert die Diskette nicht, es nimmt an, dass sie bereits formatiert und in Ordung ist sowie dass sie Files enthalten kann. Fuer jeden File auf der 'VON'-Diskette (ausser BOOT/SYS und DIR/SYS) fragt das System, ob dieser File kopiert werden soll. Wenn auf diese Art nach allen Files der 'VON'-Diskette gefragt wurde, beginnt das Kopieren. System-Files, die bereits auf der 'NACH'-Diskette sind, werden nicht kopiert.

2.15 CREATE

Reservieren von Platz fuer einen File auf der Floppy

Mit diesem Kommando kann man auf der Floppy Platz reservieren und bei Angabe des entsprechenden Parameters eine bestimmte Anzahl von Records in den File schreiben. Der File wird, soweit es moeglich ist, an einem Stueck erzeugt, also moeglichst ohne Extensions.

Es gibt Faelle, in denen ein Anwenderprogramm auf bereits existierende Files zugreifen will, auch dann, wenn dieser File noch nicht die gewuenschten Daten enthaelt. Hier muss der Anwender ueber dieses Kommando diesen File vorher erzeugen. Ebenfalls besteht die Moeglichkeit, durch vorheriges Reservieren des Fileplatzes einen einen File am Stueck zu erzeugen, er liegt dann also nicht quer ueber die Diskette verstreut. Dies spart ggf. eine gewisse Zeit beim Zugriff.

CREATE,FNAME<,LRL=lnl><,REC=anz><,ASE=yn><,ASC=yn>

Das DOS erzeugt einen neuen File mit dem Namen FNAME oder aendert ggf. die Attribute eines vorhandenen Files FNAME.

LRL=lnl gibt die Laenge der Records an, LRL muss einen Wert haben von 1 bis 255, der Defaultwert ist 255. LRL wird im Directory eingetragen.

REC=anz gibt die Anzahl der Records an, die dem File zugeordnet werden. REC wird ebenfalls im Directory eingetragen. eingetragen.

ASE=yn Dieser Parameter gibt an, ob das DOS bei weiteren Zugriffen den Filebereich vergroessern darf oder nicht. Bei ASE=Y wird dies erlaubt, bei ASE=N darf der File nicht vergroessert werden, auch dann nicht, wenn eine Vergroesserung erforderlich waere. ASE=Y ist der Defaultwert.

ASC=yn Dieser Parameter gibt an, ob das DOS bei weiteren Zugriffen den Filebereich verkleinern darf oder nicht. Bei ASC=Y wird dies erlaubt, bei ASE=N darf der File nicht verkleinert werden, auch dann nicht, wenn eine Verkleinerung beim CLOSE moeglich waere. ASC=Y ist der Defaultwert.

Bei der Durchfuehrung dieses Kommandos werden (anz) Records von der Laenge (lnl) mit Nullen gefuellt auf die Floppy geschrieben. Der EOF-Wert wird im Directory eingetragen. Bei ASE=N wird eine automatische Erweiterung unterbunden, bei ASC=N wird die automatische Freigabe nicht benutzten Platzes beim CLOSE unterdrueckt.

Beispiele fuer CREATE:

1. CREATE,XXX:1,LRL=30,REC=2000

   Der File XXX wird, falls er noch nicht existiert, auf der Diskette 1 erzeugt und 2000 Records mit der Laenge von 30 Bytes, alles 0H, auf die Floppy geschrieben. Der EOF-Wert von 60000 wird ins Directory eingetragen. Eine Freigabe ueberfluessigen Platzes sowie die Moeglichkeit der Erweiterung sind erlaubt.

2. CREATE,YYY:2,200,ASE=N,ASC=N

   Der File XXX wird, falls er noch nicht existiert, auf der Diskette 2 erzeugt und 200 Records mit der Laenge von 256 Bytes, alles 0H, auf die Floppy geschrieben. Der EOF-Wert von 51200 wird ins Directory eingetragen. Eine Freigabe ueberfluessigen Platzes sowie die Moeglichkeit der Erweiterung sind nicht erlaubt.

3. CREATE,ZZZ:0

   Der File XXX wird, falls er noch nicht existiert, auf der Diskette 0 erzeugt. Als Recordlaenge werden 256, als EOF-Wert wird 0 ins Directory eingetragen. Eine Freigabe ueberfluessigen Platzes sowie die Moeglichkeit der Erweiterung sind erlaubt.

2.16 DATE

DATE,mm/dd/yy

Mit diesem Kommando wird das Datum in die mitlaufende Uhr eingesetzt. mm ist der Monat (Wert zwischen 01 und 12), dd der Tag (01-31), yy das Jahr (00-99). Bei dem Kommando werden die eingegebenen Zahlen nicht kontrolliert, man kann also auch im deutschen Format dd/mm/yy eingeben. Allerdings wird dann, wenn um 24:00:00 die Uhr auf 00:00:00 weiterschaltet, der amerikanische Tag, bei uns also der Monat, um 1 erhoeht. Der (amerikanische) Monat und das Jahr werden nicht veraendert. Beim Reset wird das Datum entsprechend der Systemoption AY oder AZ gesetzt.

Beispiele fuer DATE:

1. DATE

   Das aktuelle Datum wird angezeigt.

2. DATE,05/11/81

   Das aktuelle Datum wird auf den 05.11.81 gesetzt.

2.17 DEBUG

DEBUG<,x>

Wenn fuer x (Y) oder nur Y oder nichts eingegeben wird, ist DEBUG aktiviert aber noch nicht gestartet. Gestartet wird DEBUG erst, wenn die Break-Taste gedrueckt wird (Systemeingabe AE=N).

Bei x = N oder (N) wird der DEBUG-Aufruf unterdrueckt.

Wenn die Systemoption AE=Y ist, hat dieses Kommando keine Auswirkung auf die Funktion '123' (gleichzeitiges Druecken der Tasten 1, 2 und 3).

Siehe Beschreibung von Debug in Kapitel "DEBUG" auf Seite 65.

2.18 DIR

DIR<:><DI><,S><,A><,I><,U><,/EXT><,P>

Mit diesem Kommando kann der Inhalt des Directory auf dem Bildschirm oder dem Drucker angezeigt werden. Wird der Parameter DI nicht mit angegeben, wird das Laufwerk angezeigt, das bei der Systemoption AN=x angegeben ist.

Die erste Zeile der Anzeige enthaelt die Laufwerksnummer, den Diskettennamen, das Datum der Diskette, Anzahl der freien Eintraege im Directory und Anzahl der freien Granules.

Wenn der Parameter 'A' mit angegeben ist, wird ein Eintrag pro Zeile folgendermassen angezeigt:

1. der Filename
2. die Erweiterung des Filenamens (/xxx) falls vorhanden
3. der EOF-Wert des Files im Format xxx/yyy, wobei xxx die relative Sektornummer innerhalb des Files ist und yyy das relative Byte innerhalb des Sektors
4. die Recordlaenge des Files (LRL) in Bytes
5. die Anzahl der Records im File
6. die Anzahl der Granules, die der File belegt
7. die Anzahl der Erweiterungen des Files. Diese Anzahl geteilt durch 4 (aufgerundet) ergibt die Anzahl der Directory-Eintraege.
8. 12 Flags mit folgender Bedeutung:

   S — Systemfile (werden nur mit dem Parameter 'S' angezeigt).

   I — unsichtbare Files (werden nur mit dem Parameter 'I' angezeigt).

   U — veraenderte Files. Dies sind die Files, die angesprochen wurden nach dem Erzeugen.

   E — Files, die nicht erweitert werden duerfen.

   C — Files, deren unbenutzter Platz nicht freigegeben werden darf.

6...9	noch nicht benutzt (ev. spaeter)
U	Es gibt ein Update-Passwort fuer diesen File.
A	Es gibt ein Zugriffs-Passwort fuer diesen File.
L	Level des Files (siehe "ATTRIB" auf Seite 7).

Wenn der Parameter 'U' angegeben ist, werden nur Files gelistet, deren Update-Flag gesetzt ist.

Bei der Angabe des Parameters /EXT werden nur die Files gelistet, die einen entsprechenden Zusatz haben. So werden beispielsweise beim Kommando DIR,1,/CMD nur Benutzerprogramme mit der Erweiterung /CMD wie EDTASM/CMD angezeigt.

Werden beide Parameter 'U,/EXT' angegeben, werden nur die Files angezeigt, die beide Bedingungen erfuellen.

Wenn der Bildschirm voll ist, erscheint ein '?'. Druecken Sie dann entweder die 'ENTER'-Taste um die Liste fortzusetzen, oder die 'BREAK'-Taste um die Liste abzubrechen.

Beim Parameter 'P' werden alle Ausgaben auf dem Drucker gemacht. Achtung: das System wartet, bis der Drucker auf 'READY' steht.

Bei der Eingabe des Parameters '@' wartet das System, bis die zu listende Floppy im Laufwerk ist. Dieser Parameter ist nur dann sinnvoll, wenn nur ein Laufwerk am System ist und von einer Datendiskette ohne System eine Liste gemacht werden soll.

Wird die Laufwerksnummer nicht angegeben im Kommando, so wird das Laufwerk gelistet, das beim Systemparameter AN angegeben wurde. Dies muss nicht unbedingt Laufwerk 0 sein.

Beispiele fuer DIR:

1. DIR 0

   Die Directoryeintraege aller Files, die keine System-Files sind und die nicht unsichtbar sind, werden mit Zusatz auf dem Bildschirm angezeigt.

2. DIR,0,S,I,P

   Alle Directoryeintraege aller System-Files und aller unsichtbarer von Laufwerk 0 werden auf dem Drucker ausgegeben.

3. DIR 1,/DAT,U

   Die Directoryeintraege aller Files auf Laufwerk 1 mit dem Zusatz /DAT werden angezeigt, wenn das Update-Flag gesetzt ist.

4. DIR 2,A

   Die Directoryeintraege aller Files auf Laufwerk 2, die nicht unsichtbar und die keine System-Files sind, werden mit allen Informationen angezeigt. Es wird ein File pro Zeile gelistet, wenn das Display voll ist, kann man die Liste entweder mit 'ENTER' fortsetzen oder mit 'BREAK' abbrechen.

5. DIR @0

   Die Anzeige erfolgt wie beim ersten Beispiel, nur dass das System die Anweisung gibt, die entsprechende Floppy ins Laufwerk zu legen und danach wieder gegen die System-Floppy auszutauschen.

2.19 DO

Aufruf eines CHAIN-Files.

DO,FNAME1<,Bereichsname>

Das DO-Kommando arbeitet genau gleich wie das CHAIN-Kommado (Beschreibung siehe "CHAIN" auf Seite 10).

2.20 DUMP

DUMP,FNAME1,Startadresse,Endadresse<,Programmstart>

Das Kommando DUMP schreibt den Inhalt des Speichers auf die Diskette unter dem Namen FNAME1. Der Dump beginnt mit der eingegebenen Startadresse und hoert mit der eingegebenen Endadresse auf.

Start-, End- und Programmstartadresse werden in dezimal (Wert kleiner als 65536) oder hexadezimal (Wert kleiner als 10000H) eingegeben. Fuer die hexadezimale Eingabe muss das H angehaengt werden.

Das DUMP-Kommando hat 2 verschiedene Funktionen:

- Wird als Programmstartadresse 65535 (0FFFFH) eingegeben, dann wird ein exakter Speicherabzug auf die Floppy geschrieben. In den ersten 2 Bytes steht dann lediglich die Startadresse. Diesen File kann man mit dem 'SUPERZAP' anzeigen oder ausdrucken (Funktion DMDB) oder auf einem anderen TRS-80 die Fehlersuche durchzufuehren.
- Ist die eingegebene Programmstartadresse kleiner als 65535 (0FFFFH) oder wird sie nicht angegeben, so wird der Speicherinhalt im Loader-Format auf die Floppy geschrieben. Dieser File kann dann spaeter einfach ueber den Loader (siehe "LOAD" auf Seite 32) wieder ins Memory geladen werden.

Achtung: Die Startadresse darf nicht kleiner als 5200H sein, da sonst das DOS zerstoert wird.

Beispiel: DUMP,FILENAME/CMD:1,5200H,89ABH,5700H

Der Speicherinhalt zwischen der Adresse 5200H und 89ABH wird auf die Floppy 1 unter dem Namen FILENAME/CMD abgespeichert und als Startadresse 5700H eingetragen. Ueber das Kommando

LOAD,FILENAME/CMD

kann dieses Programm wieder in den Speicher geladen werden oder ueber das DOS-Kommando

FILENAME<,Parameter>

direkt ausgefuehrt werden. Im zweiten Beispiel ist dargestellt, wie ein Programm bzw. der gesamte Speicherinhalt auf Diskette geschrieben wird im Fehlerfall, wenn man momentan keine Zeit hat den Speicherinhalt zu analysieren. Voraussetzung ist allerdings, dass man entweder im DOS ist oder ueber die Tasten 'DFG' ins MINI-DOS kommt.

DUMP,MEMORY/DMP:1,0,65535,65535

Mit diesem Befehl werden die kompletten 64k, einschliesslich ROM und Display, auf die Floppy 1 in den File MEMORY/DMP geschrieben. Die ersten 2 Bytes dieses Files sind dann 00 00 (Startadresse des Dumps). Im File sind sonst keine weiteren Kontrollzeichen. Zu einem spaeteren Zeitpunkt kann man dann ueber den SUPERZAP (Option 'DMDB') sich diesen Dump ansehen und auswerten. Zu beachten waere noch, dass der DOS- Speicherbereich zwischen 4000H und 51FFH nicht mehr den gleichen Inhalt hat, da 'DUMP' aufgerufen wurde.

2.21 ERROR

Anzeigen von DOS-Fehlermeldungen.

ERROR,xx

Die zugehoerige Fehlermeldung des Fehlers Nr. xx wird angezeigt. xx muss zwischen 0 und 63 liegen.

Beispiel fuer ERROR:

ERROR,24 zeigt den Fehler 'FILE NOT IN DIRECTORY' an.

2.22 FORMAT

Disketten koennen nicht direkt so benutzt werden, wie sie vom Hersteller geliefert werden. Sie muessen zuerst unterteilt werden in Spuren, und jede Spur fuer sich in Sektoren, wobei jeder Sektor in 256 Bytes unterteilt wird. Auf diese Art gehen ca. 15 bis 30 Prozent der Speicherkapazitaet verloren.

Das DOS-Kommando FORMAT nimmt diese Unterteilung vor, ebenso werden von Format die 2 benoetigten System-Files BOOT/SYS und DIR/SYS geschrieben. Sobald dies geschen ist, kann die Floppy normal im NEWDOS/80 eingesetzt werden.

FORMAT kann auch als Teil von COPY ausgefuehrt werden (Format 5 und 6 der COPY-Funktion, siehe dazu "COPY" auf Seite 12).

FORMAT,DN2<=tc2>,NAME2,mm/dd/yy,PASSWORT<,N><,Y><,NDMW><,DDND><,ODN=name1><,KDN><,DDSL=ln1><,DDGA=gc1><,DPDN=dn4><,PFST=tn3><,PFTC=tc3>

In der Version 2 des NEWDOS/80 wird jede Spur sofort gelesen, bevor der Arm auf die naechste Spur geht. Dann, wenn alle Spuren formatiert sind und die Systemoption BM=Y ist, wird die gesamte Floppy nochmals im VERIFY gelesen. Bei BM=N wird die VERIFY-Phase uebersprungen. So kann der Anwender selbst bestimmen, ob eine Floppy nochmals ueberprueft werden soll oder nicht.

Im FORMAT von NEWDOS/80 ist es nicht erlaubt, eine Spur zu sperren, auch dann nicht, wenn eine Spur einen Fehler beim Formatieren hat. Das entsprechende Byte in der Lock-Out-Table wird immer auf 0FFH gesetzt. Die Lock-Out-Table wurde nur deshalb mit initialisiert, damit eine gewisse Austauschbarkeit

zum TRSDOS bestehen bleibt. Die Entwickler von NEWDOS/80 arbeiten ganz einfach nach der Philosophie, dass eine Diskette, wenn sie nicht voll formatiert werden kann, in den Muelleimer gehoert und nicht in einen Rechner, schon gar nicht unter diesem Betriebssystem.

FORMAT benoetigt alle zu benutzenden Parameter bei der Kommandoeingabe, es werden also keine Fragen ausgegeben.

DN2 ist die Laufwerksnummer (0-3), deren Floppy initialisert werden soll. Diese Floppy erhaelt dann den Namen NAME2, ausser dann, wenn der Parameter NDN eingegeben wird, in jedem Falle ist aber NAME2 anzugeben. mm/dd/yy ist das Initialisierungsdatum das die Diskette erhaelt, ausser dann, wenn KDD angegeben wird, in diesem Falle wird das eingegebene Datum ignoriert. PASSWORT ist das neue Passwort der Diskette, es muss den ueblichen Regeln fuer Passwoerter entsprechen.

Wird kein Parameter angegeben, benutzt das FORMAT bestimmte Defaultwerte. Diese Werte sind fuer den Namen NOTNAMED, das Passwort 'PASSWORD' und fuer das Datum das Systemdatum. Jeder dieser 3 Parameter kann auch so benutzt werden, jedoch darf dann anstelle des ausgelassenen Parameters im Kommando kein Leerzeichen stehen, sondern es muessen Kommas benutzt werden.

Da FORMAT und COPY nahezu die gleichen Routinen sind im NEWDOS/80, sind auch die einzugebenden Parameter nahezu die gleichen wie bei COPY 5 und COPY 6. Der einzige Unterschied besteht darin, dass bei COPY 5 und 6 nach dem Formatieren noch das Kopieren stattfindet. Der Benutzer sollte sich also die Beschreibung von COPY 5 und 6 ansehen, da hier bei Format nur die zum COPY unterschiedlichen Parameter beschrieben werden.

'N' wird automatisch angenommen, wenn keiner der Parameter benutzt wird.

Wenn =tc2 angegeben wird, wird die Diskette mit entsprechender Spuranzahl (1-96) formatiert, andernfalls mit dem ueber das Kommando PDRIVE vorgegebenen Wert. Wenn der Wert =tc2 groesser ist als die tatsaechliche Anzahl der Spuren des Laufwerks, kann es sein, dass das Programm beim Suchen der zu hohen Spur haengen bleibt.

PFST=tn3 und PFTC=tc3 sind optionale Parameter, die es erlauben, einzelne Spuren zu formatieren. Wenn PFST angegeben wird, muss nicht unbedingt auch PFTC angegeben werden, wird jedoch PFSC angegeben, muss auch PFST angegeben werden. PFST bedeutet 'Partial Format Starting Track' und tn3 gibt die erste zu formatierende Spur an. PFTC heisst 'Partial Format Track Count', und tc3 gibt die anzahl der fortlaufenden zu formatierenden Spuren an. Wenn PFTC nicht eingegeben wird, wird der Defaultwert 1 angenommen. Wenn tc3 Spuren formatiert wurden, wird entsprechend der System-Option BM=Y die ganze Diskette nochmals ueberprueft. Sollte es dabei Probleme geben, kann das VERIFY mit dem Pfeil nach oben (Up-Arrow) abgebrochen werden, da jede Spur sowieso sofort nach dem Formatieren ueberprueft wird.

PFST darf nicht auftreten mit N, DDND, ODN, KDN, KDD, DDSL, DDGA und RWF.

FORMAT kann nicht unter MINI-DOS aufgerufen werden.

Beispiele fuer FORMAT:

1. FORMAT,0,AAA0,08/01/81,PSWD,Y

   Nach Eingabe dieses Kommandos wird die zu formatierende Diskette in das Laufwerk 0 eingelegt und dann das FORMAT entsprechend den PDRIVE-Parametern durchgefuehrt. DOS nimmt keine Ruecksicht darauf, ob die Diskette bereits Daten enthaelt oder nicht. Die Diskette erhaelt den Namen AAA0, das Datum 08.01.81 und das Passwort PSWD.

2. FORMAT,0,,,,Y

   Dieses Kommando ist im Prinzip identisch mit dem vorgehenden Beispiel, nur dass der Name NONAMED ist, das Passwort PASSWORD und als Datum wird das aktuelle Systemdatum eingesetzt.

3. FORMAT,1,XXX,,PSWD,N,NDMW,DPDN=4,DDSL=40,DDGA=6

   Die Diskette, die bereits im Laufwerk 1 stecken muss, darf keine Daten enthalten. Sie wird entsprechend den PDRIVE-Angaben von Drive 4 formatiert (nicht von Drive 1!). Sie erhaelt den Namen XXX, das Passwort PSWD und das aktuelle Systemdatum als Datum. Das Directory beginnt auf 'Lump' 40 und ist 6 Granules lang (fuer ein Maximum an 222 Eintraegen).

4. FORMAT,1,,,,Y,PFST=22,PFTC=2

   Von der Diskette, die in Drive 1 ist, wurden die Spuren 22 und 23 durch einen Netzausfall zerstoert. Mit dem SUPERZAP wurde festgestellt, dass bei mindestens einem Sektor jeder Spur der Fehler SECTOR NOT FOUND gemeldet. Mit der Funktion CDS oder SCOPY wurden die restlichen Sektoren der 2 Spuren auf andere Bereiche gerettet. Mit diesem Format jetzt kann man die 2 zerstoerten Spuren neu formatieren, der Rest der Diskette bleibt unangetastet. Wenn das getan ist, kann man die restlichen Sektoren aus dem Rettbereich wieder zurueckholen.

2.23 FORMS

Parameter fuer den Drucker setzen (gilt nur fuer Modell III).

FORMS<,WIDTH=xxx><,LINES=yyy>

Das Kommando FORMS kann einige Parameter des Printers aendern und diese Parameter ausdrucken.

WIDTH=xxx gibt die Anzahl der Zeichen je Zeile an. xxx muss eine Zahl zwischen 9 und 255 sein. Wenn WIDTH nicht angegeben wird, bleibt die Anzahl der Zeichen je Zeile unveraendert.

LINES=yyy gibt die Anzahl der Zeilen je Seite an. yyy muss eine Zahl zwischen 1 und 254 sein, wobei 254 bedeutet, dass kein Vorschub stattfindet. Wenn LINES nicht angegeben wird, bleibt die Anzahl der Zeilen je Seite unveraendert.

Beispiele fuer FORMS:

1. FORMS,WIDTH=80,LINES=60

   Der Drucker wird auf 80 Zeichen je Zeile und 60 Zeilen je Seite gesetzt.

2. FORMS,WIDTH=255,LINES=254

   Unbegrenzte Zeichen je Zeile und Zeilen je Seite.

3. FORMS

   Die momentanen Werte werden angezeigt.

2.24 FREE

FREE<,P>

Der freie Platz auf einer Diskette und die Anzahl der freien Eintraege wird angezeigt.

Fuer jede angeschlossene Diskette wird die Diskettennummer, der -name, das -datum, die Anzahl der Spuren, Anzahl der freien Eintraege und Anzahl der freien Granules angezeigt.

Wird FREE mit dem Parameter ,P eingegeben, werden die gleichen Angaben auf dem Drucker gelistet.

Beispiele fuer FREE:

1. FREE

   Die Anzahl der freien Eintraege, Diskettenname, -datum etc. aller angeschlossenen Laufwerke mit Diskette werden angezeigt.

2. FREE,P

   Gleiche Ausgabe wie in letzten Beispiel, nur auf den Drucker.

2.25 HIMEM

HIMEM<,adr1>

Die obere Grenze des freien Speicherplatzes wird von der Adresse 4049H und 404AH angezeigt (Wert zwischen 7000H und 0FFFFH).

Bei Eingabe des Parameters adr1 wird die obere Grenze des freien Speicherplatzes auf der Adresse 4049H und 404AH abgespeichert (Wert zwischen 28672 und 65535 bzw. 7000H und 0FFFFH). Diese Grenze wird bei den Routinen bzw. Programmen COPY (Option UBB), BASIC, EDTASM, DISSASEM und LMOFFSET als obere Grenze benutzt.

Achtung! Der Loader ignoriert diese Grenze beim Laden !

2.26 JKL

Der momentane Bildschirminhalt wird ausgedruckt.

JKL hat keine Parameter. Das Kommando JKL benutzt die gleiche Routine wie die Funktion 'JKL' (alle 3 Tasten gleichzeitig gedrueckt).

Bei der Systemoption AK=Y wird der Hex-Code > 80H (graphische Zeichen) auch auf den Drucker ausgegeben, bei AK=N wird dieser Code in einen Punkt umgewandelt. Grundsaetzlich umgewandelt wird der Code < 20H.

JKL kann angewandt werden z. B. in BASIC als Aufruf CMD"JKL", oder in einem Anwenderprogramm, um den Bildschirminhalt auf den Drucker auszugeben.

2.27 KILL

KILL,FNAME

File loeschen

Der File mit dem Namen FNAME wird von der Floppy geloescht. Wenn keine Laufwerksnummer angegeben ist, beginnt das System bei Laufwerk 0 und sucht so lange, bis es einen File mit dem Namen FNAME gefunden hat.

Der Platz, den der File bis dahin belegt hat, wird freigegeben zur Benutzung durch andere Files. Das Kommando KILL macht beim Aufruf folgendes:

1. War fuer den zu loeschenden File Platz auf der Diskette reserviert, so wird dieser Platz freigegeben fuer die Belegung durch andere Files. Der bisherige Inhalt des Files wird dadurch jedoch nicht veraendert. Der Inhalt wird erst dann veraendert, wenn dieser Platz durch einen neuen File belegt wird.

2. Der FPDE und alle vorhandenen FXDE's werden freigegeben, indem das 4. Bit des ersten Bytes auf 0 gesetzt wird. Ebenfalls werden alle zugehoerigen Eintrage in der HIT-Table geloescht. sonst wird nichts veraendert, bis dieser Directory-Eintrag einem anderen File zugeordnet wird.

Haben Sie versehentlich einen File geloescht, den Sie besser nur gerettet haetten, koennen Sie ihn, solange dieser Eintrag im Directory und der Platz mit den Daten noch nicht belegt ist, den File wieder herstellen. Sie muessen sich aber genau mit der Funktion des Directorys und der Bedeutung von FPDE und FXDE auskennen. Rufen Sie wegen solcher Trivialitaeten unter keinen Umstaenden bei Apparat an (duerfte fuer die deutschen Benutzer sowieso nicht in Frage kommen).

Falls mehrere Files einer Diskette zu loeschen sind, sollte man sinnvollerweise das Kommando 'PURGE' benutzen (siehe "PURGE" auf Seite 42).

Beispiele fuer KILL:

1. KILL XXX/BAS:1

   Der File XXX/BAS auf der Floppy in Laufwerk 1 wird geloescht.

2. KILL YYY

   Der File YYY wird auf den Floppies gesucht, beginnend bei Laufwerk 0, und, sobald er gefunden wird, geloescht. Wenn auch eine der nachfolgenden Floppies den gleichen File hat, wird dieser dann nicht geloescht.

2.28 LC

Tastaturabfrage setzen

LC<,yn>

Bei LC,Y oder nur LC werden von der Tastaturabfrage die Kleinbuchstaben a - z ohne Konvertierung akzeptiert.

Bei LC,N werden die Kleinbuchstaben a - z in Grossbuchstaben A - Z umgewandelt.

Dieses Kommando hat keinen Einfluss, solange im Modell I keine Kleinschreibung eingebaut ist (siehe "LCDVR").

2.29 LCDVR

(gilt nur fuer Modell I) Treiber fuer Kleinschreibung

LCDVR<,x(,s)>

In der Version 1 des NEWDOS/80 war der Treiber fuer die Kleinschreibung LCDVR noch ein separates Programm, das im oberen Teil des Memory ausgefuehrt wurde. Hier in der Version 2 ist LCDVR Bestandteil des DOS.

Bei x=Y wird der Treiber aktiviert, bei x=N wird er inaktiv. Wenn der Treiber aktiv ist, werden

1. die von der Tastatur kommenden Zeichen entsprechend dem Kommando LC behandelt und angezeigt.
2. der ASCII-Code 96 - 127 (60H - 7FH) richtig angezeigt und nicht konvertiert in 64 - 95 (40H - 5FH, A - Z) von der ROM-Routine.

Der 2. eingegebene Parameter hat nur dann eine Bedeutung, wenn x=Y eingegeben wurde. Er hat dann die gleiche Funktion wie der erste Parameter von LC und setzt das Kennbit fuer die Tastatur.

Wenn der Treiber fuer Kleinbuchstaben aktiviert ist, kann man ueber SHIFT 0 umschalten von nicht konvertiert auf konvertiert (wie bei der Version 1 mit SHIFT Blank). Das Kommando LC kann weiterhin benutzt werden fuer das Setzen des Kennbits auf einen definierten Status.

Werden bei diesem Kommando keine Parameter eingegeben, wird LCDVR.Y,N angenommen.

Beispiele fuer LCDVR:

1. LCDVR

   Die Kleinschreibroutine wird aktiviert und die Kennung ist gesetzt auf die Konvertierung von a - z auf A - Z.

2. LCDVR,Y,Y

   Die Kleinschreibroutine wird aktiviert und die Kennung ist gesetzt auf keine Konvertierung von a - z auf A - Z. Kleinbuchstaben werden angezeigt.

3. LCDVR,N

   Die Kleinschreibroutine wird ausgeschaltet.

2.30 LIB

LIB-alle NEWDOS/80 Befehle anzeigen.

Dieses Kommando wird ohne Parameter aufgerufen. Es werden alle in NEWDOS/80 vorkommenden Kommandos angezeigt.

2.31 LIST

LIST,FNAME<,STARTZEILE(ZEILENANZAHL)>

Mit diesem Kommando wird der File mit dem Namen FNAME auf dem Bildschirm angezeigt. Die ausgegebenen Character werden nicht ueberprueft, es koennen also auch Steuerzeichen sein.

Wenn eine Startzeile angegeben wird (zwischen 1 und 65535), beginnt die Liste bei dieser Zeile.

Beispiele fuer anzeigbare Textfiles sind BASIC-Programme, die mit der Option ,A abgespeichert wurden, BASIC-Files, die mit dem PRINT-Statement erzeugt wurde, Assembler-, Fortran- oder Cobol-Source-Files, Scripsit- und Electric-Pencil-Files.

Die gelisteten Zeichen werden nicht ueberprueft, ob sie einen auf dem Bildschirm darstellbaren Code haben, ausser den Zeichen, die zwischen 80H und 0FFH liegen, die je nach der Systemoption AX konvertiert werden.

Wenn eine Startzeile angegeben wird (Wert zwischen 1 und 65535), beginnt die Liste mit dieser Zeile.

Die Liste kann durch Druecken der Taste Right Arrow (Pfeil nach rechts) beim naechsten Character 0DH (Return) oder beim naechsten Blockende nach 256 Bytes unterbrochen werden. Mit der ENTER-Taste wird die Liste fortgesetzt. Durch Druecken der Taste Up Arrow (Pfeil nach oben) wird das Listen abgebrochen.

Ein kleiner Hinweis noch am Rande: manchmal kann es hilfreich sein, wenn man in einem Listfile Datum und Uhrzeit am Anfang angibt, da man mit dem LIST-Kommando sofort festellen kann, ob es sich um die neueste Liste handelt.

Um andere Files als Text-Files anzuzeigen benutzt man sinnvollerweise den 'SUPERZAP'.

Beispiele fuer LIST:

1. LIST,BASEPROG/BAS

   Listet das Programm BASEPROG/BAS auf dem Bildschirm

2. LIST,XXX,1,6

   Listet die ersten 6 Zeilen des Files XXX.

3. LIST,YYY:1,200

   Listet den File YYY ab der Zeile 200.

2.32 LOAD

LOAD,FNAME

Maschinenprogramm in das RAM laden

Mit diesem Kommando lassen sich Z80-Maschinenprogramme in den Speicher laden. Die Startadresse dieses Programms wird in der Adresse 4403H (17411 dez.) abgespeichert. Das Maschinenprogramm muss auf der Diskette im korrekten Format, wie es z. B. von 'DUMP' oder 'EDTASM' erzeugt wird, abgespeichert sein. Der Loader kontrolliert nicht die Startadresse des Programms, es darf also nicht im residenten DOS-Teil (4000H bis 4CFFH) oder der Overlay-area (4D00H bis 51FFH) beginnen, da sonst Teile des Betriebssystem zerstoert werden.

Der Befehl LOAD wird benutzt, wenn man ein Programm oder Daten fuer spaeteren Gebrauch laden will. Man kann z. B. ein solches

Programm in BASIC mit der Funktion 'USR' aufrufen. Die Startadresse steht, wie oben beschrieben, in 4403H bzw. 17411 dez. (nicht mit im TRSDOS). **Beispiele fuer LOAD:**

1. LOAD,OVERLAY/OBJ:1

   Das Object-Modul OVERLAY/OBJ wird von der Diskette in Laufwerk 1 in den Speicher geladen. In dem Kontrollblock des Object-Moduls stehen die Angaben, wo es im Speicher stehen soll.

2. Angenommen, dass ein geladenes BASIC-Programm nicht den gesamten Speicher benutzt und dieses BASIC-Programm moechte das Object-Modul USR3PGM/OBJ in den oberen Teil des Memorys laden zum spaeteren Ansprung als USR3-Funktion, kann man im BASIC-Programm folgendes schreiben:

   ```
   CMD"LOAD,USR3PGM/OBJ"
   DEFUSR3 = (PEEK(17411) + 256 * PEEK(17412) - 65636
   ```

2.33 MDBORT

MINI-DOS abbrechen

Dieses Kommando hat keinen Parameter. Es sollte nur dann benutzt werden, wenn man im MINI-DOS arbeitet. Der MINI-DOS Status wird geloescht und das System kehrt in das DOS zurueck.

Das Kommando kann dann angewendet werden, wenn man nicht in das vorhergehende Programm, von dem aus MINI-DOS durch Druecken der 3 Tasten 'DFG' aufgerufen wurde, zurueckkehren will.

2.34 MDCOPY

MDCOPY,FNAME1<,TO>,FNAME2

Das normale COPY-Kommando kann nicht im MINI-DOS aufgerufen werden, da es im Bereich ab 5200H arbeitet. Im MINI-DOS muss also das MDCOPY benutzt werden, es benutzt einen Buffer von nur 256 Bytes (1 Sektor).

MDCOPY kopiert den File mit der Bezeichnung FNAME1 in den File FNAME2. Dabei kann es sich um einen neuen oder bereits angelegten File handeln. Der File FNAME1 wird nicht veraendert. Die Bezeichnung des Files FNAME2 muss komplett angegeben werden, man kann also nicht die gleichen Abkuerzungen wie in 'COPY' benutzen.

2.35 MDRET

Ruecksprung von MINI-DOS in das aufrufende Programm

MDRET wird ohne Parameter aufgerufen. Das Betriebssystem verlaesst den MINI-DOS Status und kehrt in das Programm zurueck, von dem aus es durch gleichzeitiges Druecken der Tasten 'DFG' aufgerufen wurde. Der Ruecksprung erfolgt genau auf die Stelle, von der aus das laufende Programm unterbrochen wurde. Wurde also eine Eingabe z. B. nur teilweise gemacht oder noch nicht mit ENTER abgeschlossen, so ist diese Eingabe noch vorhanden, auch wenn sie nicht mehr am Bildschirm angezeigt wird, weil das MINI-DOS vielleicht den Bildschirm geloescht hat.

Das MINI-DOS wird vom 25-ms-Interrupt angestossen, es kann also vorkommen, dass dadurch mehrere 'DFG' auf dem Bildschirm

angezeigt werden, bis MINI-DOS angestossen ist. Der Benutzer kann diese Character aber mit 'Backspace' loeschen.

2.36 PAUSE

Meldung auf dem Bildschirm ausgeben und warten, bis die ENTER-Taste gedrueckt wird.

PAUSE,msg

Die Meldung msg wird nicht angezeigt, wenn das Kommando selbst bereits angezeigt wird, also z. B. im DOS. Wenn das Kommando aber nicht angezeigt wird, z. B. bei einem Aufruf als DOS-CALL, wird die Meldung msg angezeigt und in der naechsten Zeile erscheint PRESS "ENTER" WHEN READY TO CONTINUE. Dieses Kommando ist eines von insgesamt 4, mit denen das Chaining unterbrochen werden kann. Es kann ebenfalls benutzt werden, wenn mehrere Kommandos ueber DOS-CALL hintereinander ausgefuehrt werden sollen.

Beispiel fuer PAUSE:

PAUSE,DISKETTE "XXXX" IN LAUFWERK 1 EINLEGEN.

Diese Meldung wird auf dem Bildschirm erscheinen, in der naechsten Zeile steht dann PRESS "ENTER" WHEN READY TO CONTINUE. DOS wartet, bis die ENTER-Taste gedrueckt wird nachdem die besagte Diskette eingelegt ist. DOS checkt aber nicht, ob dies wirklich geschehen ist, sondern wartet nur auf die ENTER-Taste.

2.37 PDRIVE

Standardwerte fuer die Floppy-Laufwerke eingeben

PDRIVE,<passwort1:>dn1,<dn2(=dn3)><,TI=typ1><,TD=typ2>
<,TC=tc1><,SPT=sc1><,TSR=rc1><,GPL=gc2>
<,DDSL=ln1><,DDGA=gc1><,A>

PDRIVE kann nicht im MINI-DOS aufgerufen werden.

NEWDOS/80 hat eine begrenzte Anzahl von Moeglichkeiten, verschiedene 5-Zoll Floppies und 8-Zoll Floppies anzusprechen. Ueber das Kommando PDRIVE koennen fuer jede Floppy die physikalischen Eigenschaften angegeben werden.

Jedes PDRIVE-Kommando listet die Angaben fuer max. 10 Laufwerke, obwohl die angeschlossene Anzahl der Laufwerke und die beim SYSTEM-Parameter AL angegebene Anzahl der Laufwerke 4 auf keinen Fall ueberschreiten darf. Die Laufwerke innerhalb des Bereichs von AL werden beim Kommando PDRIVE werden mit einem Stern nach der Laufwerksnummer angezeigt. Die Angaben ueber die 10 Laufwerke werden vom System von der Diskette in Laufwerk 0 uebernommen. Aus Gruenden der Zeitersparnis werden die Laufwerksangaben von den unter AL angegebenen Laufwerken von Diskette 0 im Speicher gehalten und bei jedem Reset neu in den Speicher geholt. Ist jedoch eine der 10 moeglichen Angaben falsch, kann auch die Uebernahme der 4 gueltigen Angaben nicht erfolgen, das System bleibt beim Reset haengen. Die Tabelle wird ebenfalls beim PDRIVE-Kommando mit dem Parameter ,A uebernommen.

Die Angabe :dn1 bezieht auf das Floppy-Laufwerk, in dem sich die Floppy mit dem Betriebssystem befindet. Im 3. Sektor von BOOT/SYS dieser Floppy werden die Angaben ueber die einzelnen

Laufwerke abgespeichert und mit dem Kommando PDRIVE veraendert.

- Wird beispielsweise das Kommando PDRIVE,1,4,TC=80 eingegeben, so erhaelt die Diskette in Laufwerk 1 die Kontrollinformation fuer Laufwerk 5 mit 80 Spuren. Da dn einen Wert von 0 bis 9 annehmen kann, bedeutet 4 das Laufwerk 5. Die Laufwerksparameter der anderen Laufwerke werden nicht beeinflusst.

Wenn die Passwoerter aktiv sind muss das Haupt-Passwort fuer die Diskette eingegeben werden, auf der die Parameter veraendert werden (dn1). Andernfalls kann das Passwort weggelassen werden.

Es werden nur die Parameter eines Laufwerks veraendert, die auch mit dem Kommando angegeben werden. Da das System die eingegebenen Parameter nicht ueberprueft, muss bei diesem Kommando besonders auf die Richtigkeit der Eingaben geachtet werden.

Mit dem Kommando PDRIVE,dn1 werden die 10 Laufwerksparameter gelistet, die auf der Diskette in Laufwerk dn1 gespeichert sind.

TI=typ1 Gibt die Art der Anschaltung zwischen Z80-CPU und der Floppy an. typ1 muss ein einzelner Buchstabe sein und zwar bei Modell I nur A, B, C oder E. Bei Modell III darf es nur A oder D sein. Die anderen Angaben sind optional, abhaengig von der Art des Interfaces zu den Floppies. Manche Angaben duerfen nicht im Zusammenhang mit anderen Angaben weiterer Laufwerke stehen. Wird also beispielsweise fuer Drive 1 eine bestimmte Anschaltung eingegeben, darf Drive 2 nicht eine andere Anschaltungsart enthalten. Im Moment sind fuer das Modell I die Angaben B, C und E entweder fuer alle oder fuer kein Laufwerk anzugeben.

A bedeutet, dass das Standard-Interface fuer Diskettenoperationen installiert ist. Beim Modell I kann dieses Interface die Optionen A und C unterstuetzen, bei Modell III die Optionen A, C, E und G.

B (nur Modell I) Omikron 8-Zoll Floppy-Interface. Dieses Interface unterstuetzt die Optionen A, B, C und D. auch wenn es sich um ein Omikron-Interface handelt.

C (nur Modell I) PERCOM 'Doubler' - Interface ist installiert und wird fuer Floppy ein- und Ausgaben benutzt. Mit diesem Interface koennen die Optionen A, C, E und G benutzt werden.

D (nur Modell III) bedeutet, dass der Disk-Controller von Apparat installiert ist und fuer Floppy Ein- und Ausgabe benutzt wird. Die Optionen A bis H koennen benutzt werden.

E (nur Modell I) bedeutet, dass der LNW-Disk-Controller installiert ist und fuer Floppy Ein- und Ausgabe benutzt wird. Die Optionen A bis H koennen benutzt werden.

H bedeutet, dass beim Wechsel von einer Diskette auf diese mit H angegebene Diskette eine Zeitverzoegerung fuer die Kopfpositionierung eingehalten wird. Bei 5"-Floppies fuer Modell I und III wird der

Kopf mit dem Motorstart neu positioniert, eine Zeitverzoegerung ist also nicht noetig. Sie ist nur fuer Floppies mit 8".

I bedeutet, dass der niedrigste Sektor die Nummer 1 hat. Dies ist der Standard fuer Disketten von Modell III TRSDOS. Wenn I nicht angegeben ist, ist der 1. Sektor einer Spur Sektor 0. Dies ist Standard bei Modell I und beim NEWDOS/80 im Modell III.

J bedeutet, dass der niedrigste Sektor die Nummer 0 hat. Dies ist der Standard fuer Disketten von Modell I und III im NEWDOS/80.

K bedeutet, dass die Spur 0 formatiert wird (oder ist) mit einer anderen Schreibdichte als die anderen Spuren. Dadurch wird die Spur 0 fuer normale Disk Ein- und Ausgabe unbrauchbar. Es kann nur entweder J oder K gesetzt werden. Der Zweck fuer die unterschiedliche Formatierung liegt darin, dass ein System von einer solchen Diskette geladen werden kann (Spur 0 einfache Schreibdichte bei Modell I, alle anderen Spuren doppelte Schreibdichte, bei Modell III die erste Spur doppelte Schreibdichte, alle anderen Spuren einfache). Mit der Flag K teilt man dem FORMAT und dem COPY mit, die Spur 0 in umgekehrter Schreibdichte wie die anderen Spuren zu formatieren oder zu kopieren. Flag K darf nicht angegeben werden fuer Disketten von Modell III, ausser der Benutzer will unbedingt die Standard-Diskette von Modell III auf Modell I ohne NEWDOS/80 Version 2 laufen lassen. Wenn K dann angegeben wird, muss bei TC eine Spur weniger als eigentlich vorhanden angegeben werden.

L bedeutet, dass von einer Spur zur naechsten zwei Impulse gegeben werden. Damit koennen Disketten mit 35 oder 40 Spuren auf einem Laufwerk mit 80 Spuren gelesen werden. Es kann zwar auch damit geschrieben werden, dies kann aber beim Lesen zu Problemen fuehren und sollte deshalb vermieden werden.

M bedeutet, dass die Floppy im Standard Modell III TRSDOS beschrieben ist. Nur das DOS-Kommando COPY wird aber beruecksichtigen, dass die Floppy in diesem Format beschrieben ist und darin von der NEWDOS/80-Diskette unterschiedlich ist.

Wurde mit Aenderung #030 erweitert auf Modell I TRSDOS 2.3B.

Die Optionen F und G sowie N bis Z werden nicht gebraucht und sind reserviert fuer spaetere Anwendung.

TD=typ2 Gibt den Typ der Floppy an. typ2 muss ein einzelner Buchstabe sein nach folgendem Schema: Schema:

1. A = 5" Floppy, einfache Dichte, einseitig
2. B = 8" Floppy, einfache Dichte, einseitig
3. C = 5" Floppy, einfache Dichte, doppeltseitig
4. D = 8" Floppy, einfache Dichte, doppeltseitig

5. E = 5" Floppy, doppelte Dichte, einseitig
6. F = 8" Floppy, doppelte Dichte, einseitig
7. G = 5" Floppy, doppelte Dichte, doppeltseitig
8. H = 8" Floppy, doppelte Dichte, doppeltseitig

Wenn das Modul installiert ist, das die Geschwindigkeit der CPU erhoeht und diese Verschnellerung waehrend der Ein- und Ausgabe auf die Floppy aufgehoben wird, muss darauf geachtet werden, dass die Ein-/Ausgabegeschwindigkeit nicht niedriger wird als die Geschwindigkeit der nicht umgebauten CPU. Die Disk Ein- und Ausgabeloops im NEWDOS/80 koennen keine unterschiedliche Geschwindigkeit tolerieren. Dies gilt besonders fuer die Optionen B, D, E und G. Mit der SYSTEM-Option BJ kann die CPU jedoch mit der schnelleren Geschwindigkeit laufen, auch bei Disk Ein- und Ausgabe. Diese SYSTEM-Option wurde von Apparat jedoch noch nicht genuegend ausgetestet, eine Garantie fuer das Funktionieren kann also nicht gegeben werden.

Die Optionen TD=F oder H verlangen jedoch den Umbau der CPU auf eine hoehere Geschwindigkeit, sie muss bei der Ein- und Ausgabe von der Floppy mindestens doppelt so hoch sein.

Bei Laufwerken vom Typ C, D, G und H behandelt das NEWDOS/80 beim normalen Interface (TI=A, B, C, D oder E) eine doppeltseitig beschriebene Floppy als eine gesamte Floppy. Die niedrigen Sektoren einer Spur liegen auf der ersten Seite, die hoeheren Sektoren auf der Rueckseite. Eine solche Floppy hat auch nur ein Directory. Der Pin 32 der Floppy wird fuer den Seitenwechsel angesteuert (Spezialkabel). Doppeltseitige Floppies mit 5" und 8" koennen beim Modell I und III benutzt werden mit dem NEWDOS/80 Version 2.

Apparat hat in der Version 1 des NEWDOS/80 Floppies mit doppelter Schreibdichte nicht unterstuetzt. Einer der Hauptgruende war der, dass Laufwerke mit doppelter Schreibdichte bei weitem nicht die Datensicherheit erreichen wie mit einfacher Schreibdichte. Einer der Gruende hierfuer war der Data-Separator. Obwohl sich das im letzten Jahr etwas gebessert hat, ist die Datensicherheit dieser Floppies noch wesentlich geringer.

TC=tc1 Gibt die Anzahl der Spuren fuer die Diskette in Drive dn2 an (ausschliesslich Spur 0 bei der Option K). Wenn das Flag K nicht gesetzt ist, gilt TC=35 fuer ein Laufwerk mit 35 Spuren, TC=40 fuer 40 Spuren. Wenn K gesetzt ist, gilt TC=34 fuer Floppies mit 35 Spuren und TC=39 fuer 40 Spuren.

SPT=sc1 Gibt die Anzahl der Sektoren je Spur an. sc1 muss bei doppeltseitigen Laufwerken (TD=C, D, G oder H) doppelt so hoch sein wie bei gleichen Laufwerken, die nur einseitig sind. sc1 kann jeden Wert zwischen 1 und der maximalen Anzahl von 256 Byte langen Sektoren einer Spur sein. Hier noch die Maximalwerte fuer die verschiedenen Laufwerkstypen: A=10, B=17, C=20, D=34, E=18, F=26, G=36 und H=52.

TSR=rc1 Gibt die Zeit der Impulse fuer die Kopfpositionierung an. rc1 kann einen Wert von 0 bis 3 haben. Er wird benutzt bei den Kontroller-Kommandos SEEK, STEP und RESTORE. Beim Standard-Kontroller von Modell I und III ergibt TSR=0 eine Wartezeit von 5 ms, TSR=1 ergibt 10 ms, mit TSR=2 werden 20 ms gewartet und bei TSR=4 40 ms. Beim Modell I war TSR=3 der Standardwert, benutzen kann man auch TSR=2 oder TSR=1, je nach Fabrikat des Laufwerks. Bei Modell

III wird TSR=0 benutzt. Bei Problemen mit der Floppy ist TSR=3 der sicherste Wert.

GPL=gc2 Gibt die Anzahl der Granules pro 'Lump' an. gc2 darf zwischen 2 und 8 liegen. Beim TRSDOS fuer Modell I und III, sowie bei den aelteren NEWDOS-Versionen wurde die Unterteilung der Floppy ueber Granules (5 Sektoren pro Granule beim Modell I und 3 beim Modell III) und Spuren (2 Granules pro Spur bei Modell I 6 Granules pro Spur bei Modell III) vorgenommen. Beim NEWDOS/80 Version 2 fuer beide Modelle werden noch immer 5 Sektoren pro Granule erzeugt, und 2 bis 8 Granules pro 'Lump' (nicht Spur). Wenn eine Spurnummer im Directory steht (im GAT-Sektor oder im FXDE) wird diese ersetzt durch eine 'Lump'-Nummer. Auf diese Art kann ein Granule in einer Spur anfangen und in der naechsten Spur aufhoeren. So kann bei einer 8"-Diskette mit doppelter Schreibdichte die Anzahl der Sektoren maximiert werden unter Beibehaltung des bisherigen Formats des Directorys. Bei GPL=2 ist das Directory gleich aufgebaut wie beim TRSDOS und allen anderen NEWDOS/80 Versionen. Floppies und einzelne Files koennen dann zwischen den einzelnen Betriebssystemen hin- und hertransferiert werden. Bei GPL=8 kann das Directory jetzt 192 * 8 * 5 = 7680 Sektoren oder 1966000 Bytes behandeln.

DDSL=lnl entspricht in seiner Funktion dem DDST-Parameter von NEWDOS/80 Version 1. lnl gibt die 'Lump'-Nummer an, in der der erste Sektor des Directorys steht. Dieser Wert wird im 3. Byte des BOOT-Sektors abgespeichert beim FORMAT. In den aelteren Betriebssystemen enthielt dieses 3. Byte die Spurnummer, wo das Directory anfaengt. Da aber bei der Version 2 nur noch der Ausdruck 'Lump' anstelle von Spur benutzt wird, steht hier eben die 'Lump'-Nummer. Um die relative Sektornummer des ersten Sektors des Directory zu erhalten, greift das System auf das 3. Byte des BOOT-Sektors zu und multipliziert es mit 5 * GPL. DDSL=17 ergibt die gleiche Startadresse bei der einseitigen, mit einfacher Dichte beschriebenen Floppy wie bei aelteren Betriebssystemen. DDSL sollte auf den gleichen Wert gesetzt werden wie der Parameter DDST bei Version 1.

DDGA=gc1 (Wert zwischen 2 und 6) gibt die Anzahl der Granules an, die fuer das Directory reserviert werden bei FORMAT oder beim COPY, falls der Parameter DDGA bei diesen Kommandos nicht mit eingegeben wird. Der Standardwert von DDGA ist fuer die Verwendung mit anderen Betriebssystemen ausser dem NEWDOS/80 = 2. Wenn dieser Wert ungleich 2 ist, enthaelt das 32. Byte im HIT-Sektor die Anzahl der zusaetzlichen Sektoren fuer das Directory (5, 10, 15 oder 20). Jeder dieser zusaetzlichen Sektoren enthaelt 8 zusaetzliche Eintraege. So kann das Directory bei den Werten DDGA=2, 3, 4, 5 und 6 entsprechend 64, 104, 144, 184 und 224 Eintraege enthalten. Es koennen alle Eintraege ausser 2 (BOOT/SYS und DIR/SYS) benutzt werden.

gibt an, dass die Eingaben von PDRIVE (allerdings nur, wenn sie fehlerfrei sind) sofort auf das Laufwerk geladen werden und auch aktiv werden, das im SYSTEM-Parameter AL angegeben ist. Dadurch entfaellt nach einer Eingabe das Reset. Wenn A angegeben wird, muss dnl = 0 sein.

Beispiele fuer PDRIVE:

Die Kommandos in den Beispielen werden aus Formatierungsgruenden in 2 Zeilen geschrieben, Sie sollten diese aber am Stueck, also einzeilig eingeben.

1. PDRIVE,dn1,dn2,TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,
DDSL=17,DDGA=2

Dies ist die Standardeingabe fuer Disketten, die 5" haben, 35 Spuren, einfache Schreibdichte und fuer den Austausch mit Version 1 und anderen DOS-Versionen wie TRSDOS geeignet sind beim Modell I. Diese Angabe kann ebenfalls beim Modell III benutzt werden.

Fuer Disketten mit TRSDOS 2.3B im Modell I benutzen Sie TI=AM (siehe Aenderung #030).

2. PDRIVE,dn1,dn2,TI=A,TD=E,TC=40,SPT=18,TSR=3,GPL=2,
DDSL=17,DDGA=2

Dies ist die Standardeingabe fuer Disketten, die 5" haben, 40 Spuren, doppelte Schreibdichte und doppeltseitig, also im Modell III eingesetzt werden. Bei Modell I gibt man anstelle von TI=A den Wert TI=C an. Mit dieser PDRIVE-Angabe kann man die Diskette ebenfalls in Modell I mit Laufwerken doppelter Schreibdichte verarbeiten.

3. PDRIVE,dn1,dn2,TI=AM,TD=E,TC=40,SPT=18,TSR=3,GPL=6,
DDSL=17,DDGA=2

Diese Eingabe kann beim Modell III gemacht werden (Modell I muss TI=CM oder EM haben), um Disketten mit TRSDOS zu lesen oder zu schreiben. Die Diskette hat 40 Spuren, doppelte Schreibdichte und ist 5" einseitig. Dies ist das einzige TRSDOS-Format, das auch mit NEWDOS/80 zu verarbeiten ist. Da die Diskette von Modell III im NEWDOS nicht formatiert werden koennen, ist die Angabe von DDSL und DDGA ohne Bedeutung. GPL=6 muss angegeben werden. Im NEWDOS/80 kann nur das COPY-Kommando mit TRSDOS-Disketten von Modell III benutzt werden, ausser dem SUPERZAP mit den Funktionen DD, DM, DTS, VDS, CDS, CDD usw., nicht aber Funktionen, die sich auf einen File beziehen (z. B. DFS darf nicht benutzt werden).

4. PDRIVE,dn1,dn2,TI=A,TD=C,TC=80,SPT=20,TSR=2,GPL=8,
DDSL=20,DDGA=6

Diese Eingabe behandelt eine 5"-Floppy mit 80 Spuren, einfacher Schreibdichte, doppeltseitig, mit einer Wartezeit von 20 ms, 8 Granules pro 'Lump' und groesstmoeglichem Directory in der Mitte der Floppy. Bei Modell III ist diese Eingabe verboten, da keine Floppies mit einfacher Schreibdichte in Laufwerk 0 sein duerfen.

5. PDRIVE,dn1,dn2,TI=A,TD=G,TC=80,SPT=20,TSR=2,GPL=8,
DDSL=35,DDGA=6

Diese Eingabe behandelt eine 5"-Floppy mit 80 Spuren fuer Modell III (bei Modell I TI=C oder E benutzen) mit doppelter Schreibdichte, doppeltseitig, mit 20 ms Wartezeit von Spur zu Spur, 8 Granules pro 'Lump' und groesstmoeglichem Directory in der Mitte der Floppy. Bei Modell I ist diese Eingabe verboten, da keine Floppies mit doppelter Schreibdichte in Laufwerk 0 sein duerfen.

6. PDRIVE,dn1,dn2,TI=CK,TD=E,TC=39,SPT=18,TSR=3,GPL=2,
DDSL=17,DDGA=2

Diese Eingabe behandelt eine 5"-Floppy mit 40 Spuren fuer Modell I (bei Modell III TI=AK benutzen) mit doppelter Schreibdichte und einseitig beschrieben. Bei dieser Floppy ist die Spur 0 formatiert mit einfacher Schreibdichte, folglich sind nur 39 Spuren fuer den normalen Gebrauch zugelassen. Mit dieser Angabe koennen Disketten verarbeitet werden, die mit dem PERCOM-Doubler unter NEWDOS/80 Version 1 und TRSDOS erzeugt wurden. Ebenfalls kann diese Angabe benutzt werden, wenn eine Diskette mit doppelter Schreibdichte im Modell I auf dem Laufwerk 0 benutzt wird. Beim Interface LNW bei Modell I benutzt man die Angabe TI=EK.

7. PDRIVE,dn1,dn2,TI=CK,TD=G,TC=79,SPT=36,TSR=3,GPL=8,
DDSL=35,DDGA=6

Diese Eingaben koennen gemacht werden beim Modell I (bei Modell III benutzen Sie TI=AK) fuer eine 5"-Diskette mit 80 Spuren, doppelter Schreibdichte und doppeltseitig, deren erste Spur mit einfacher Dichte beschrieben ist. Mit einem LNW-Interface setzt man beim Modell I TI=EK ein.

Achtung

Doppelseitige Disketten mit doppelter Schreibdichte, die unter der geaenderten Version 1 von NEWDOS/80 beschrieben wurden, koennen nicht in der Version 2 laufen.

8. PDRIVE,dn1,dn2,TI=AL,TD=A,TC=35,SPT=10,TSR=3,GPL=2,
DDSL=17,DDGA=2

Diese Eingaben koennen gemacht werden bei 5"-Floppies mit 35 Spuren, einseitig mit einfacher Schreibdichte um sie mit einem Laufwerk mit 80 Spuren lesen zu koennen. Das Laufwerk mit 80 Spuren hat nur die halbe Schrittweite von Spur zu Spur. Mit der Angabe TI=AL bewirkt man, dass von Spur zu Spur 2 Schritte zur Positionierung gemacht werden.

9. PDRIVE,dn1,dn2,TI=BH,TD=B,TC=77,SPT=17,TSR=3,GPL=3,
DDSL=17,DDGA=6

Mit dieser Eingabe fuer Modell I kann man 8"-Floppies mit 77 Spuren, einseitig und mit einfacher Dichte verarbeiten. Beachten Sie, dass in der NEWDOS/80 Version 1 bei solchen Disketten SPT=15 und GPL=3 vorgegeben war, so dass diese Disketten in der Version 2 nur mit diesen Parametern zu verarbeiten sind. Es sollte von solchen Disketten ein COPY auf andere Disketten mit DDGA=17 erfolgen, da darauf 12% mehr Platz ist. Mit dem Flag ti=bH wird bewirkt, dass eine Verzoegerungszeit bei der Kopfpositionierung eingehalten wird. Dies muss bei den meisten 8"-Floppies getan werden.

10. PDRIVE,dn1,dn2,TI=BH,TD=D,TC=77,SPT=34,TSR=3,GPL=8,
DDSL=17,DDGA=4

Mit dieser Eingabe fuer Modell I kann man 8"-Floppies mit 77 Spuren, doppeltseitig und mit einfacher Dichte verarbeiten, eine Verzoegerungszeit bei der Positionierung wird eingehalten.

11. PDRIVE,dn1,dn2=dn3

Mit dieser Eingabe werden die Parameter von Drive 2 fuer den Drive 3 uebernommen.

12. PDRIVE,dn1,dn2=dn3,TC=40,TSR=2

Mit dieser Eingabe werden die Parameter von Drive 2 fuer den Drive 3 uebernommen. Zusaetzlich werden die Parameter TC und TSR fuer Drive 3 veraendert.

13. PDRIVE,0,A

Mit dieser Eingabe werden die Parameter fuer AL (siehe "SYSTEM" auf Seite 46) Floppies in den Speicher uebernommen.

14. PDRIVE,0,dn2=dn3,A

Mit dieser Eingabe werden die Parameter fuer das Laufwerk dn2 von denen von dn3 ueberschrieben und ebenso wie beim letzten Beispiel in den Speicher uebernommen und somit sofort aktiv.

2.38 PRINT

PRINT,FNAME<,STARTZEILE(ZEILENANZAHL)>

Dieses Kommando funktioniert genau gleich wie LIST (siehe "LIST" auf Seite 32) mit dem Unterschied, dass die Liste auf den Drucker ausgegeben wird.

2.39 PROT

PROT,Passwort1:DN1<,NAME=name1><,DATE=mm/dd/yy>
<,RUF><,LOCK><,UNLOCK><,PW=Passwort2>

Veraendern der Kontrolldaten einer Diskette

Mindestens einer der angegebenen Parameter muss angegeben werden. Die zu veraendernde Diskette befindet sich in Laufwerk :DN1. Wenn die Passwoerter aktiv sind, muss das gueltige Hauptpasswort fuer diese Diskette eingegeben werden.

NAME=name1 Die Diskette erhaelt den neuen Namen 'name1'

DATE=mm/dd/yy Die Diskette erhaelt das neue Datum mm/dd/yy

RUF Dieser optionale Parameter setzt alle Update-Kennzeichen fuer alle Files der Diskette auf 0. Wenn ein Benutzer nur die Files einer Diskette mit dem Update-Kennzeichen kopiert hat (siehe COPY mit Option UPD), kann er nach dem Kopieren mit PROT und RUF alle Kennzeichen loeschen fuer weitere Kopien von veraenderten Files. Einfaches Schreiben oder Zurueckschreiben eines Sektors genuegt bereits, damit DOS das Update-Flag setzt.

LOCK Alle Files dieser Diskette erhalten sowohl als Zugriffs- als auch als Veraenderungspasswort das Hauptpasswort dieser Diskette. Wird gleichzeitig der Parameter PW=xxx eingegeben, wird anstelle des Hauptpasswortes das unter xxx angegebene Passwort benutzt. Dies ist fuer den Benutzer eine der Moeglichkeiten, bei aktiven Passwoertern das Passwort eines Files zu veraendern, falls er das Passwort eines speziellen Files vergessen hat. Der einfachere Weg waere allerdings, vorausgesetzt der Benutzer hat eine Systemdiskette ohne aktive Passwoerter (System-Option AA=N), wenn er von dieser Systemdiskette aus mit dem Kommando ATTRIB direkt das Passwort dieses Files aendern wuerde. Dies haette dann keine Auswirkung auf andere Files.

UNLOCK Die Passwoerter aller Files dieser Diskette, ausgenommen der Systemfiles und der nicht sichtbaren Files, werden auf Blank gesetzt, das heisst, diese Files haben kein Passwort mehr.

PW=Passwort2 Das neue Hauptpasswort dieser Diskette wird Passwort2

1. PROT,2,RUF

   Die Updatekennzeichen fuer jeden File der Diskette 2 werden geloescht.

2. PROT,DISK1:1,NAME=AAB3,DATE=07/15/81,PW=NEWPSWD

   In diesem Beispiel werden die Passwoerter aktiviert. Dafuer war die Eingabe des Hauptpasswortes der Diskette DISK1 noetig. Das neue Hauptpasswort fuer die Diskette ist

jetzt NEWPSWD, der Name der Diskette wurde geaendert in AAB3, das neue Datum ist 07/15/81.

2.40 PURGE

PURGE,Passwort:DN1<,/EXT><,USR>

Selektives Loeschen von Files

Es wird die Diskette in Drive :DN1 benutzt fuer dieses Kommando. Wenn die Passwoerter aktiv sind, muss das Hauptpasswort dieser Diskette eingegeben werden.

Fuer jeden auf der Diskette vorhandenen File, ausser bei BOOT/SYS und DIR/SYS, fragt das System ueber Bildschirm beim Benutzer nach, ob dieser File geloescht werden soll. Antwortet der Benutzer mit Y wird dieser File geloescht, bei Eingabe von N wird nach dem naechsten File gefragt. Der File wird sofort geloescht, so, als wuerde das Kommando KILL benutzt. Bei Eingabe von Q wird das Kommando PURGE sofort beendet.

Die Parameter sind:

/EXT bei dieser Option werden vom Kommando PURGE nur solche Files beruecksichtigt, die die Erweiterung /EXT haben. Diese darf 0 bis 3 Zeichen haben.

USR bei dieser Option werden vom Kommando PURGE alle unsichtbaren und System-Files nicht beruecksichtigt.

Beispiele fuer PURGE:

1. PURGE,1

   Alle Files auf der Diskette 1, ausser BOOT/SYS und DIR/SYS werden auf dem Bildschirm angezeigt und bei Eingabe von Y geloescht.

2. PURGE,0,/DAT

   Alle Files auf der Diskette 0, die die Erweiterung /DAT haben, werden auf dem Bildschirm angezeigt und bei Eingabe von Y geloescht.

3. PURGE,0,USR

   Alle Files auf der Diskette 0, die keine System-Files und die nicht unsichtbar sind, werden auf dem Bildschirm angezeigt und bei Eingabe von Y geloescht.

2.41 R

Letztes DOS-Kommando wiederholen.

Dieses Kommando bewirkt, dass das letzte eingegebene DOS-Kommando wiederholt wird, ausgenommen das Kommando R selbst. Hier ein Beispiel:

```
DIR 1        gefolgt von
R
```

wird ausgefuehrt, als haette man die 2 DOS-Kommandos

```
DIR 1
DIR 1
```

eingegeben. Das Kommando R kann nicht aufgerufen werden vom BASIC aus als CMD"R" ohne ENTER, diese Funktion verlangt die Eingabe von 2 Zeichen oder mehr.

Das Kommando R hat keine Parameter und muss exakt eingegeben werden als R und ENTER. Hat man versehentlich mehr eingegeben und die zusaetzlichen Zeichen mit Backspace geloescht, wurde dadurch auch das vorhergehende, zu wiederholende Kommando veraendert. Das Ergebnis wird dann garantiert ein anderes sein als erwartet.

Wenn das vorhergehende Kommando durch irgend einen Umstand veraendert wurde, wird das Kommando R falsche Ergebnisse bringen.

Bei der SYSTEM-Option BE=N arbeitet das Kommando R nicht, es wird ignoriert und DOS READY angezeigt.

2.42 RENAME

RENAME,FNAME1<,to>,FNAME2

Umbenennen eines Files

Der File mit dem Namen FNAME1 wird in FNAME2 umbenannt, wobei FNAME2 einen Namen und optional einen Filetyp enthaelt. Falls FNAME1 keine Angabe ueber das Laufwerk enthaelt, sucht das System ueber alle Floppies, bis es die erste Floppy gefunden hat, auf der ein File mit dem Namen FNAME1 enthalten ist.

Beispiel fuer RENAME:

1. RENAME XXX/DAT:1 YYY/OBJ

   Der File XXX/DAT auf der Diskette in Laufwerk 1 aendert seinen Namen in YYY und seinen Zusatz in OBJ.

2.43 ROUTE

1. ROUTE
2. ROUTE,CLEAR
3. ROUTE,dev1<,dev2><,dev3>.....

Die Anwendung des Kommandos ROUTE erlaubt dem Programmierer mehr Flexibilitaet in der Ein- und Ausgabe. Er kann bestimmen, von woher die Eingabe erfolgen soll (Tastatur, RS232) und wohin die Ausgabe gelegt werden soll (Display, Drucker oder RS232).

Bei der ersten Eingabe von nur ROUTE werden alle geaenderten Verbindungen angezeigt, falls welche existieren. Gibt es keine umgeleiteten Verbindungen, wird nichts angezeigt.

Das Kommando ROUTE,CLEAR loescht alle umgeleiteten Verbindungen.

Bei der dritten dargestellten Eingabe gibt dev1 die Ein- oder Ausgabe an, die umgeleitet werden soll. dev2, dev3 etc. geben an, wohin oder woher umgeleitet werden soll. Beim Modell I lauten die benutzten Bezeichnungen fuer die Ein- und Ausgabengeraete KB (Keyboard) fuer Tastatur, DO (Display out) fuer

Bildschirm, PR (Printer) fuer Drucker und NL fuer Null (Geraet abgeschaltet). Beim Modell III kommen noch RI (RS232 In) fuer den Eingabeteil der RS232-Schnittstelle und RO (RS232 Out) fuer den Ausgabeteil der RS232-Schnittstelle hinzu. Eine Eingabeschnittstelle (KB oder RI) darf nicht an ein Ausgabegeraet angeschlossen werden (DO, PR oder RO).

Wenn dev1 angegeben wird, loescht das Kommando ROUTE alle vorher definierten Umleitungen fuer diese Einheit.

Anstelle von dev2, dev3 usw. kann auch der Parameter MM=adr angegeben werden. adr gibt an, dass im Memory unter dieser Adresse eine Routine steht, welche die entsprechende Ein- oder Ausgabe macht. Die eingegebene Adresse muss groesser als 51FFH sein. Die ersten 12 Bytes dieser Routine werden fuer das DOS benoetigt und duerfen vom Benutzer nicht veraendert werden. Beim Umleiten wird diese Routine mit einem CALL auf das 13. Byte angesprungen. Der Anwender, der diese Routine geschrieben hat, ist selbst dafuer verantwortlich, dass alle Register ausser AF gerettet werden. Wenn dev1 ein Eingabegeraet ist, so muss das eingegebene Zeichen im Accu stehen, ist dieses Zeichen binaer 0, so wird angenommen, dass von diesem Geraet keine Eingabe mehr erfolgt. Wenn dev1 ein Ausgabegeraet ist, so steht im Register C das auszugebende Byte.

Die maximale Anzahl der umzuleitenden Geraete ist beim Modell I vier, beim Modell III sechs. Diese Zahl schliesst den Parameter MM=adr nicht ein.

Achtung Input- oder Outputcharacter werden nicht veraendert. Dies kann zu Problemen fuehren (z. B. koennen die Steuerzeichen fuer den Bildschirm andere Steuerzeichen fuer den Drucker sein).

Beispiele fuer ROUTE:

1. ROUTE,PR,DO

   Die Ausgabe fuer den Printer wird auf den Bildschirm umgeleitet.

2. ROUTE,DO,DO,PR

   Die Ausgabe fuer den Bildschirm wird sowohl auf dem Bildschirm als auch auf dem Drucker gemacht.

3. ROUTE,PR,DO,PR

   Die Ausgabe fuer den Drucker wird sowohl auf dem Drucker als auch auf dem Bildschirm gemacht. Dieses und das letzte Beispiel zusammen ergeben beim Modell III die TRSDOS-Funktion DUAL.

4. ROUTE,KB,RI (nur beim Modell III)

   Die Eingabe erfolgt statt von der Tastatur von der RS232-Schnittstelle.

5. ROUTE,DO,RO (nur beim Modell III)

   Die Ausgabe erfolgt statt auf den Bildschirm auf die RS232-Schnittstelle.

6. ROUTE,PR,MM=0FE80H

   Die Ausgabe erfolgt statt auf den Drucker auf die Routine im Memory auf der Adresse 0FE80H (die eigentliche Einsprungstelle der Routine ist 0FE8CH).

7. ROUTE,KB,KB,MM=0FE00H

Die Eingabe erfolgt statt von der Tastatur von der Routine im Memory auf der Adresse 0FE00H (die eigentliche Einsprungstelle der Routine ist 0FE8CH) oder von der Tastatur. Die Tastatur hat Vorrang.

8. ROUTE,PR,NL

   Die Ausgabe fuer den Drucker wird unterdrueckt.

9. ROUTE,PR

   alle Umleitungen fuer den Drucker werden geloescht.

10. ROUTE,CLEAR

    alle Umleitungen werden geloescht.

2.44 SETCOM

RS232-Parameter setzen (nur Modell III)

SETCOM<,OFF><,WORD=wl><,BAUD=br><,STOP=sb>
<PARITY=pp><,WAIT><,NOWAIT>

Das Kommando SETCOM aendert die Parameter fuer die RS232-Schnittstelle oder, wenn keine Parameter eingegeben werden, zeigt es die aktuellen Parameter an. Ueber die Funktionsweise der RS232-Schnittstelle gibt das Kapitel 8 des Modell III Operation and BASIC Language Reference Manual Auskunft. Das Kommando SETCOM aendert nur die Standardkontrollbloecke und die Routinen der Schnittstelle.

Mit dem Parameter OFF wird die RS232 Schnittstelle abgeschaltet. Es duerfen keine weiteren Parameter angegeben werden.

Wird einer der Parameter WORD, BAUD, STOP oder PARITY nicht mit angegeben, wird dieser nicht veraendert.

WORD=wl gibt die Anzahl der Bits pro uebertragenem Byte an. wl muss entweder 5, 6, 7 oder 8 sein.

BAUD=br gibt die Uebertragungsrate an fuer sowohl senden als auch empfangen. Die 16 erlaubten Werte fuer br sind 50, 75, 110, 134, 150, 300, 600, 1200, 2400, 3600, 4800, 7200, 9600 oder 19200.

STOP=sb gibt die Anzahl der Stopbits pro uebertragenem Byte an. sb kann entweder 1 oder 2 sein.

PARITY=pp gibt die Art des Paritybits an, wobei 1 = ungerade, 2 = gerade und 3 = kein Paritybit bedeutet.

WAIT oder NOWAIT Es kann nur entweder WAIT oder NOWAIT benutzt werden. WAIT bedeutet, dass die Input-Routine wartet, bis sie ein Byte empfangen hat, bzw. die Output-Routine wartet, bis das Byte gesendet wurde. Wenn weder WAIT noch NOWAIT angegeben wurde, bleibt der momentane Status unveraendert.

Beispiele fuer SETCOM:

1. SETCOM,WORD=8,BAUD=300,STOP=1,PARITY=1,WAIT

   Dieses Kommando aktiviert die RS232-Schnittstelle, falls nicht bereits aktiv, und setzt sie auf 8 Bits pro Byte, 300 Baud Uebertragungsrate, ein Stopbit, ungerade Paritaet und

wartet, wenn ein Byte ausgegeben oder eingelesen wird, bis diese Uebertragung fertig ist.

2. SETCOM,NOWAIT,PARITY=3,WORD=7

   Dieses Kommando aktiviert die RS232-Schnittstelle, falls nicht bereits aktiv, und setzt sie auf 7 Bits pro Byte, kein Paritybit und wartet nicht, wenn ein Byte ausgegeben oder eingelesen wird, bis diese Uebertragung fertig ist. Die Abhandlung nach der Ein- oder Ausgabe uebernehmen die TRS80-Interrupt-Routinen. Die anderen Parameter werden nicht beeinflusst.

2.45 STMT

Meldung ausgeben

STMT,msg

Da normale DOS-Kommados immer angezeigt werden, hat dieses Kommando normalerweise keine Funktion. Nur dann, wenn dieses Kommando ueber einen DOS-CALL aufgerufen wird (der das Kommando nicht anzeigt), wird die Meldung msg auf dem Bildschirm ausgegeben.

STMT ist eine der drei Moeglichkeiten, beim Chaining eine Meldung anzuzeigen ohne dass das Chaining unterbrochen wird. Damit koennen auch mehrere Zeilen angezeigt werden, wenn die letzte Zeile dann ein PAUSE-Kommando ist.

Beispiele fuer STMT:

1. STMT,Meldung Pass 1 fertig

   Dies ist die einfache Ausgabe einer Meldung, die dem Operator mitteilt, dass Pass 1 eines Programmes fertig ist. DOS haelt dabei nicht an.

2. STMT,DISKETTE XXX AUS DEM LAUFWERK 2 NEHMEN
   PAUSE,UND DISKETTE YYY EINLEGEN

   In diesem Beispiel wird dem Operator mitgeteilt, dass er eine Diskette aus dem Laufwerk 2 herausnehmen soll und die naechste einlegen soll. Danach wartet das DOS auf das Druecken der ENTER-Taste.

2.46 SYSTEM

Veraendern der Systemparameter

SYSTEM,Password:DN1<,AA=aa><,AB=ab><,AC=ac><,AD=ad>
<,AE=ae><,AF=af><,AG=ag><,AH=ah><,AI=ai><,AJ=aj>
<,AK=ak><,AL=al><,AM=am><,AN=an><,AO=ao>
<,AP=ap><,AQ=aq><,AR=ar><,AS=as><,AT=at>
<,AU=au><,AV=av><,AW=aw><,AX=ax><,AY=aY>
<,AZ=az><,BA=ba><,BB=bb><,BC=bc><,BD=bd>
<,BE=be><,BF=bf><,BG=bg><,BH=bh><,Bi=bi>
<,BJ=bj><,BK=bk><,BL=bl><,BM=bm><,BN=bn>

Die System-Werte einer NEWDOS/80-Systemdiskette im Laufwerk :DN1 werden veraendert bzw. angezeigt. Wenn die Passwoerter aktiv sind, muss das Hauptpasswort dieser Diskette eingegeben werden, andernfalls kann das Passwort weggelassen werden. Wenn ausser dem Passwort und der Laufwerksnummer keine Parameter eingegeben werden, werden nur die Systemparameter angezeigt.

Werden neue Parameter eingegeben, kann das in beliebiger Reihenfolge geschehen. Es werden auch nur die Systemparameter veraendert, die mit dem Kommando SYSTEM eingegeben werden, alle anderen bleiben unveraendert.

Falls viele Parameter veraendert werden sollen, kann es notwendig sein, dieses Kommando mehrmals aufzurufen, da der DOS-Eingabebuffer max. 79 Character enthalten kann.

Die eingegebenen Aenderungen der Parameter werden erst dann wirksam, wenn ein Reset erfolgt und die geaenderte Diskette in Drive 0 ist.

AA=aa wenn aa=y sind alle Passwortkontrollen eingeschaltet, bei aa=n kann ohne Passwoerter gearbeitet werden.

AB=ab wenn ab=y laeuft das System nur im RUN-MODE. Die Optionen ae=n und af=n werden beim Reset aktiviert, die Unterbrechung der Ausfuehrung des 'AUTO'-Kommandos beim Reset durch Druecken der 'ENTER'-Taste ist verboten. Ein ausfuehrbares 'AUTO'-Kommando musste vorher eingegeben geben werden. Wenn das Betriebssystem in den 'DOS READY' oder 'MINI-DOS READY' Status kommen sollte im RUN-ONLY-MODE, wird 'RUN ONLY STOPPED!! PRESS 'R' FOR RESET' angezeigt und das System wartet auf diese Eingabe. Danach wird ein Reset ausgefuehrt. In BASIC wird die Funktion der Break-Taste unterdrueckt, die Eingabe von LOAD ergibt einen Fehler, es wird nur RUN akzeptiert. Bei ab=n geht das System beim naechsten Reset in den normalen Modus.

AC=ac Wenn ac=y und aj=y wird die NEWDOS/80 Tastatur-Entprellung aktiv. Bei ac=n oder aj=n wird diese Routine umgangen (nur Modell I).

AD=ad Wenn ad=y ist die normale Tastenfunktion 'JKL' (Drucken des Bildschirminhaltes) moeglich, bei ad=n ist diese Funktion unterdrueckt.

AE=ae Wenn ae=Y kann das DEBUG ueber die Tastenfunktion '123' aufgerufen werden, ohne dass das DEBUG ueber das DOS- Kommando DEBUG,Y aktiviert wurde. Bei ae=n muss zuerst das DEBUG geladen werden ueber DEBUG,Y und kann dann ueber die Breaktaste aufgerufen werden.

AF=af Wenn af=Y ist die normale Tastenfunktion 'DFG' (MINI-DOS) moeglich, bei af=n kann MINI-DOS nicht aufrufen werden.

AG=ag Bei ag=Y hat die Breaktaste den normalen Code von 01H, bei ag=n wird der Code der Breaktaste in 00H umgewandelt, die Breaktaste hat also nicht mehr die urspruengliche Funktion. Diese Option wird bei jedem 'DOS-READY' abgefragt und wird bei einer Aenderung sofort aktiv. Anstelle dieser Systemfunktion kann auch der Wert der Adresse 4312H auf 0C9H gesetzt werden fuer normale Funktion, 0C3H entspricht ag=n (Adresse 4478H bei Modell III).

AH=ah AH hat keine Auswirkung in NEWDOS/80 Version 2.

AI=ai (gilt nur fuer Modell I) Bei ai=y muss die Modifikation fuer Kleinbuchstaben installiert sein, bei ai=n nur Grossbuchstaben. Anwenderprogramme koennen Bit 4 der Adresse 436CH ueberpruefen, bei 1 ist ai=y, bei 0 ist ai=n. Die Programme DEBUG und SUPERZAP kontrollieren dieses Bit und geben dann entsprechend in Gross- oder Kleinbuchstaben aus.

AJ=aj Bei aj=y ist die spezielle Tastaturroutine aktiv. Diese Routine enthaelt zum einen eine Entprellung und zum anderen eine andere Art der Erkennung der Funktionen

'JKL', '123' und 'DFG'. Diese Funktionen koennen sonst nur ueber die Interrupts erkannt werden.

AK=ak Nicht in NEWDOS/80 Version 2. Die Funktion dieses Parameters uebernimmt jetzt die Option AX.

AL=al al gibt die Anzahl der angeschlossenen Floppylaufwerke an. Bei einer angeschlossenen Floppy gibt man also AL=1 ein. Der Wert von AL kann zwischen 1 und 256 liegen, sollte aber auf keinen Fall groesser als 4 sein.

AM=al am gibt die Anzahl der Versuche von Zugriffen auf die Floppies an. Der Wert kann zwischen 0 und 255 liegen (0 entspricht 256 Zugriffen). Im TRSDOS werden max. 10 Zugriffe gemacht.

AN=an an gibt den Defaultwert fuer das Laufwerk beim Kommando DIR an.

AO=ao Wenn ein File neu auf eine Floppy geschrieben wird, sucht das Betriebssystem zuerst auf allen angeschlossenen Floppies nach einem Directoryeintrag mit dem gleichen Namen. Findet das System keinen gleichen Eintrag, sucht es zuerst nach einem freien Eintrag auf einer Diskette. Auf diese Diskette wird dann dieser File kopiert (immer vorausgesetzt, dass keine Laufwerksnummer angegeben wurde). Wenn AO eingegeben wird, sucht das System auf keinem Laufwerk, dessen Nummer kleiner ist als ao.

AP=ap ap ist die Memoryadresse, die als hoechster Wert in 4049H eingetragen wird. Auf die gleiche Adresse haben auch das DOS-Kommando HIMEM Zugriff und beim Initialisieren das BASIC. Der Wert 0 entspricht der Adresse 0FFFFH.

AQ=aq Wenn aq=y ist, ist die Clear-Taste aktiv, bei aq=n ist kein Clear moeglich.

AR=ar Bei ar=Y ist ein volles Copy einer Diskette ohne Passwort auch dann moeglich, wenn die Passwoerter aktiviert sind. Bei ar=N muss dagegen das Passwort eingegeben werden.

AS=as (gilt nur fuer Modell I) Wenn as=Y eingegeben wurde, werden alle Eingaben an das BASIC von Klein- auf Grossbuchstaben konvertiert. Dies ist vor allem dann nuetzlich, wenn die Treiber fuer Kleinbuchstaben nicht benutzt werden. Es ist trotzdem moeglich, Kleinbuchstaben mit der Shifttaste einzugeben. Die Buchstaben werden dann in Gross angezeigt, sind aber Kleinbuchstaben. Bei as=n wird der Text der Eingabe belassen.

AT=at AT=N setzt das Chaining in den Record-Mode, was bedeutet, dass nur volle Records vom Chain-File kommen, ueber die Tastatur koennen einzelne Character eingegeben werden. Bei AT=Y kommt alles, also auch einzelne Character, vom Chain-File.

AU=au AU=Y setzt die Repeat-Funktion auf aktiv. Die erste Wiederholung einer Taste erfolgt nach AV (siehe AV) * 25 ms. Weitere Wiederholungen erfolgen so schnell wie moeglich, jedoch nicht mehr als 12 pro Sekunde. Bei AS=N wird diese Funktion abgeschaltet.

AV=av AV wird benutzt, wenn AU=Y ist. av ist die Anzahl der 25 ms Intervallen, nach denen eine Wiederholung der momentanen Taste erfolgen soll.

AW=aw aw ist die Anzahl der erlaubten Schreibzugriffe auf einen File mit Verify. Dieser Zaehler arbeitet zusammen mit der Option AM und kommt zur Anwendung, wenn das Lesen beim Verify einen Fehler ergeben hat. Bisher war es so, dass wenn beim Schreiben auf die Diskette kein Fehler aufgetreten ist und nur das Verify einen Fehler

gebracht hat, es dem Operator ueberlassen war, ob er nochmals versuchen moechte, zu schreiben. Jetzt wird automatisch, wenn aw groesser 1 ist, nochmals der letzte Satz zurueckgeschrieben.

AX=ax ax ist der ASCII-Code des hoechsten Characters, den der Drucker verarbeiten kann. Dieser Parameter wird vom System benutzt, um anstelle eines hoeheren Characters ein Blank oder einen Punkt auszugeben. Der Wert von ax soll 255 nicht uebersteigen. Er wird in einem Byte, bei Modell I auf Adresse 4370H, bei Modell III auf Adresse 4290H, abgespeichert.

AY=ay AY wird nur beim Reset benutzt. Damit stellt das DOS fest, ob es vorher schon aktiv war oder nicht (z. B. nach dem Einschalten). AY=Y bewirkt, dass der Operator nach der Zeit und dem Datum gefragt wird, wenn das DOS vorher noch nicht aktiv war. Bei AY=N werden Zeit und Datum auf 0 gesetzt.

AZ=az AZ wird nur beim Reset benutzt. Damit stellt das DOS fest, ob es vorher schon aktiv war oder nicht (z. B. nach dem Einschalten). AZ=Y bewirkt, dass der Operator nach der Zeit und dem Datum gefragt wird, wenn das DOS vorher noch nicht aktiv war. Bei AZ=N werden Zeit und Datum auf dem gleichen Wert wie vorher belassen.

BA=ba BA=Y bewirkt, dass beim Reset das Kommando 'ROUTE,DO,NL' aktiviert wird. Dadurch werden alle Meldungen auf dem Bildschirm unterdrueckt, bis der Operator oder das aufgerufene Programm entweder ROUTE,CLEAR oder ROUTE,DO eingeben. Bei BA=N wird dies unterdrueckt.

BB=bb (nur beim Modell III) BN=N informiert das System darueber, dass die Interrupts von der Uhr 60 Mal pro Sekunde kommen, bei BB=Y kommen diese Interrupts 50 Mal pro Sekunde.

BC=bc BC=Y bedeutet, dass der Operator von Hand den Chaining-Modus unterbrechen kann. Bei BC=N ist dies unterdrueckt. Bei der Option fuer den RUN-ONLY-Modus wird BC=N angenommen.

BD=bd BD=Y bedeutet, dass der Operator die Ausfuehrung des AUTO-Kommandos unterbrechen kann. Bei BD=N ist dies unterdrueckt. Bei der Option fuer den RUN-ONLY-Modus wird BD=N angenommen.

BE=be BE=Y bedeutet, dass das Kommando R eingegeben werden kann. Bei BD=N ist dies unterdrueckt.

BF=bf (nur Modell I) BF=Y bewirkt, dass beim Reset sofort das Gleiche ausgefuehrt wird wie beim Kommando LCDVR,Y. Bei BF=N wird es LCDVR=N. Stellt das System beim Reset fest, dass die Kleinschreibung nicht installiert ist, wird BF=N angenommen.

BG=bg BG=Y bewirkt, dass beim Reset sofort das Gleiche ausgefuehrt wird wie beim Kommando LC,Y. Bei BF=N wird es LC,N.

BH=bh BH=Y bewirkt, dass beim Reset der Cursor sofort blinkt wie bei BLINK,Y. Bei BH=N blinkt der Cursor nicht.

BI=bi Beim Reset wird der Wert von BI als Cursor-Zeichen ausgegeben. Bei BI=0 wird der Standard-Cursor (bei Modell I = 95, bei Modell III = 176) ausgegeben.

BJ=bj Diese Option ist zu benutzen, wenn die CPU auf eine hoehere Geschwindigkeit umgebaut wurde. Der numerische Wert von BJ bedeutet, um wieviel die CPU verschnellert wurde (bei Verdopplung der Geschwindigkeit BJ=2 eingeben). Die Warteloops bei der Ein- und

Ausgabe werden um den entsprechenden Faktor verlaengert. Bei BJ=1 wird die Wartezeit nicht verlaengert.

BK=bk Diese Option ist zu benutzen, wenn das Programm DIRCHECK die Optionen W und C ausfuehren darf. Bei BK=N meldet das System bei diesen Optionen 'DISK ACCESS DENIED'.

BM=bm Diese Option ist zu benutzen, wenn das FORMAT erst nach dem Formatieren den VERIFY-Lauf startet. Normalerweise wird im NEWDOS/80 Version 2 eine Spur sofort nach dem Formatieren kontrolliert. Bei BM=Y unterbleibt dies, die Diskette werden erst nach dem kompletten FORMAT kontrolliert.

BN=bn (nur bei Modell I) Bei BN=N werden die Sektoren des Directorys einer mit einfachen Schreibdichte beschriebenen Diskette so markiert, dass sie im TRSDOS auf Modell III lesbar sind. Bei BN=N sind diese Disketten nur im NEWDOS/80 auf Modell III lesbar.

Obwohl die Information im Directory selbst die gleichen sind, ausgenommen einige Erweiterungen beim NEWDOS/80, ist der Status der Directory-Sektoren selbst anderst. Die Sektoren muessen beim Modell III geschuetzt sein.

Das einfache Aendern der System-Option BN genuegt jedoch nicht, um den Sektorstatus zu aendern. Er wird nur geaendert, wenn diese Sektoren auf die Diskette geschieben werden. Um diese Sektoren richtig zurueckzuschreiben, muss man entweder DIRCHECK oder das Kommando WRDIRP mit der Option W aufrufen. Wurde eine Diskette im Modell III benutzt oder im Modell I mit der Option BN=Y, muessen die Sektoren entweder mit WRDIRP oder DIRCHECK mit der Option BN=N als ungeschuetzt zurueckgeschrieben werden, damit sie mit den normalen Optionen unter NEWDOS/80 zu benutzen sind.

BO=bo Reserviert fuer spaetere Verwendung.

Beispiele fuer SYSTEM:

1. SYSTEM,0,AL=4,AA=Y,AU=Y,AV=20,AT=Y

   Die Parameter der System-Funktion AL, AA, AU, AV und AT werden auf der Floppy in Laufwerk 0 veraendert. Alle anderen Parameter bleiben gleich. Danach werden alle Parameter angezeigt. Die veraenderten Parameter werden erst beim naechsten Reset wirksam.

2. SYSTEM,2,AP=0FF00H,AN=1,AX=126

   Die Parameter der System-Funktion AP, AN und AX werden auf der Floppy in Laufwerk 2 veraendert. Alle anderen Parameter bleiben gleich. Danach werden alle Parameter angezeigt. Die veraenderten Parameter werden erst beim naechsten Reset mit dieser Floppy in Laufwerk 0 wirksam.

2.47 TIME

TIME<,hh:mm:ss>

Mit diesem Kommando wird die Uhrzeit in die mitlaufende Uhr eingesetzt. hh ist die Stunde (Wert zwischen 00 und 23), mm die Minuten (00-59), ss die Sekunden (00-59). Bei diesem Kommando werden die eingegebenen Zahlen nicht kontrolliert.

Wir beim Kommando kein Parameter (Uhrzeit) eingegeben, wird die aktuelle Uhrzeit angezeigt.

Beim Reset wird die Zeit auf 00:00:00 gesetzt. Die Zahlen werden im Speicher ab 4041H in der Reihenfolge Sekunden, Minute und Stunden abgespeichert und, solange die Interrupts aktiv sind, im richtigen Format weitergezaehlt. Wenn die Interrupts zeitweise gesperrt werden, entweder durch Kommando oder durch Floppyzugriffe, gehen natuerlich Interrupts, damit also auch Sekundenimpulse verloren. Die Uhr geht dann nach.

Beispiele fuer TIME:

1. TIME,15:23:00

   Die interne Uhr wird auf 15:23 Uhr gesetzt.

2. TIME

   Der momentane Stand der Internen Uhr wird angezeigt.

2.48 VERIFY

Kontroll-Lesen auf der Floppy nach jedem Schreiben

VERIFY<,x>

NEWDOS/80 liest nach jedem Schreiben auf die Floppy das Geschriebene noch einmal zurueck, sowohl beim byteweisen Schreiben als auch beim blockweisen Schreiben. Nur beim sektorweisen Schreiben ueber die Routine 4439H erfolgt kein Verify.

Wird als Parameter im VERIFY-Kommando Y oder ON mit oder ohne Klammer angegeben, wird auch das Schreiben ueber die Routine 4439H ueberprueft, bei N oder OFF bzw. (N) oder (OFF) wird dieses Verify unterdrueckt.

COPY, EDTASM und BASIC (SAVE) schreiben normalerweise ohne Verify auf die Floppy, lesen dann aber den geschriebenen Satz zur Kontrolle zurueck. In BASIC werden alle PRINT-Befehle sowie PUT auf "MI"- und "FI"-Files automatisch ueberprueft, da sie mit byteweiser Ausgabe arbeiten. Die anderen Filetypen benutzen sektorweise Ausgabe, sie werden nur dann ueberprueft, wenn Verify aktiviert wird.

2.49 WRDIRP

Schreiben des Directory schreibgeschuetzt.

WRDIRP,dn1

WRDIRP bewirkt, dass die Sektoren des Directory gelesen werden und dann entsprechend den System-Optionen BN und BK wieder zurueckgeschrieben werden.

Dieses Kommando wird nur benoetigt, wenn die Disketten mit einfacher Schreibdichte zwischen Modell I und Modell III unter NEWDOS/80 Version 2 austauschbar sein sollen.

Mit diesem Kommando wird das Directory waehrend MINI-DOS auf den richtigen Status gesetzt. Im normalen DOS READY Status kann dies mit DIRCHECK getan werden, meist stellt man jedoch erst beim Arbeiten, also mitten in einer Funktion fest, dass das Directory nicht stimmt. Da dann das Programm DIRCHECK nicht aufgerufen werden kann, wurde WRDIRP fuer das Ausfuehren unter MINI DOS ausgelegt. Es ist dabei aber Vorsicht geboten. WRDIRP liest zuerst den 1. Sektor der Diskette, um aus dem 3. Byte den Startsektor des Directory zu entnehmen. Dann wird das Directory ueberprueft, ob die Eintraege fuer BOOT/SYS und DIR/SYS enthalten sind. Erst dann wird der Status der entsprechenden Sektoren nach der System-Option BN und BK veraendert. Sollte es Probleme geben, wird die Anwendung von DIRCHECK empfohlen.

Bei der Systemfunktion BK=N kann das Kommando WRDIRP nicht ausgefuehrt werden.

Beispiel fuer WRDIRP:

1. WRDIRP,1

   Das Directory der Diskette in Laufwerk 1 erhalt die Protect-Marken entsprechend der Systemoption BN.

3.0 DOS - ROUTINEN

Dieses Kapitel behandelt die DOS-Routinen, die dem Benutzer von einem Maschinen-Programm aus zur Verfuegung stehen.

3.1 ALLGEMEINES

Jede dieser Routinen hat eigene Einsprung- und Rueckkehrkonditionen, die vom Benutzer strikt einzuhalten sind. Einige dieser Konditionen sind hier unter a, b und c aufgefuehrt, in der Beschreibung der Routine wird nur noch auf die entsprechenden den Unterpunkte verwiesen.

1. (a) Nur das Register AF (ACCU, Status) wird von der Routine veraendert. Alle anderen benutzten Register werden beim Einsprung gerettet und vor dem Ruecksprung zurueckgespeichert.

2. (b) Beim Ruecksprung wird das Zero-Bit gesetzt wenn kein Fehler aufgetreten ist, einen DOS-Fehler kann man ueber 'NOT ZERO' abfragen. Die DOS-Fehlermeldung steht im ACCU.

3. (c) Beim Einsprung muss das Registerpaar DE auf einen geoeffneten File-Control-Block (FCB) deuten.

Einige dieser Routinen sind gegenueber dem TRSDOS unterschiedlich bzw. haben eine andere Funktion. Bei einer Umstellung von Assembler-Programmen auf NEWDOS/80 ist genau darauf zu achten. Auf die einzelnen Unterschiede wird jeweils in den Routinen hingewiesen. Diese treten vor allem in der Behandlung von FCB's auf (EOF und Next-Record).

Von jeder Routine ist die Einsprungadresse fuer ein JUMP oder CALL, der Titel und die Funktion aufgefuehrt.

Wenn nicht anders aufgefuehrt, benutzt die Routine den Stack des aufrufenden Programmes und kehrt auch in dieses Programm zurueck, ausgenommen die Ruecksprueng e ins Betriebssystem.

3.2 402DH - NO ERROR EXIT

Ruecksprung ins Betriebssystem ohne Fehler

Diese Routine springt in das Betriebssystem oder das vorangegangene Programm zurueck, von dem es aufgerufen wurde.

- Dies kann entweder MINI-DOS oder DOS gewesen sein, der Stackpointer wird auf den Wert gesetzt, den er beim Aufruf des Anwenderprogrammes hatte bzw. auf den DOS-Stackbereich. Die Breaktaste wird entsprechend der Option 'AG' (siehe "SYSTEM" auf Seite 46) freigegeben oder gesperrt.

- Wurde das Anwenderprogramm direkt ueber DOS gestartet und nicht ueber ein Chaining - Kommando, werden alle Register ausser AF zurueckgespeichert und das Zerobit gesetzt. Die Routine kehrt dann in den Eingabemodus zurueck.

- Wurde beim Kommando SYSTEM der RUN-ONLY-MODE vereinbart und es erfolgt ein Sprung auf diese Routine, wenn das System nicht im Chainingmode ist, erscheint auf dem Bildschirm die Meldung 'RUN ONLY STOPPED. KEY 'R' FOR RESET'. DOS wartet auf die Eingabe und fuehrt dann ein Reset aus.

- Wurde das Anwenderprogramm vom MINI-DOS aus gestartet erscheint die Meldung 'MINI-DOS READY', von DOS aus aufgerufen die Meldung 'DOS READY'.

3.3 4030H - NO DOS-ERROR EXIT

Ruecksprung ins Betriebssystem nach einem Fehler

Der Sprung zu dieser Routine erfolgt dann, wenn ein Fehler bereits angezeigt wurde oder von DOS nicht angezeigt zu werden braucht. Die Routine hat die gleiche Funktion wie 402DH mit folgenden Ausnahmen:

- Wenn sich das System im Chainingmode befindet, wird das Chaining abgebrochen.

- Wurde das Anwenderprogramm von DOS aufgerufen, erfolgt ein Ruecksprung ins DOS, jedoch mit gesetztem Carrybit.

Folgender Unterschied besteht zum TRSDOS: Die DEBUG-Routine wurde bisher automatisch aufgerufen, nicht jedoch beim NEWDOS/80.

3.4 4400H - WIE 402DH

Gleiche Funktion wie bei Routine 440DH.

3.5 4405H - ENTER DOS AND EXECUTE COMMAND

Ruecksprung ins DOS und sofortiges Ausfuehren eines Kommandos. Der Stackpointer wird wieder auf den DOS - Stackbereich gesetzt. Registerpaar HL muss die Startadresse eines Kommandos enthalten, das mit Return (0DH) abschliesst.

3.6 4409H - DOS-ERROR EXIT

Ruecksprung ins DOS nach einem Fehler.

Der Ruecksprung ueber diese Routine erfolgt nach einem DOS - Fehler. Wenn Bit 7 des ACCU's 0 ist wird dieser Fehler nicht angezeigt.

- Wenn sich das System im Chaining - Mode befindet, wird das Chaining abgebrochen.

- Wurde das Anwenderprogramm von DOS aufgerufen, erfolgt ein Ruecksprung ins DOS, jedoch mit Not Zero und Not Carry (NZ und NC).

Ein CALL auf diese Routine mit der DOS - Fehlernummer im A-Register und Bit 7 gesetzt (1) bewirkt eine Anzeige des entsprechenden DOS - Fehlers und ein Ruecksprung zum aufrufenden Programm

Unterschied zum TRSDOS: Anstelle von Bit 7 wird im TRSDOS Bit 6 benutzt. NEWDOS/80 ignoriert aber Bit 6.

3.7 440DH - ENTER DEBUG

Anwenderprogramme haben 2 Moeglichkeiten, das DEBUG aufzurufen:

1. ueber den Z80 Befehl RST 30H
2. ueber einen CALL auf 440DH

Fuer einen Ruecksprung aus DEBUG in das Anwenderprogramm genuegt die Eingabe 'G', vorausgesetzt, dass am Programmzaehler nichts veraendert wurde. Das Anwenderprogramm fuehrt dann die dem RST oder CALL folgende Instruktion aus.

3.8 4410H - ACTIVATE USER INTERRUPT ROUTINE

Eine Interrupt-angestossene Routine wird aktiviert.

Eine Routine bzw. deren Adresse im Speicher wird in die Interruptkette eingefuegt und vom 25 ms Interrupt zyklisch angestossen. Die Register AF, BC, DE und HL werden veraendert. Beim Aufruf dieser Routine muss die Adresse des anzustossenden Programms oder der Anwenderroutine in DE stehen. Dieses Programm muss folgende Form haben:

1. und 2. Byte Werden vom DOS als Pointe benutzt und koennen beim Aufruf einen beliebigen Wert haben.

3. Byte Anzahl der 25 ms Interrupts, nach denen die Routine aufgerufen werden soll. Soll die Anwenderroutine z. B. jede Sekunde gestartet werden, muss das 3. Byte den Wert 40 (28H) enthalten. DOS veraendert diesen Wert nicht.

4. Byte DOS - Zaehler fuer den naechsten Aufruf. Solange dieser Zaehler noch ungleich 0 ist, wird er bei jedem Interrupt um 1 heruntergezaehlt. Sobald er 0 erreicht hat, wird der Wert des 3. Bytes hier abgespeichert und die Anwenderroutine, die mit dem 5. Byte beginnt, gestartet.

Solange eine vom Anwender definierte Routine sich in der Interruptkette befindet und aktiv ist, darf sie nicht veraendert werden, ausser, die Interrupts werden gesperrt. Die ersten zwei Bytes sind ein DOS - Pointer und duerfen nicht veraendert werden.

Unterschied zum TRSDOS: Jede Routine, die ueber einen Interrupt aufgerufen wird, muss fuer die Verwendung unter NEWDOS/80 geaendert werden, da TRSDOS insgesamt 4 Routinen (4410H, 4413H, 4416H und 4419H) benutzt. NEWDOS/80 dagegen benutzt nur die zwei Routinen 4410H und 4413H, die zudem nicht kompatibel sind.

3.9 4413H - DEACTIVATE USER INTERRUPT ROUTINE

Die Routine, die ueber 4410H in die Interruptkette eingefuegt wurde, wird bei einem CALL auf 4413H aus dieser Kette herausgenommen. Die Register AF, BC, DE und HL werden veraendert. Unterschied zum TRSDOS: siehe 4410H

3.10 4416H - RESELECT DISK

Die momentan sich drehende Floppy wird nochmals selektiert, um sie fuer weitere 2.4 sec. am Laufen zu halten. Register AF wird veraendert. Diese Routine existiert nicht im TRSDOS (siehe 4410H).

3.11 4419H - DOS-CALL

Ein DOS-Kommando wird ausgefuehrt und zurueckgesprungen.

Diese Routine wird DOS-CALL genannt. DOS benutzt nicht seinen eigenen Stack, sondern den des aufrufenden Programmes. Alle Register ausser AF werden im Stack gerettet und vor dem Ruecksprung wieder zurueckgespeichert. Das Registerpaar HL muss auf den ersten Character des auszufuehrenden Kommandos deuten, welches mit Return (0DH) abschliessen muss. DOS setzt dann den DOS-CALL-Status, rettet den aktuellen Stackpointer und fuehrt das Kommando aus.

Ein DOS-CALL darf nicht im Chaining - Mode auftreten, ein DOS-CALL darf aber selbst ein CHAIN - Kommando sein.

Auf diese Art wird in BASIC der Befehl CMD"xx" aufgerufen, xx ist das entsprechende DOS - Kommando. Der Benutzer muss selbst ueberpruefen, ob es nicht zwischen dem DOS-Kommando und dem eigenen Programmplatz zu einem Konflikt kommt, ebenso, ob genuegend Platz fuer das Stack vorhanden ist.

Verschachtelte DOS-CALLs sind zulaessig, das DOS kehrt jeweils zum naechsthoeheren Level zurueck. Beim Verlassen des hoechsten Levels wird der DOS-CALL-Status geloescht.

Im DOS-CALL aufgerufene Programme koennen ihren eigenen Stack benutzen, muessen aber ueber eine der drei folgenden Routinen zum aufrufenden Programm zurueckkehren: 402DH, 4030H oder 4409H. Das Programm kann seinem aufrufenden Programm beim Ruecksprung 2 Parameter auf 4403H und 4404H (17411 und 17412 dezimal) uebergeben.

Die Routine 4419H ist unterschiedlich in ihrer Funktion von TRSDOS, siehe "4410H - ACTIVATE USER INTERRUPT ROUTINE" auf Seite 55

Siehe Kapitel "DOS-CALL" auf Seite 74 fuer weitere Informationen.

3.12 441CH - EXTRACT FILESPEC

Filename formatieren

Diese Routine erzeugt aus dem Text, dessen Adresse in HL steht, ein Filename und schreibt ihn in einen Puffer, dessen Adresse in DE steht und fuegt ein 03H - Byte als Endekennung an. Die Register AF, BC und HL werden veraendert.

Wenn das erste Zeichen des Eingabetextes zwischen A und Z oder zwischen 0 und 9 liegt, oder das erste Zeichen ein * ist und das 2. Zeichen entweder zwischen A und Z oder zwischen 0 und 9 liegt, wird der Text in den Puffer geschrieben, dessen Adresse in DE steht. Der Transfer des Textes hoert dann auf, wenn ein Zeichen gelesen wird, das nicht zwischen A und Z bzw. zwischen 0 und 9 liegt und ungleich /, ., oder : ist. Ebenso wird der Transfer nach dem 32. Byte beendet.

Bei weniger als 32 Bytes wird dem Text ein 03H angefuegt als Kennung fuer Textende, falls der Eingabestring nicht bereits 0DH oder 03H enthalten hat. Beim Ruecksprung steht in HL die Adresse des 0DH- oder 03H-Bytes, falls diese nicht vorhanden sind, auf dem naechsten Byte.

In dieser Routine kontrolliert das NEWDOS/80 nicht die richtige Einhaltung des Filenamens, dies wird von der OPEN - Routine durchgefuehrt.

3.13 4420H - OPEN FCB

Neuen File erzeugen

Gleiche Einsprungkonditionen wie bei 4424H. Die Routine 4420H wird praktisch als Subroutine von 4424H aufgerufen, wenn ein File auf der Floppy nicht gefunden wurde. 4420H erzeugt dann den fuer 4424H notwendigen FPDE auf der Floppy und baut den FCB auf.

Wenn der Filename im FCB eine Laufwerksnummer enthaelt, wird auf dieser Floppy in einem freien FDE der FPDE erzeugt. Wenn keine Laufwerksnummer angegeben ist, beginnt das System die Suche nach einem freien FDE auf dem Laufwerk, das mit der Systemoption 'A0' angegeben wurde, und setzt die Suche auf den hoeheren Laufwerken fort, falls kein FDE gefunden wurde. Falls kein freier FDE gefunden wurde, kehrt die Routine mit dem Fehlerstatus zurueck.

Das Erzeugen eines Files besteht aus der Konvertierung eines FCB's in einen FPDE und beinhaltet folgendes:

- Name und Typ einsetzen
- Hash - Code der Passwoerter bilden (falls vorhanden)
- lrecl aus Register B einsetzen
- NEXT auf 0
- Zugriffsart auf 'FULL' setzen
- EOF auf 0 setzen
- Status des Files auf 'Nicht-System' und 'nicht unsichtbar'

Es wird auf der Floppy kein Platz reserviert fuer diesen File, es wird nur der Eintrag gemacht im Directory. DOS kontrolliert zu diesem Zeitpunkt nicht einmal, ob auf der Diskette auch noch Platz vorhanden ist. Der Hersteller gibt zwar an, dass die Recordlaenge im FPDE gespeichert wird, dies ist aber nach bisherigen Erfahrungen falsch. NEWDOS benutzt jedenfalls nicht mehr die Recordlaenge aus dem FPDE, sondern immer die Recordlaenge aus dem uebergeben B-Register.

Wurde ein Directory - Eintrag ueber diese Routine erzeugt, erfolgt ein Aufruf auf die Routine 4424H, um diesen File zu eroeffnen. Wurde das Erzeugen des Eintrags und das nachfolgende Eroeffnen erfolgreich abgeschlossen, erfolgt ein Ruecksprung mit dem Status Z und C.

3.14 4424H - OPEN FCB

Bereits existierenden File eroeffnen

Beim Einsprung in diese Routine muss DE die Adresse eines FCB's enthalten, in dem der Name des zu eroeffnenden Files steht, HL muss die Adresse eines 256 Bytes langen Puffers enthalten, auf den bei der Ein- und Ausgabe mit der Floppy zugegriffen wird. Das B-Register muss die Recordlaenge des Files enthalten (0=256!). Wurde im Filenamen eine Laufwerksnummer eingegeben, so wird die Suche nach diesem File auf dieses Laufwerk begrenzt, andernfalls faengt das System beim niedrigsten Laufwerk an und sucht, bis es diesen File gefunden hat.

Sind die Passwoerter aktiv und es wurde beim Filenamen weder ein Zugriffs- noch ein Updatepasswort angegeben, kommt es zu einer Fehlermeldung. Sind die Passwoerter nicht aktiviert oder der Filename enthaelt das Updatepasswort, wird die Zugriffsart des FCB auf 'FULL' gesetzt, jede Art von Lesen und Schreiben sind dann erlaubt. Wurde im Filenamen nur das Zugriffspasswort angegeben, wird die Zugriffsart des FCB entsprechend dem FPDE gesetzt.

Der Filename im FCB wird ueberschrieben mit der gelesenen Fileinformation, um weitere Zugriffe auf das Directory auf ein Minimum zu begrenzen. Folgende Informationen werden dem FPDE entnommen bzw. in den FCB gesetzt:

- EOF
- die ersten 4 Erweiterungs-Granules des FPDE's
- lrecl aus Register B
- NEXT auf 0
- Laufwerksnummer
- DEC setzen
- Pufferadresse aus HL
- Bit 5 des 2. Bytes auf 1 setzen (Puffer enthaelt nicht den momentanen Sektor)
- Bit 7 des 2. Bytes auf 1, wenn das B-Register <> 0 (Zugriff auf Records)
- Bit 7 des 2. Bytes auf 0, wenn das B-Register = 0
- Bit 7 des 1. Bytes auf 1 setzen (FCB ist eroeffnet)

Veraenderte Register siehe unter 3.1.a, b und c

3.15 4428H - CLOSE FCB

File mit zugehoerendem FCB schliessen.

Der zum FCB gehoerende File wird geschlossen, ein Zugriff ist dann nicht mehr moeglich. Wenn Bit 4 des 2. Bytes im FCB auf 1 gesetzt ist, wird der letzte Sektor noch automatisch ueber einen CALL auf 4439H auf die Floppy zurueckgeschrieben. Falls der EOF-Wert im FCB ungleich ist vom EOF-wert im FPDE, wird auch der FPDE neu ins Directory zurueckgeschrieben. Wurden vorher noch Granules fuer den File reserviert, die beim CLOSE hinter EOF liegen, werden diese Granules freigegeben. Der FCB erhaelt wieder seinen alten Inhalt mit Filenamen, Typ und Laufwerksnummer. Muss dieser File nochmals eroeffnet werden, kann der alte FCB erneut benutzt werden. Ein Passwort muss dann nicht mehr eingetragen werden, auch wenn dies fuer das System erforderlich waere.

Veraenderte Register siehe unter 3.1.a, b und c

3.16 442CH - KILL FCB

File loeschen

Der File, dem der File-Kontroll-Block (FCB) enthaelt, wird geloescht. Dies geschieht auf die gleiche Art wie das DOS - Kommando KILL. Der FCB wird komplett auf 0 gesetzt.

Veraenderte Register siehe unter 3.1.a, b und c

3.17 4430H - LOAD PROGRAM FILE

Der File, in dessen File-Kontroll-Block der Name steht und dessen Adresse im Registerpaar DE steht, wird geladen. Beim Rueck-sprung steht in HL die Startadresse des geladenen Programms ebenso in 4403H und in 4404H (17411 und 17412 dezimal). Diese Routine hat die gleiche Funktion wie das DOS-Kommando LOAD.

Veraenderte Register siehe unter 3.1.a, b und c, ausser AF, BC und HL.

3.18 4433H - LOAD AND EXECUTE FILE

Diese Routine hat zuerst die gleiche Funktion wie das LOAD (4403H), das geladene Programm wird dann aber gestartet. Das Registerpaar HL bleibt unveraendert, es koennen also Daten an das nachfolgende Programm uebergeben werden. Wenn beim OPEN oder LOAD ein Fehler auftritt, springt das DOS ueber 4409H zurueck.

3.19 4436H - READ FILE

Lesen eines Sektors oder Record vom FCB - Puffer in den Anwenderpuffer schreiben. Veraenderte Register siehe unter 3.1a, b und c.

Wenn das Bit 7 des 2. Bytes im File-Control-Block 0 ist, wird der Sektor, dessen Adresse in den 2 hoechsten Bytes des FCB's steht (NEXT-Feld), eingelesen. Wenn kein Fehler oder Fehlercode 6 (geschuetzter Sektor) auftritt, wird das NEXT- Feld um 256 erhoeht. Wenn ein anderer Fehler auftritt als Fehler 6 bleibt das NEXT-Feld auf dem gleichen Wert stehen, was bedeutet, dass das aufrufende Programm nochmals versuchen kann, diesen Sektor zu lesen.

Wenn das Bit 7 des 2. Bytes im File-Control-Block 1 ist, wird ein Record von der Laenge, die in lrecl steht, in den Anwenderpuffer (Adresse muss in HL stehen) geschrieben. Wenn alle Bytes im Anwenderpuffer stehen, wird das NEXT- Feld erhoeht. Wenn der FCB - Puffer leer ist, wird automatisch der naechste Sektor eingelesen und der eventuell noch vorhandene Rest in den Anwenderpuffer geschrieben. Wenn ein Fehler auftritt, auch bei Fehlercode 6, wird das Schreiben in den Anwenderpuffer abgebrochen, das NEXT-Feld wird um die Anzahl der bisher geschriebenen Bytes erhoeht.

Wenn das Bit 1 des 1. Bytes im File-Control-Block 1 ist, wird angenommen, dass das NEXT- und das EOF-Feld RBA's innerhalb einer Diskette und nicht eines Files enthalten. Auf diese Art kann anstelle eines Files direkt auf der Diskette adressiert werden. Gleiches geschieht, wenn das Bit 2 dieses Bytes 1 ist, nur wird dann angenommen, dass im mittleren Byte des

FCB-Blockes die relative Sektornummer einer Spur, und im hoechsten Byte dieses Blockes die Spur innerhalb der Diskette steht. Der Gebrauch von Bit 0 in diesem Byte ist in 4439H beschrieben. Die DOS - Routinen 0013H, 001BH, 4439H, 443CH und andere Routinen, die sektorweise mit der Floppy arbeiten koennen, arbeiten mit diesen 3 Bits. Die Benutzung dieser 3 Bits ist anders als im TRSDOS.

3.20 4439H - WRITE WITHOUT VERIFY

Schreiben eines Sektors auf die Diskette oder einen Record aus dem Anwenderpuffer in den FCB-Puffer schreiben. Die Konditionen 3.1.a, b und c werden gehalten.

Wenn das Bit 7 des 2. Bytes im File-Control-Block 0 ist, wird der FCB-Puffer in den Disksektor geschrieben, dessen Adresse im NEXT-Feld steht. Ein Verify wird nicht gemacht, solange nicht VERIFY,ON aktiv ist. Wenn kein Fehler auftritt und das niedrigste Byte von NEXT = 0 ist, wird das NEXT-Feld um 256 erhoeht. Wenn Bit 6 des 2. Bytes 0 ist oder NEXT den Wert EOF ueberschreitet, egal ob NEXT erhoeht wurde oder nicht, wird das EOF auf NEXT gesetzt. Wenn ein Fehler auftritt, wird NEXT nicht veraendert, der Anwender kann das Schreiben auf diesen Sektor nochmals versuchen.

Wenn das Bit 7 des 2. Bytes im File-Control-Block 1 ist, wird ein Record von der Laenge, die in lrecl steht, vom Anwenderpuffer (Adresse muss in HL stehen) in den FCB - Puffer geschrieben. Bei jedem Byte, das uebertragen wird, wird das NEXT-Feld erhoeht. Wenn NEXT groesser wird als EOF oder wenn Bit 6 des 2. FCB-Bytes 0 ist, wird EOF auf NEXT gesetzt. Sobald der FCB-Puffer voll ist, wird er auf die Floppy geschrieben, die Uebertragung vom Anwender zum FCB-Puffer wird dann fortgesetzt. Sobald ein Fehler auftritt, wird der Uebertrag beendet, das NEXT-Feld bleibt auf dem momentanen Wert stehen. Das Schreiben auf die Floppy erfolgt mit Verify.

Die Funktion der Bits 1 und 2 des 1. Bytes im FCB wurden bereits in 3.18 beschrieben. Wenn Bit 0 dieses Bytes 1 ist, wird der geschriebene Sektor geschuetzt (Fehler 6 beim Lesen).

Nach einem Kontrollesen von einem geschuetzten Sektor kann kein Fehler 6 auftreten !

Der Unterschied in dieser Routine zum TRSDOS liegt darin, dass, wenn ein Sektor im NEWDOS/80 auf die Floppy geschrieben wird und das niedrigste Byte des NEXT-Feldes ungleich null ist, das NEWDOS/80 das NEXT-Feld nicht um 256 erhoeht. In diesem Fall nimmt NEWDOS/80 an, dass der Anwender den letzten Sektor auf die Floppy schreibt, der aber nur teilweise gefuellt ist, und dass das NEXT-Feld bereits den richtigen RBA-Wert fuer das EOF hat (falls EOF erhoeht wird beim Schreiben).

3.21 443CH - WIE 4439H MIT VERIFY

Diese Routine hat die gleiche Funktion wie 4439H, ausser dass nach dem Schreiben ein Verify erfolgt.

3.22 443FH - POSITION AT START OF FILE

File Control Block (FCB) auf Fileanfang positionieren. Die Konditionen 3.1.a, b und c werden gehalten. Falls im Puffer

dieses Files ein Sektor steht, der noch auf die Floppy geschrieben werden soll (angezeigt durch Bit 4 des 2. FCB - Bytes auf 1), so wird dieses Schreiben ausgefuehrt wie ueber die Routine 4439H. Das FCB NEXT-Feld wird auf 0 gesetzt. Bit 5 des 2. FCB-Bytes wird geloescht um anzuzeigen, dass der momentane Puffer nicht den aktuellen Sektor enthaelt. Zuletzt wird noch das Laufwerk selektiert.

3.23 4442H - POSITION AT SPEC. RECORD

Der FCB wird auf einen bestimmten Record positioniert. Prinzipiell gleiche Funktion wie die Routine 443FH mit dem Unterschied, dass das NEXT-Feld des FCB auf 'RBA' gesetzt wird, die relative Recordnummer muss in BC stehen.

3.24 4445H - POSITION FCB 1 RECORD BACK

Der FCB wird um einen Record zurueckgesetzt. Prinzipiell gleiche Funktion und gleiche Konditionen wie 443FH, nur dass das NEXT- Feld um die Recordlaenge (lrecl) vermindert wird.

3.25 4448H - POSITION FCB TO EOF

FCB auf End Of File positionieren. Gleiche Funktion und gleiche Konditionen wie 443FH mit dem Unterschied, dass das NEXT-Feld auf das EOF-Feld gesetzt wird.

3.26 444BH - ALLOCATE FILE SPACE

Platz fuer File reservieren. Die Konditionen 3.1.a, b und c werden gehalten. Falls der Sektor, dessen Wert in den 2 hoechsten Bytes des FCB NEXT-Feldes steht, nicht bereits fuer diesen File reserviert wurde, wird dies damit getan, gleichzeitig werden auch alle darunter liegenden Sektoren bzw. Granules fuer diesen File reserviert. Dadurch kann der Programmierer bereits im voraus Platz fuer den File reservieren und auch auf die Diskette schrieben, bevor in den Puffer geschrieben wird. Diese Vorreservierung kann u. U. viel Zeit sparen. Sie wird unter anderem auch beim COPY benutzt.

ACHTUNG! Diese Routine unterscheidet sich vom TRSDOS!

3.27 444EH - POSITION FCB TO RBA

FCB auf RBA-Adresse positionieren.

Konditionen und Ausfuehrung sind im Prinzip gleich wie bei der Routine 4442H, nur dass der Wert des RBA aus den Reistern H, L und C entnommen wird. In H steht das hoechstwerige Byte der neuen Adresse, in C das niedigstwertige Byte.

Die Adresse dieser Routine ist unterschiedlich belegt im TRSDOS.

3.28 4451H - WRITE EOF TO DIRECTORY

EOF-Wert aus FCB ins Directory schreiben.

Konditionen A, B und C werden gehalten. Weicht der Wert von EOF im Directory von dem Wert im FCB ab, wird der neue Wert aus dem FCB in den FPDE des Directorys eingetragen.

Die Adresse dieser Routine ist unterschiedlich belegt im TRSDOS.

3.29 445BH - SELECT SPEC. DRIVE

Angegebenes Laufwerk starten und ansprechen.

Konditionen A und B werden gehalten. Beim Ansprung muss der Accu die Nummer des Laufwerks enthalten, das dann ggf. gestartet wird.

3.30 445EH - TEST FOR MOUNTED DISKETTE

Ueberpruefung, ob Diskette im Laufwerk ist.

Konditionen und Abarbeitung im Prinzip gleich wie bei der Routine 445BH, mit dem Unterschied, dass das Laufwerk kontrolliert wird, ob die Diskette rotiert. Wenn nicht, erhaelt man im Accu die Fehlernummer 8, 'DEVICE NOT AVAILABLE'.

3.31 4461H - CHAIN USER ROUTINES

Anwenderprogramme in eine Eingabewarteschlange setzen.

In NEWDOS/80 besteht die Moeglichkeit, eine Eingabewarteschlange fuer Anwenderprogramme zu bilden. Diese Programme stehen im Speicher und werden erst dann gestartet, wenn eine Eingabe an sie erfolgt. Die Eingabe muss dann folgendes Format haben: *FNAME oder *FNAME,Parameter.

Mit einem CALL auf 4461H wird ein Programm in diese Warteschlange eingereiht, im Registerpaar HL muss die Adresse dieses Programms stehen. Der Programmanfang muss folgendes Format haben:

4 Bytes	Reserviert fuer DOS
8 Bytes	Der Name, unter dem es bei der Eingabe angesprochen werden soll (bei *.....). Der Name muss linksbuendig in diesem Feld stehen, wenn er kuerzer ist, muss rechts mit Leerzeichen aufgefuellt werden.

Existiert dieser Name bereits in der Warteschlange, wird die Fehlernummer von 'FILE ALREADY EXISTS' im A-Register mit dem Status NZ uebergeben. Bei erfolgreichem Abschluss wird das Zeroflag gesetzt. HL, DE, BC und AF werden veraendert.

Wird nun im DOS das Kommando *FNAME aufgerufen, sucht es die Warteschlange durch und springt, wenn es diesen Namen gefunden hat, auf das 13. Byte dieser Routine. Die Routine muss abschliessen ueber 402DH, 4409H oder 4030H. Fuer eine Parameteruebergabe an andere Programme stehen die Bytes 4403H und 4404H zur Verfuegung. Wurde die Routine mit dem Namen FNAME noch nicht in die Warteschlange gestellt, aber aufgerufen

ueber *FNAME, wird der Fehler 'FILE NOT IN DIRECTORY' angezeigt.

Diese Routine ist neu im NEWDOS/80

3.32 4464H - UNCHAIN USER ROUTINES

Routine bzw. Anwenderprogramm aus der Eingabewarteschlange herausnehmen.

Ein Programm bzw. eine Routine, die ueber 4461H in die Eingabewarteschlange eingefuegt wurde, kann mit 4464H wieder aus dieser Warteschlange herausgenommen werden. HL muss genau wie bei 4461H die Adresse dieses Programms enthalten. Ist der Name nicht in dieser Warteschlange, erfolgt ein Ruecksprung mit NZ und der Fehlernummer von 'FILE NOT IN DIRECTORY' im A-Register. HL, DE, BC und AF aendern sich. Bei einem weiteren Aufruf von *FNAME wird dann der Fehler 'FILE NOT IN DIRECTORY' angezeigt.

Diese Routine ist neu im NEWDOS/80

3.33 4467H - SEND MESSAGE TO DISPLAY

Meldung auf dem Bildschirm ausgeben

Die Meldung, mit deren Adresse HL geladen ist, wird auf dem Bildschirm ausgegeben. Diese Meldung kann entweder mit 0DH abschliessen (wird mit Return ausgegeben) oder mit 03H (wird ohne Return ausgegeben).

Alle Register ausser AF bleiben unveraendert.

3.34 446AH - SEND MESSAGE TO PRINTER

Meldung auf dem Drucker ausgeben

Gleiche Funktion wie die Routine 4467H, nur dass die Meldung auf dem Drucker ausgegeben wird.

3.35 446DH - CONVERT CLOCK

Konvertierung der Uhrzeit

Die momentane Uhrzeit in den Adressen 4041H - 4043H wird in Ziffern umgesetzt und in einen Buffer geschrieben, dessen Adresse in HL stehen muss. Der Buffer muss 8 Bytes lang sein.

Die Register AF, BC, DE und HL werden veraendert, HL steht auf dem 9. Byte nach dem Bufferanfang.

3.36 4470H - CONVERT DATE

Datum konvertieren

Prinzipiell die gleiche Routine wie 446DH, ausser dass das Datum aus den Adressen 4044H - 4046H in Ziffern umgesetzt wird.

3.37 4473H - INSERT DEFAULT INTO FILESPEC.

Typ in Filenamen einsetzen

Das Registerpaar DE enthaelt die Adresse eines Filenamens. Wenn dieser Name ohne Typangabe ist, kann man ueber diese Routine den Typ einsetzen. Registerpaar HL muss die Adresse des 3 Bytes langen Typs enthalten. Der erzeugte Name kann nicht laenger sein als 31 Bytes.

Register AF und HL werden veraendert.

3.38 0013H - READ BYTE FROM DISK

Einzelnes Byte von der Floppy lesen.

Die Startadresse dieser Routine liegt im ROM. Falls der Sektor der Floppy, in dem dieses Byte steht, noch nicht eingelesen wurde, wird er eingelesen und der File-Kontroll-Block (FCB) entsprechend veraendert. Sonst wird nur der Zaehler fuer das naechste Byte erhoeht. Das gelesene Byte steht im A-Register.

Veraenderte Register siehe unter 3.1.a, b und c

3.39 001BH - WRITE BYTE TO DISK

Byte auf Floppy schreiben

Das im A-Register uebergebene Byte wird, falls der entsprechende Sektor bereits im Buffer steht, in diesen Buffer eingesetzt. Steht dieser Sektor noch nicht im Buffer, wird dieser erst eingelesen. Falls dieses eingesetzte Byte das letzte Byte im Sektorbuffer ist oder der Zaehler am Fileende steht, wird dieser Buffer auf die Floppy zurueckgeschrieben wie bei einem Aufruf der Routine 443CH. Andernfalls wird nur der Zaehler erhoeht.

Veraenderte Register siehe unter 3.1.a, b und c

3.40 447BH - NUR MODELL III

Diese Subroutine hat im Modell III die gleiche Funktion wie die 4410H bei Modell I. Die Routine 4410H darf bei Modell III nicht aufgerufen werden.

4.0 DOS-FEATURES

In diesem Kapitel werden DEBUG, MINI-DOS, CHAINING, DOS-CALL, JKL und die asynchrone Ausfuehrung von Programmen beschrieben. Dabei duerften DEBUG , DOS-CALL und die asynchrone Ausfuehrung von Programmen hauptsaechlich fuer Programmierer interessant sein, die in Maschinensprache programmieren. MINI-DOS und JKL duerften fuer alle Anwender interessant sein. Vor allem das CHAINING kann sehr komplex sein, Anfaenger sollten sich langsam darin einarbeiten und vor allem das Testprogramm CHAINTST/JCL zusammen mit dem BASIC-Programm CHAINBLD/BAS durcharbeiten.

4.1 DEBUG

Als Hilfe fuer Benutzer, die mit Z80 Maschinenprogrammen und auch mit anderen Programmen einer hoeheren Programmiersprache arbeiten, hat das NEWDOS/80 die Moeglichkeit zur Benutzung des DEBUG.

- Damit koennen laufende Programme unterbrochen werden um den Speicherinhalt zu ueberpruefen oder zu aendern,
- Die Floppy kann ueberprueft oder der Dateninhalt veraendert werden, oder
- Einzelschrittausfuehrung kann angestossen werden.

In das DEBUG kommt man entweder durch gleichzeitiges Druecken der Tasten 1, 2 und 3 (wenn die Systemoption AE=Y ist) oder durch Druecken der Break-Taste (Systemoption AE=N, DEBUG muss aktiviert sein durch das DOS-Kommando DEBUG), wenn ein NEWDOS-Overlayprogramm nicht aktiv ist, die Interrupts eingeschaltet sind oder das Programm auf eine Eingabe von der Tastatur (ueber den Tastatur-Kontrollblock und Systemoption AJ=Y). DEBUG wird ebenfalls aktiv, wenn ein Breakpoint erreicht ist (Z80-Befehl RST 30H) oder das ausgefuehrte Programm einen CALL oder JUMP auf die Subroutine 440dh ausfuehrt. Wenn DEBUG ueber das DOS-Kommando 'DEBUG' aktiviert wurde, wird es ebenfalls falls bei jedem Programmaufruf (wie BASIC, EDTASM ....) angestossen.

Beim Einsprung in die DEBUG-Routine werden

- Alle Register im Benutzer-Stack gerettet
- Das Stack wird fuer DEBUG benutzt
- Alle Breakpoints geloescht
- Das Memory angezeigt wie bei vorhergehenden Operationen (X oder S)
- Der Cursor in die linke untere Ecke des Bildschirms gestellt um anzuzeigen, dass das DEBUG auf eine Eingabe wartet.

Alle Kommandos, auch die, welche nur aus einem Buchstaben bestehen, muessen durch Druecken der 'ENTER'-Taste abgeschlossen werden. Ein Kommando kann also, wenn es falsch eingegeben wurde, durch die Backspace-Taste (linker Pfeil) korrigiert werden. Gegebenenfalls kann die ganze Zeile durch Druecken von Shift und Backspace geloescht werden.

Die Anzeige des Memories erfolgt sowohl bei der Anzeigeart X als auch S als hexadezimaler Wert und als ASCII-Zeichen. Wenn die Systemoption AI=Y ist, auch in Kleinbuchstaben (sofern vorhanden).

Wird ein Kommando fehlerhaft eingegeben, erscheint 'ERROR' auf dem Bildschirm und das DEBUG wartet, bis die 'ENTER'-Taste gedrueckt wird.

Folgende Kommandos koennen in DEBUG benutzt werden :

X Das DEBUG geht in den X-Anzeigemodus. Dieser sieht folgendermassen aus:

Anzeigeformat 15 Zeilen, die ersten vier Zeilen enthalten die ersten 64 Bytes des Memories , Zeile 5 enthaelt die aktuellen Werte der Register AF, BC, DE und HL, die Zeilen 6 bis 9 enthalten die naechsten 64 Bytes des Memories, in Zeile 10 werden die aktuellen Registerstaende der Z80-Register AF', BC', DE' und HL' angezeigt, in den Zeilen 11 bis 14 kommen die naechsten 64 Bytes des Memories, Zeile 15 enthaelt die Register PC, SP, IX und IY. Die Anzeige von AF und AF' erfolgt auch als Bitmaske fuer das Statusbyte F, die Bits werden als Buchstaben angezeigt, wenn sie gesetzt sind, und als '-', wenn sie nicht gesetzt sind. Folgende Buchstaben werden benutzt :

Bit 7 - S fuer minus (sign)

Bit 6 - Z fuer 0 (zero)

Bit 5 - nicht benutzt im Z80

Bit 4 - H Digit- oder halber Uebertrag (half carry)

Bit 3 - nicht benutzt im Z80

Bit 2 - P fuer gerade Paritaet oder Ueberlauf (Parity / Overflow)

Bit 1 - N fuer Subtraktion

Bit 0 - C fuer Uebertrag (carry)

In der Anzeigeart X kann der Benutzer 3 verschiedene Speicherbereiche gleichzeitig anzeigen.

S Das DEBUG geht in den S-Anzeigemodus. In dieser Anzeigeart werden 256 Bytes angezeigt, die Startadresse ist die Adresse des 1. Anzeigebereichs der X-Anzeige, die Adresse wird auf eine durch 256 teilbare Zahl abgerundet (endet mit 00). Die Anzeigelaenge ist 16 Zeilen.

<n>Dx Wenn sich die Anzeige im S-Modus befindet wird der 256-Byte-Block angezeigt, in dem sich die Adresse x befindet. Wenn n mit eingegeben wird, wird nur intern die Adresse geaendert, aber erst angezeigt, wenn das DEBUG sich im S-Modus befindet. Im X-Modus kann ueber die Eingabe von n der Bereich der Anzeige angegeben werden (1., 2. oder 3. Drittel). Als Defaultwert fuer n wird 1 angenommen.

<n>; Wenn sich die Anzeige im S-Modus befindet, wird der naechste 256-Byte-Block angezeigt, im X-Modus der naechste Block mit 64 Bytes im Bereich n. Wenn n im X-Modus nicht angegeben ist, wird Bereich 1 angenommen.

<n>- Wenn sich die Anzeige im S-Modus befindet, wird der vorhergehende 256-Byte-Block angezeigt, im X-Modus der vorhergehende Block mit 64 Bytes im Bereich n.

Wenn n im X-Modus nicht angegeben ist, wird Bereich 1 angenommen.

Mx Aendern der Adresse x. DEBUG springt in den S-Anzeigemodus und zeigt den 256-Bytes-Block an, in dem sich die Adresse x befindet. Der Cursor steht auf dem Byte, das geaendert werden soll. Durch Eingabe von 0-9 oder A-F kann das entsprechende Byte geaendert werden, bei Eingabe von Right Arrow (Pfeil nach rechts) springt der Cursor auf die naechste Position, ohne dieses Byte zu aendern, bei Left Arrow (Pfeil nach links) auf die vorhergehende Position, bei Shift ... Arrow jeweils um 4 Positionen, ohne das Memory zu aendern. Bei Up Arrow wird auf die vorhergehende, bei Down Arrow auf die naechste Zeile ohne Aenderung gewechselt. Durch Druecken der 'ENTER'- Taste wird die Routine M verlassen, bei jeder anderen Taste wird ein Fehler angezeigt. Der angezeigte Block mit 256 Bytes kann nicht verlassen werden.

F<x><,b1><,b2><,b3><,b4> Suchen eines bestimmten Bytes oder einer Folge aus max. 4 Bytes im Memory ab der Startadresse x. b1-b4 bestehen aus jeweils 2 Buchstaben bzw. Ziffern fuer 1 Byte. Falls ein oder mehrere Bytes angegeben werden, muss auch die Adresse x mit eingegeben werden. Werden weder Adresse noch Bytes mit eingegeben, beginnt die Suche bei der letzten benutzten Adresse+1 dieses Kommandos. Dies erlaubt es dem Benutzer, ein bestimmtes Byte oder eine Folge von Bytes zu finden, auch wenn diese mehrfach im Memory vorhanden sind. Sobald DEBUG das gesuchte Byte oder die Bytefolge im Memory gefunden hat, wird der entsprechende Bereich im X-Format am Bildschirm angezeigt, die Startadresse ist 20H tiefer, das gesuchte Byte steht also in der 3. Zeile. Wird kein Byte oder keine Bytefolge gefunden, wird das Memory ab der Adresse 0FFE0H angezeigt.

Beispiel : **F5200,01,23,45**

Dieses Kommado bewirkt eine Suche nach der Bytefolge 01 23 45, die Startadresse fuer die Suche ist 5200H.

Folgendes ist zu beachten :

Es kann sein, dass die gesuchte Sequenz sich im Stackbereich befindet und bereits geloescht ist, bevor dieser Bereich angezeigt wird.

Das bzw. die gesuchten Bytes werden bei der Adresse im Bereich um 51..H vermutet. Es kann dann vorkommen, dass man, wenn man die Suche bei 0 oder unter 5100H gestartet hat, als erstes die eigentlich zu suchenden Bytes im Bereich von DEBUG findet.

I Einzelschrittausfuehrung. Der naechste Befehl des durch DEBUG unterbrochenen Programms wird ausgefuehrt. Der Benutzer hat dann die Moeglichkeit, sich das Memory oder die Registerstaende anzusehen.

Folgendes ist besonders bei der Einzelschrittausfuehrung zu beachten :

1. Eine volle Interrupt-Routine kann in dieser Zeit ablaufen.

2. Einzelschrittausfuehrung ist nicht erlaubt bei Adressen unter 5200H, ebenso ist auch ein Sprung in den Bereich unter 5200H nicht erlaubt.

3. Die DEBUG-Routine benutzt fuer die Unterbrechung den Z80-Befehl RST 30H fuer den Sprung zurueck ins DEBUG. Aus diesem Grunde darf das zu testende Programm keinen Sprung auf sich selbst enthalten, und das naechste Byte beim Einzelschritt darf kein Datenbyte sein.

C Gleiche Funktion wie 'I', jedoch mit dem Unterschied, dass ein CALL auf eine Subroutine komplett ohne Unterbrechung ausgefuehrt wird.

Rzz xxxx Das Registerpaar zz wird auf den Wert xxxx gesetzt. Es darf nur ein Leerzeichen zwischen dem Registerpaar und dem Wert stehen.

Beispiel: RDE' 67AB

Der Wert des Registerpaares DE' wird auf 67ABH gesetzt.

Ldn1,drs1 Der relative Sektor drs1 der Diskette dn1 wird in den Buffer von 4200H bis 42FFH eingelesen. Der Benutzer ist selbst fuer die korrekten hexadezimalen Werte von dn1 und drs1 verantwortlich, da das NEWDOS/80 keinen dieser Werte ueberprueft. Wenn der Sektor in den Buffer eingelesen ist, kann der Anwender diesen Buffer anzeigen ueber X oder S, Bytes oder Strings suchen ueber F, oder auch ueber M den Buffer veraendern. Es wird jedoch nur der Buffer und nicht der Sektor auf der Diskette veraendert. Soll der veraenderte Buffer auf die Diskette zurueckgeschrieben werden, muss das Kommando W benutzt werden. Da alle Systemprogramme des NEWDOS/80 den gleichen Buffer fuer die Ein/Ausgabe auf die Diskette benutzen, sollten die Kommandos L und W nicht angewandt werden, wenn ein Interrupt durch das DOS erfolgt ist (speziell bei FORMAT und COPY).

ACHTUNG: Wenn die Passwoerter benutzt werden, erfolgt bei den Kommandos L und W eine Fehlermeldung.

Wdn1,drs1 Der Speicherbereich von 4200H bis 42FFH wird auf die Diskette dn1 auf den relativen Sektor drs1 geschrieben. (bei Modell III liegt der Buffer bei 4300H - 43FFH) Es gelten die gleichen Definitionen wie fuer das Kommando L. Wenn der entsprechende Sektor geschuetzt ist, wird auch geschuetzt zurueckgeschrieben.

Q Das DEBUG wird beendet und die Ausfuehrung des DOS gestartet. Das aufrufende Programm wird vergessen, ebenso ggf. das MINI-DOS. Es wird direkt 'DOS-READY' angezeigt.

G<adr1><,adr2><,adr3> Die Werte der Register werden in die Register zurueckgespeichert und das Programm an der Stelle fortgefuehrt, an der es unterbrochen wurde. Wurde der Wert adr1 eingegeben, wird das Programm an dieser Stelle gestartet, andernfalls an der Stelle, die unter PC angezeigt wurde. Mit adr2 und adr3 koennen 2 Adressen angegeben werden, an denen das Programm angehalten werden soll. An diesen Stellen wird der Z80 1-Byte-Befehl RST 30H eingesetzt, der urspruengliche Wert zwischengespeichert. Sobald das Programm an wieder in das DEBUG zurueckkehrt, werden die alten Werte wieder zurueckgeschrieben. Wird

adr2 angegeben, kann adr1 weggelassen werden. Beispiel :

G7000,8400,89AB

Setzt je einen Breakpoint bei der Adresse 8400H und 89ABH, speichert die urspruenglichen Registerdaten zurueck und springt auf die Adresse 7000H.

G

Speichert die urspruenglichen Registerdaten zurueck und springt auf die Adresse, die unter PC angezeigt wird.

Wartet ein Programm zur Zeit des Eintritts in das DEBUG auf eine Eingabe, so muss die Eingabe nach Ausfuehrung des Kommandos G gemacht werden, auch wenn kein Cursor mehr anzeigt wird.

G402D

Beendet das unterbrochene Programm und kehrt in das DOS zurueck.

Der Aufruf ueber die Tastenfunktion '123' (Systemoption AE=Y) wurde aus dem Grunde eingefuehrt, weil dadurch die Break-Taste fuer ihre urspruengliche Funktion wieder freigegeben wurde. '123' hat ebenso den Vorteil, dass das DEBUG nicht zuerst ueber das DOS-Kommando DEBUG freigegeben werden muss.

4.2 MINI-DOS

Es gibt viele Faelle, in denen man, waehrend ein Anwenderprogramm laeuft, ein oder mehrere DOS-Kommandos ausfuehren moechte, ohne das laufende Programm abzubrechen und etwas daran zu aendern. In NEWDOS/80 wurde die Moeglichkeit dazu durch Einfuehrung des sog. MINI-DOS geschaffen.

Folgende Voraussetzungen sind fuer den Gebrauch von MINI-DOS zu erfuellen:

1. Systemoption AB=N
2. Systemoption AF=Y
3. Entweder muessen die Interrupts eingeschaltet sein, oder das laufende Programm erwartet eine Eingabe von der Tastatur und die Systemoption AJ=Y.

Sind diese Funktionen erfuellt, besteht die Moeglichkeit, durch gleichzeitiges Druecken der Tasten D, F und G das MINI-DOS zu starten. Das momentan laufende Programm wird

unterbrochen und die Registerstaende gerettet, 'MINI-DOS READY' wird auf dem Bildschirm angezeigt.

ACHTUNG !!! Waehrend gerade auf die Floppy zugegriffen wird, ist es gefaehrlich, MINI-DOS aufzurufen. Es besteht die Moeglichkeit eines Schreibfehlers! Nur wenn MINI-DOS mit dem Kommando MDBORT beendet wird, besteht diese Gefahr nicht.

Im MINI-DOS Status kann der Benutzer alle DOS-Kommandos ausser APPEND, CHAIN, COPY, FORMAT, PDRIVE und SYSTEM aufrufen. Andere Programme sollten nicht aufgerufen werden.

Das MINI-DOS wird beendet durch das Kommando MDRET. Die Register werden wieder zurueckgespeichert und die CPU faehrt an der Stelle fort, wo die Unterbrechung erfolgt ist. Der Cursor wird auf dem Bildschirm angezeigt, egal, ob das gerade laufende bzw. unterbrochene Programm eine Eingabe erwartet oder nicht. Hat das Programm beim Aufruf von MINI-DOS eine Eingabe erwartet, so ist jetzt diese Eingabe zu machen, auch wenn nicht mehr der alte Bildschirminhalt angezeigt wird und der Cursor jetzt an einer anderen Position steht.

Soll das unterbrochene Programm nicht mehr fortgefuehrt werden, so kann man direkt ueber das Kommando MDBORT in das NEWDOS/80 zurueckkehren. Am Bildschirm wird dann 'DOS READY' angezeigt.

Da das Kommando COPY nicht unter MINI-DOS ausgefuehrt werden kann, steht dem Benutzer MDCOPY zur Verfuegung. MDCOPY kopiert jedoch nur sektorweise.

Es besteht im NEWDOS/80 beim Aufruf von MINI-DOS durch gleichzeitiges Druecken der Tasten 'DFG' die Moeglichkeit, dass eine oder mehrere dieser Tasten noch am Bildschirm angezeigt werden, speziell bei der Systemoption AJ=N. Der Benutzer kann diese zusaetzlichen Buchstaben einfach loeschen durch Backspace.

Beispiel fuer die Anwendung von MINI-DOS:

Geben Sie im DOS-READY folgendes ein:

```
BASIC
10 PRINT "HELLO" : GOTO 10
RUN
```

Jetzt laeuft ein BASIC-Programm, das staendig HELLO auf dem Bildschirm ausgibt. Druecken Sie jetzt gleichzeitig die Tasten D, E und F. Das BASIC-Programm wird damit unterbrochen und die Meldung MINI-NEWDOS/80 READY wird auf dem Bildschirm angezeigt. Geben Sie jetzt folgende DOS-Kommandos ein:

```
DIR 0
FREE
CLOCK
CLOCK,N
LIB
SYSTEM,0
PDRIVE,0
MDRET
```

Das Kommando MDRET bewirkt, dass das MINI-DOS beendet wird und das BASIC-Programm an der Stelle fortgesetzt wird, an der es unterbrochen wurde. Jetzt kann man auch, waehrend des laufenden Testprogrammes, das DEBUG ausprobieren. Druecken Sie gleichzeitig die Tasten 1, 2 und 3. Das Laufende BASIC-Programm wird wieder unterbrochen und das DEBUG gestartet. Auf dem Bildschirm wird, je nach Anzeigeart, das Memory und die Register im

X- oder S-Format angezeigt. Geben Sie G ein und druecken Sie die ENTER-Taste. Das BASIC-Programm wird wieder fortgesetzt. Gehen Sie jetzt nochmals ins MINI-DOS.

4.3 CHAINING

Die Kommandos CHAIN und DO unterscheiden sich ausschliesslich in ihrer Schreibweise, nicht aber in ihrer Funktion voneinander. Hier in der Beschreibung wird nur vom Kommando CHAIN die Rede sein, gleiches gilt jedoch auch fuer die Schreibweise DO.

Das NEWDOS/80 hat die Moeglichkeit, anstelle einer Eingabe von der Tastatur ueber den Standardtreiber eine Eingabe von einem Disk-File im Chaining-Modus zu uebernehmen. Der Benutzer hat die Moeglichkeit, eine beliebige Kommandofolge in einen ASCII-File zu schreiben und diesen ueber das Chain-Kommando aufzurufen und ausfuehren zu lassen (siehe "CHAIN" auf Seite 10).

Nehmen wir einmal an, dass der Benutzer nach dem Einschalten des Systems immer die gleiche Kommandofolge einzugeben hat. Als Beispiel:

```
HIMEM,0E800H                DOS-Kommando HIMEM
PROGRAM1                    ausfuehren Programm 1
Y                           1. Antwort auf Programm 1
50                          2. Antwort auf Programm 1
PROGRAM2                    nachdem Programm 1 fertig
                            ist, Programm 2 aufrufen
1                           1. Antwort auf Programm 2
WORKF1                      2. Antwort auf Programm 2
WORKF2                      3. Antwort auf Programm 2
BASIC,RUN"BASPGM1/BAS"      nachdem Programm 2 fertig
                            ist, BASIC-Programm aufrufen
Y                           Antwort auf das BASIC-Programm
```

Weitere Eingaben sind von Start zu Start unterschiedlich. Dies ist aber fuer unser Beispiel nicht massgeblich. Massgeblich ist im Moment nur, dass beim Start bis zu der letzten dargestellten Eingabe immer der gleiche Ablauf und die gleichen Eingaben erfolgen.

Waere nun diese Kommandofolge auf dem File XXX/JCL geschrieben, so braeuchte man nur das Kommando

CHAIN,XXX/JCL bzw. DO,XXX

einzugeben. Dies hat zu Folge, dass von der Tastatureingabe umgeschaltet wird auf eine Fileeingabe, in diesem Falle von dem File XXX/JCL. Sobald von der Tastatur eine Eingabe angefordert wird, wird der naechste String von dem File geholt und in den Eingabebuffer des DOS gestellt, und zwar so lange, bis das Ende des Files erreicht wird. Hier in unserem Beispiel braucht der Benutzer absolut nichts zu tun, bis das Programm BASPGM1 seine 2. Eingabe erwartet.

Um diesen File aber schon beim System-Start aufzurufen, muss der Aufruf des Chain-Files beim Start mit dem Kommando

AUTO,CHAIN,XXX/JCL

angegeben werden. Nun wird jedes Mal beim Einschalten oder beim Reset dieser File durchlaufen.

Wird das Chain-Kommando ausgefuehrt, geht das NEWDOS/80 in den sog. Chaining-Modus (falls nicht bereits darin) und liest

anstelle der Tastatureingabe Zeile fuer Zeile von dem angegebenen File. Jede Zeile in diesem File muss mit 0DH (Return) abgeschlossen sein. Dazu eroeffnet NEWDOS/80 zuerst diesen File und laesst die Positionierung auf der Startzeile stehen, wenn ein Abschnitt angegeben wurde, wird dieser File durchsucht, bis der betreffende Abschnittsname gefunden wurde und dort positioniert.

In diesem Falle wird, sobald eine Eingabe, egal ob von NEWDOS/80, BASIC oder einem anderen Programm, verlangt wird, der naechste Record von diesem File gelesen und in den Eingabepuffer gestellt.

Wenn die uebergebene Zeile aus dem Chain-File nicht mit einem EOL-Zeichen (0DH) abschliesst, verlaesst das NEWDOS/80 den Chain-Modus und erwartet die Eingabe von der Tastatur. Es wird dabei keine Meldung ausgegeben, das System haengt nur in der Eingabeschleife und wartet. Das gleiche geschieht auch, wenn beim Aufruf des Chain-Files kein Abschnittsname eingegeben wird, der File jedoch mit einem Abschnittskennzeichen beginnt.

Wenn die gerade gelesene Zeile mit einem EOF-Zeichen abschliesst oder diese Zeile bereits die Kennung fuer den naechsten Abschnitt enthaelt, wird der Chaining-Modus abgebrochen. Dies ist die normale Beendigung des Chaining-Modus.

Das Chaining arbeitet auf 2 unterschiedliche Arten, die von der System-Option AT abhaengig sind.

AT=Y bedeutet, dass das alle Eingabemoeglichkeiten von der Tastatur auch fuer den Chain-File zulaessig sind, wie z. B. in BASIC das Kommando "LINEINPUT" oder "INPUT", aber auch einzelne Zeichen, wie sie z. B. bei "INKEY$" eingegeben werden.

AT=N bedeutet, dass vom Chain-File nur volle Records, die mit ENTER (0DH) abschliessen, akzeptiert werden. Die entspricht den Kommandos "LINEINPUT" und "INPUT" vom BASIC. Einzelne Character, wie sie z. B. beim BASIC-Kommando "INKEY$" eingegeben werden, muss man direkt ueber die Tastatur eingeben.

Im Chaining gibt es fuenf verschiedene Methoden, den Ablauf innerhalb des Chain-Files zu aendern.

1. Das momentan ausgefuehrte Programm macht einen DOS-CALL mit dem Kommando CHAIN, DO oder CHNON (z. B. von BASIC aus mit CMD"xxx").

2. Der auszufuehrende Befehl im Chain-File ist selbst ein CHAIN- (oder DO-) Kommando. Dabei muss allerdings das naechste Kommando im Chain-File angefordert werden. Die andere Moeglichkeit waere die, dass ein gerade ausgefuehrtes Programm selbst intelligent genug ist zu erkennen, dass es von einem Chain-File aufgerufen wurde und nun selbst einen anderen Chain-File ueber DOS-CALL (z. B. von BASIC aus mit CMD"xxx") startet.

3. Die einfachere Moeglichkeit waere die, dass in einem Chain-File ueber das Kommando /./4xxxx einen Label innerhalb des Chain-Files angesprungen wird. Dieser Label muss innerhalb des gleichen Chain-Files liegen, er kann nicht in einem anderen File sein.

4. Das DOS-Kommando CHNON (siehe "CHNON" auf Seite 11) darf Teil des Chain-Files sein. Es wird aber erst dann ausgefuehrt, wenn DOS dieses Kommando als naechstes anfordert. Wird CHNON,N angegeben, wird das CHAINING unterbrochen, der File jedoch nicht geschlossen und die momentane Position wird gespeichert fuer ein eventuell folgendes CHNON,Y oder ,D. Danach wird die naechste Eingabe von der Tastatur erwartet. Wird dann CHNON,Y eingegeben und der DOS-CALL Status ist aktiv, wird das CHAINING fortgesetzt, aber der momentane DOS-CALL Status wird verlassen.

5. Anstelle von CHNON kann auch die Abkuerzung /./5 benutzt werden. Diese Abkuerzung wird auch akzepiert, wenn das DOS kein Kommando, sondern eine andere Eingabe erwartet.

Wird waehrend des Chaining-Modus ein Fehler gemeldet, wird das Chaining abgebrochen und die Meldung 'CHAINING ABORT' ausgegeben.

Ein im Chaining-Modus ausfuehrbarer Befehl ist auch der Aufruf eines neuen Chain-Files. Der alte File wird geschlossen und die Kontrolle geht an den neuen File ueber.

Folgende sechs Zeichen stehen im Chaining-File als Steuerzeichen zur Verfuegung :

/./0 In der Version 2 des NEWDOS/80 koennen diese Zeichen auch durch 128 dezimal oder 80H ersetzt werden.

Dieses Zeichen muss an erster Stelle vor einem Abschnittsnamen stehen, ausser, wenn die erste Zeile keinen Abschnittsnamen enthaelt. Der Rest der Zeile enthaelt dann den Abschnittsnamen, der mit dem im Kommando eingegebenen Namen verglichen wird. Alle folgenden Zeilen bis entweder EOF oder die naechste Abschnittskennung gehoeren zu diesem Abschnitt.

/./1 In der Version 2 koennen diese Zeichen auch durch 129 dezimal oder 81H ersetzt werden. Dieses Zeichen am Anfang einer Zeile bewirkt, dass der Rest dieser Zeile auf dem Bildschirm angezeigt wird und das System auf das Druecken der ENTER- Taste wartet. Es fuehrt also eine Pause-Funktion aus.

/./2 In der Version 2 koennen diese Zeichen auch durch 130 dezimal oder 82H ersetzt werden. Dieses Zeichen am Anfang einer Zeile bewirkt, dass der Rest dieser Zeile ueberlesen wird. Es koennen also auf diese Art Kommentare in den File eingefuegt werden, die nicht auf dem Bildschirm angezeigt werden.

/./3 In der Version 2 koennen diese Zeichen auch durch 131 dezimal oder 83H ersetzt werden. Dieses Zeichen am Anfang einer Zeile bewirkt, dass der Rest der Zeile auf dem Bildschirm erscheint. Auf diese Art kann dem Anwender der momentane Status angezeigt werden. Es wird keine Eingabe erwartet, das System faehrt mit der naechsten Zeile fort.

/./4 Diese Zeichen koennen auch durch 132 dezimal oder 84H ersetzt werden. Der Rest des Records besteht aus einem bis zu 32 Stellen langen Namen, an dem das Chaining innerhalb des Chain-Files fortgesetzt werden soll.

/./5 Diese Zeichen koennen auch durch 133 dezimal oder 85H ersetzt werden. Der Rest des Records enthaelt nur entweder die Buchstaben Y, N oder D. Die Zeichenkette steht im Prinzip nur fuer das Kommando CHNON, die Buchstaben sind der Parameter dieses Kommandos.

1. /./5N bedeutet, dass das momentane Chaining abgebrochen wird, der Chain-File wird jedoch nicht geschlossen.

2. /./5Y bedeutet, dass das momentane Chaining aktiv bleibt, der momentane DOS-Level wird jedoch verlassen.

Die Moeglichkeit einer abschnittsweisen Ausfuehrung eines Chain- Files wurde deshalb eingebaut, damit der Benutzer einzelne Abschnitte mit verschiedenen Kommandofolgen ausfuehren kann. Eine Kommandofolge besteht normalerweise aus relativ wenigen Bytes und es waere Platzverschwendung, deshalb einen eigenen File zu machen, der einen ganzen Granule belegt. Durch diese abschnittsweise Ausfuehrung kann in einem Granule ein File mit ca. 12 Abschnitten zu je 100 Byte Laenge angelegt werden.

Auf der NEWDOS/80 Diskette wird ein BASIC-Programm mit dem Namen CHAINTST/BAS mitgeliefert. Der Benutzer kann darin sehen, wie ein Chainfile aufgebaut und aufgerufen wird. Beim Aufruf des Chainfiles wird, wenn kein Typ im Namen angegeben ist, der Defaultwert /JCL angenommen.

Das Chain-Kommando kann entweder als DOS-CALL oder ueber die BASIC-Funktion CMD"..." aufgerufen werden.

CHAIN kann nicht im MINI-DOS benutzt werden! Ein DOS-CALL darf nicht im Chaining-Modus erscheinen!

Der Benutzer ist selbst dafuer verantwortlich, dass im Chaining- Modus keine undurchfuehrbaren Befehle oder Situationen auftreten koennen.

Waehrend ein Chain-File abgearbeitet wird, kann der Anwender die Ausfuehrung durch Druecken der Taste Up Arrow abbrechen oder durch Druecken der Taste Right Arrow eine Pause herbeifuehren. Die Pause wird wieder durch Druecken der ENTER-Taste abgebrochen und der Chain-File weiter bearbeitet.

4.4 DOS-CALL

In aelteren NEWDOS-Versionen war das BASIC das einzige Assembler-Programm, das das DOS aufrufen konnte, um einen DOS-Befehl auszufuehren (CMD"..."). Im NEWDOS/80 ist es jetzt jedem Assembler-Programm moeglich, einen DOS-CALL auszufuehren.

Das Anwenderprogramm hat den DOS-Befehl in einem Puffer, der mit Return (0DH) enden muss und ruft das Unterprogramm 4419H (siehe "4419H - DOS-CALL" auf Seite 56) mit der Adresse des Puffers in HL auf. Dieses Kommando wird von Kleinbuchstaben (falls vorhanden) umgesetzt in Grossbuchstaben und in den DOS-Puffer gestellt zur Ausfuehrung. Dieser DOS-Befehl kann auch der Aufruf eines anderen Programms sein.

Nach der Rueckkehr aus diesem DOS-CALL muss das Anwenderprogramm selbst folgende drei Stati abfragen :

- Bei gesetztem Carry-Bit ist ein Fehler aufgetreten und er wurde bereits angezeigt. Der Anwender muss in seinem Programm selbst entscheiden, ob es fortgefuehrt werden oder ein Abbruch ueber die Subroutine 4030H erfolgen soll.

- Bei den Konditionen 'NC' und 'NZ' ist ein Fehler aufgetreten, der allerdings noch nicht auf dem Bildschirm angezeigt wurde. Im A-Register steht in den rechten 6 Bits die Fehlernummer. soll der Fehler angezeigt werden, kann dies ueber die Routine 4409H geschehen, das Bit 7 im A-Register muss gesetzt sein. Ein Abbruch mit Anzeige des Fehlers geschieht ebenfalls ueber 4409H, jedoch Bit 7 im A-Register muss 0 sein. Wurde das Anwenderprogramm von DOS her aufgerufen, erscheint im Falle eines Abbruchs die Meldung 'DOS READY', andernfalls wird das aufrufende Programm

wieder fortgesetzt mit der Fehlernummer im A-Register und 'NC' und 'NZ'.

- Bei den Konditionen 'NC' und 'Z' ist kein Fehler aufgetreten, die aufgerufene Funktion wurde normal beendet. Da nur AF vom DOS veraendert wurde, ist die einzige Moeglichkeit, einen Parameter zu uebergeben, die Benutzung der zwei Bytes auf der Adresse 4403H und 4404H (17411 und 17412). Eine andere Moeglichkeit waere ebenfalls, die hoeheren, nicht benutzten Bytes im DOS-Inputpuffer zu benutzen. Es muss aber darauf geachtet werden, dass das DOS alle Kommandos vor der Ausfuehrung in diesen Buffer schreibt.

4.5 JKL

Das NEWDOS/80 bietet durch eine residente Routine die Moeglichkeit, den gesamten Bildschirminhalt auf den Drucker auszugeben. Damit kann der Anwender den momentanen Bildschirminhalt auf dem Drucker sichern, bevor er ueberschrieben wird.

Folgende Bedingungen muessen zur Ausfuehrung der Funktion 'JKL' erfuellt sein :

- Systemoption AD=Y
- Interrupts sind aktiv oder das momentane Programm wartet gerade auf eine Eingabe von der Tastatur und die Systemoption AJ=Y.

Unter diesen Bedingungen kann durch gleichzeitiges Druecken der Tasten J, K und L das laufende Programm unterbrochen werden und der Bildschirminhalt ausgedruckt werden. Alle Zeichen auf dem Bildschirm werden ohne Modifikation ausgedruckt, man kann sie jedoch ueber die Systemoption AK beeinflussen.

ACHTUNG !!! Wenn der Drucker nicht Ready ist oder waehrend des Druckes ausfaellt, bleibt die Routine haengen! Es wird keine Meldung auf dem Bildschirm ausgegeben!

Bei der Systemoption AK=N wird jedes Zeichen auf dem Bildschirm, dessen Code groesser ist als 127 (7FH), in einen Punkt umgewandelt, da die meisten Drucker diese Zeichen nicht darstellen koennen. Bei Druckern mit dieser Moeglichkeit (z. B. EPSON) koennen die graphischen Zeichen bei der Option AK=Y ausgegeben werden.

Sobald der Ausdruck fertig ist, wird das unterbrochene Programm an der gleichen Stelle fortgefuehrt (siehe auch "MINI-DOS" auf Seite 69).

4.6 ASYNCHRONE AUSFUEHRUNG

In NEWDOS/80 wird, ebenso wie in TRSDOS, eine begrenzte Art der asynchronen Ausfuehrung eines Programms zugelassen. Dies wird dadurch ermoeglicht, dass ein Programm in die Interrupt-Kette eingefuegt wird. Dies muss jedoch ueber DOS-Routine 4410H (siehe "4410H - ACTIVATE USER INTERRUPT ROUTINE" auf Seite 55) geschehen, das Herausnehmen aus dieser Kette ueber 4413H (sie-

he "4413H - DEACTIVATE USER INTERRUPT ROUTINE" auf Seite 56).
In diesen Kapiteln genaue Format des Aufrufs beschrieben.

ACHTUNG !!! Es wird nochmals darauf hingewiesen, dass diese Routinen unterschiedlich sind zum TRSDOS!

5.0 DOS-MODULE, DATENSTRUKTUREN UND ZUSAETZLICHE INFORMATIONEN

Dieses Kapitel gibt Informationen ueber die verschiedenen NEWDOS/80 Module, ueber das Directory und die File-Control-Blocks.

5.1 BENOETIGTE FILES

5.1.1 DIR/SYS

Directory (Inhaltsverzeichnis)

Die Groesse kann frei festgelegt werden zwischen 2 und 6 Granules. Das Directory muss auf jeder Diskette sein fuer das NEWDOS/80. Es enthaelt alle Informationen ueber die Files auf einer Diskette. Wird von 'FORMAT' automatisch angelegt.

5.1.2 BOOT/SYS

Bootstrap-Loader

Die Groesse des Bootstrap-Loaders ist immer 1 Granule. Er belegt immer das erste Granule auf einer Diskette. Dieses Granule enthaelt im NEWDOS/80 ausser dem Bootstrap-Loader (1. Sektor) noch zusaetzlich im 3. Sektor die allgemeine Systeminformation (Kommando SYSTEM). Bei Power-On und beim Reset wird der Bootstrap-Loader geladen um das System hochzufahren. Bei der nachfolgenden Systeminitialisierung wird der 3. Sektor mit den Systemdaten eingelesen.

5.2 NEWDOS/80-MODULS

Das DOS-Betriebssystem besteht aus insgesamt 10 verschiedenen Programmen, die jeweils bei Bedarf in drei verschiedene Speicherbereiche geladen werden. SYS0/SYS liegt als einziges Modul im Bereich von 4000H bis 4CFFH, die Moduls SYS1/SYS bis SYS9/SYS (ausgenommen SYS6/SYS) werden bei Bedarf in den Bereich von 4D00H bis 51FFH geladen, SYS6/SYS liegt zwischen 5200H und 6FFFH.

Name	Groesse	Funktion
SYS0/SYS	3 Granules	Floppy-Ein- Ausgabe, Interruptchain, nachladen anderer Moduls. Einziges residentes Modul.
SYS1/SYS	1 Granule	DOS-Kommandos interpretieren
SYS2/SYS	1 Granule	Files generieren, eroeffnen, Platz belegen, Directory-Eintrag erzeugen, Passwoerter verarbeiten, Anwenderprogramme laden und starten, Kommandos RENAME und LOAD bearbeiten.
SYS3/SYS	1 Granule	Files schliessen, loeschen, Eintraege und Loeschungen in der 25ms-Interrupt-Chain. Kommandos BLINK, BREAK, CLOCK, DEBUG, JKL, LCDVR, LC, VERIFY und den groessten Teil von PURGE.
SYS4/SYS	1 Granule	DOS-Fehlermeldungen
SYS5/SYS	1 Granule	DEBUG
SYS6/SYS	6 Granules	Kommandos FORMAT, COPY, APEND, PDRIVE und SYSTEM
SYS7/SYS	1 Granule	Kommandos TIME, DATE, AUTO, ATTRIB, PROT, DUMP, HIMEM sowie der erste Teil von PURGE, SYSTEM und PDRIVE.
SYS8/SYS	1 Granule	Kommandos DIR, FREE
SYS9/SYS	1 Granule	Kommandos BASIC2, BOOT, CHAIN CHNON, MDCOPY, PAUSE und STMT. Stellt Benutzer-Routinen in die Warte-schlange (DOS-Routinen 4461H und 4464H im Kapitel 3).
SYS14/SYS	1 Granule	Kommandos CLEAR, CREATE, ERROR, LIST, PRINT und ROUTE.
SYS15/SYS	1 Granule	Kommandos FORMS und SETCOM.
SYS16/SYS	1 Granule	Das meiste des Kommandos PDRIVE.
SYS17/SYS	1 Granule	Kommando WRDIRP und das meiste des Kommandos SYSTEM.

5.3 NEWDOS/80 BASIC-MODULS

Das NEWDOS/80-BASIC ergaenzt das TRS80-ROM-BASIC und besteht aus einem residenten Hauptmodul und 4 zusaetzlichen Moduls. Diese Moduls werden nur bei Bedarf geladen und liegen im Bereich von 4000H bis 51FFH.

BASIC/CMD

Laenge 4 Granules. Das residente Hauptprogramm wird in den Bereich ab 5700H geladen. Es muss nicht auf der Systemdiskette sein, sondern kann auch, wie andere Programme, auf einer Datendiskette liegen. Es wird nur zum Aufruf von BASIC benoetigt. Die Floppy mit dem BASIC kann dann herausgenommen werden.

SYS13/SYS

Laenge 1 Granule mit folgender Funktion: Anzeige der BASIC-Fehlermeldungen und der erste Teil von RENUM. Muss beim Aufruf auf der Systemdiskette sein.

SYS12/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung des BASIC-Befehls 'REF'. Muss beim Aufruf auf der Systemdiskette sein.

SYS11/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung des BASIC-Befehls 'RENUM'. Muss beim Aufruf auf der Systemdiskette sein.

SYS10/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung der BASIC-Befehle 'GET' und 'PUT'. Muss beim Aufruf auf der Systemdiskette sein.

SYS18/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung der direkten Kommandos. Muss beim Aufruf auf der Systemdiskette sein.

SYS19/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung der Kommandos LOAD, RUN, MERGE, SAVE und CMD"F"DELETE. Muss beim Aufruf auf der Systemdiskette sein.

SYS20/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung verschiedener BASIC-Statements. Ist Normalerweise bei dem Ablauf eines

BASIC-Programms im Speicher. Muss beim Aufruf auf der Systemdiskette sein.

SYS21/SYS

Laenge 1 Granule mit folgender Funktion: Abarbeitung des Kommandos CMD"O Muss beim Aufruf auf der Systemdiskette sein.

5.4 ANDERE MODULS

DIRCHECK/CMD

Laenge 3 Granules. Ueberprueft das Directory einer bestimmten Diskette und listet es auf dem Bildschirm oder Drucker. Siehe "DIRCHECK" auf Seite 101.

EDTASM/CMD

Laenge 5 Granules. Editor und Assembler fuer Z80 Maschinenprogramme mit Ausgabe auf Floppy oder Kassette. Siehe "EDTASM" auf Seite 103.

SUPERZAP/CMD

Programm fuer direkten file- oder sektorweisen Zugriff auf die Floppy zum Ueberpruefen oder Modifizieren der Floppy. Siehe "SUPERZAP" auf Seite 90.

CHAINTST/BAS

BASIC-Programm zur Demonstration der Erzeugung und der direkten Ausfuehrung von Chain-Files.

CHAINBLD/BAS

BASIC-Programm zum Erzeugen und Editieren einfacher CHAIN-Files zur Ausfuehrung mit CHAIN oder DO.

DISASSEM/CMD

Disassembler fuer Z80 Maschinenprogramme. Siehe "DISASSEM" auf Seite 94.

LMOFFSET/CMD

Erzeugung einer anderen Ladeadresse eines Programmes. Siehe "LMOFFSET" auf Seite 98.

ASPOOL/MAS

Automatisches Spooler Programm, modifiziert von APPARAT fuer NEWDOS/80. Siehe "Spooler-Programm" auf Seite 106.

5.5 MINIMAL-SYSTEME

Ein Minimalsystem kann erzeugt werden, indem man ueber 'COPY' eine Diskette mit dem ganzen NEWDOS/80 auf eine leere Diskette kopiert und die nicht benoetigten Files loescht. Ein Minimalsystem mit Filehandling mit OPEN und CLOSE belegt zehn Granules und besteht aus BOOT/SYS, DIR/SYS, SYS0/SYS-SYS4/SYS. Wenn DEBUG benoetigt wird, muss SYS5/SYS auf der Diskette bleiben. Dem Kapitel "NEWDOS/80-Moduls" auf Seite 77 sind weitere Moduls und ihre Funktion zu entnehmen. Die fuer BASIC benoetigten Files sind in "NEWDOS/80 BASIC-Moduls" auf Seite 79 aufgefuehrt.

Wenn ein benoetigtes Systemprogramm beim Nachladen nicht gefunden wird, gibt es folgende Moeglichkeiten:

- Wenn SYS4/SYS auf der Systemfloppy ist, wird die Meldung 'SYSTEM PROGRAM NOT FOUND' angezeigt und ueber 4409H zurueckgesprungen.
- Wenn SYS4/SYS auch nicht vorhanden ist, erfolgt ein Reset.

Dies gilt fuer alle Systemprogramme von SYS0/SYS bis SYS21/SYS. Tritt ein solcher Fehler waehrend eines BASIC-Programmes auf, kann man ueber BASIC * in das BASIC-Programm zurueckkehren.

ACHTUNG! Wenn ein Systemfile auf der Floppy geloescht wird, kann er nicht einfach zurueckkopiert werden von einer anderen Floppy, da fuer die Systemfiles im Directory an ganz bestimmter Stelle der Eintrag erfolgen muss. Ferner muss z. B. SYS0/SYS immer die gleichen Granules belegen. Vor dem Konieren eines Systemfiles muss also erst der Directory-Eintrag restauriert werden, dann kann ein COPY erfolgen (ZAP benutzen !).

5.6 STRUKTUR DES DIRECTORY

Beim Modell I koennen NEWDOS/80- und TRSDOS-Disketten untereinander vertauscht werden, wenn die NEWDOS/80-Diskette nach folgenden Standardkriterien aufgebaut wurde :

- Directorygroesse 2 Granules (siehe Parameter DDGA von "FORMAT" auf Seite 26, und "COPY" auf Seite 12)
- 10 Sektoren je Spur, 2 Granules je "Lump" und 5 Sektoren je Granule (dies sind die Standardangaben fuer das NEWDOS/80).

Files auf der Diskette koennen zwar kopiert werden, sind jedoch teilweise im anderen Betriebssystem nicht funktionsfaehig. Dies trifft auf jeden Fall zu auf System-Files, BASIC, SCRIPSIT ELECTRIC PENCIL etc.

Beim Modell III sind die Directories im NEWDOS/80 und TRSDOS unterschiedlich und nicht austauschbar. TRSDOS Modell III Disketten koennen nicht direkt mit NEWDOS/80 ausgetauscht werden, NEWDOS/80 Disketten fuer Modell III koennen nicht mit TRSDOS ausgetauscht werden. Wenn eine NEWDOS/80 Diskette mit einfacher Schreibdichte das Directory-Standardformat von TRSDOS hat, gibt es im TRSDOS ein Konvertierungsprogramm fuer Datenfiles. In der COPY-Funktion gibt es ebenfalls einige Parameter, mit denen man Files kopieren kann auf Disketten mit anderem Format.

Das NEWDOS/80 initialisiert alle Directoryeintraege einer Floppy und macht sie dem Benutzer zugaenglich (ausser BOOT/SYS und DIR/SYS). Aus diesem Grunde sind in einem NEWDOS/80 Betriebssystem 62 Eintraege moeglich, im Gegensatz zu nur 48 Eintraegen beim TRSDOS. Ferner kann das Directory bis auf 6 Granules vergroessert werden bei der Formatierung (siehe FORMAT und COPY, DDGA-Parameter). Dann sind bis zu 222 Eintraege moeglich.

5.6.1 GAT-Sektor

GAT = Granule Allocation Table

Der GAT-Sektor ist immer der 1. Sektor im Directory und enthaelt folgende Informationen :

Byte 00H-5FH

Belegtkennzeichen der Granules, jedes Byte ist einem "Lump" zugeordnet, jedes Granule ist einem Bit zugeordnet. Die Anzahl der Granules pro "Lump" wird bit dem GPL-Parameter bei PDRIVE angegeben und hat einen Wert zwischen 2 und 8. Das erste Bit (von rechts gezaehlt) deutet auf das erste Granule des "Lumps" (Bit 0), Bit 1 deutet auf den 2. "Lump" usw. bis Bit 7. Wenn das entsprechende Bit 0 ist, ist dieser "Lump" frei. Ist dieses Bit auf 1 gesetzt, so ist der "Lump" entweder belegt oder existiert nicht.

Byte 60H-0BFH

Tabelle der gesperrten oder nicht vorhandenen "Lumps". Die Bytes 60H bis 0BFH korrespondieren ebenso wie die Belegt-Tabelle mit je einer Spur, 60H entspricht Spur 0, 0BFH entspricht Spur 95. Ein 0FFH im entsprechenden Byte bedeutet, dass diese Spur komplett gesperrt ist. Ist ein Bit auf 0 gesetzt bedeutet dies, dass dieser Granule zu benutzen ist. Im Moment ist es so, dass das NEWDOS/80 diesen Teil nur aus Gruenden der Kompatibilitaet zum TRSDOS anlegt (hier sind diese Bytes die sog. Lock-Out-Bytes), sie aber nicht benutzt. Diese Bits werden im NEWDOS/80 nie gesetzt. Im Bedarfsfall werden diese Bytes als zusatzliche GAT-Bytes benutzt.

Um den Platz der Diskette besser auszunutzen, wird im NEWDOS/80 erlaubt, die Bytes 60H - 0BFH als GAT-Bytes sofort anzulegen, falls dies notwendig ist. In diesem Falle gehen die GAT-Bytes also nicht von 00H - 5FH, sondern von 00H - 0BFH. Diese Erweiterung wird automatisch

ausgefuehrt, wenn die Anzahl der "Lumps" 96 uebersteigt.

Byte 0CEH und 0CFH	Komprimiertes Passwort dieser Diskette.
Byte 0D0H bis 0D7H	Diskettenname
Byte 0D8H bis 0DFH	Datum
Byte 0E0H bis 0FFH	Nach Restart ausgefuehrter Befehl, der durch 'AUTO' gesetzt wurde oder nur Return (0DH).

5.6.2 HIT-Sektor

HIT - Hash Index Table

Der HIT-Sektor ist der 2. Sektor im Directory und dient als sogenannter Prefind-Sektor. Damit lassen sich sehr schnell freie Eintraege im Directory finden (das entsprechende Byte ist 0) oder bestimmte Eintraege. Da pro Directory-Eintrag nur ein Byte zur Verfuegung steht, muss fuer den Namen eines Programmes erst ein sogenannter Kurz-Name erzeugt werden. Wird nun ein bestimmter Eintrag gesucht, so muss zu dem gesuchten Namen erst der sogenannte HASH-Code errechnet werden, mit diesem HASH-Code wird dann der HIT-Sektor durchsucht. Der Name BOOT/SYS hat z. B. den HASH-Code 0A2H, DIR/SYS z. B. 2CH. Wird dieser HASH-Code in dem HIT-Sektor gefunden, so wird in dem entsprechenden Directory-Eintrag der Name ueberprueft. Stimmt dieser Name nicht mit dem gesuchten Namen ueberein, wird weiter nach einem uebereinstimmenden HASH-Code gesucht.

Aus der Position des gefundenen HASH-Codes im HIT-Sektor laesst sich die genaue Adresse des Directory-Eintrags errechnen. So ist Byte 0 im HIT-Sektor der 1. Eintrag im 1. FDE-Sektor, Byte 1 der erste Eintrag im 2. FDE-Sektor etc. Byte 20H ist der 2. Eintrag im ersten FDE-Sektor, Byte 21H der 2. Eintrag im 2. Sektor usw. Ist dieses Byte 0, so ist der entsprechende Eintrag frei und kann belegt werden.

Nur das 32. Byte im HIT-Sektor hat bei NEWDOS/80 eine andere Bedeutung als alle anderen Bytes. Hier wird die Anzahl der zusaetzlich zur Verfuegung stehenden Directory-Sektoren angegeben (siehe FORMAT- oder COPY-Parameter DDGA). Der Wert kann 0, 5, 10, 15 oder 20 sein und wird beim formatieren gesetzt.

5.6.3 FDE-Sektoren

FDE - File Directory Entries. Directoryeintraege der Files.

Der Rest des Directorys ist belegt mit den FDE-Sektoren. Jeder Sektor enthaelt maximal 8 Eintraege zu je 32 Byte. Ein FDE ist frei, wenn das 4. Bit seines ersten Bytes 0 ist und belegt bei Bit 4 = 1. Bei einem belegten FDE ist das Bit 7 geloescht (FPDE-File Primary Directory Entry). Ist dieses Bit gesetzt, handelt es sich um eine Erweiterung des FPDE (FXDE-File Extension Directory Entry). Wird ein Eintrag z. B. durch 'KILL' geloescht, so wird nur das 4. Bit des 1. Bytes auf 0 gesetzt. Falls ein Benutzer ein "saubers" Directory ohne geloeschte Eintraege haben moechte, kann er alle 32 Bytes eines

bereits geloeschten Eintrags mit der Funktion C des Programms DIRCHECK auf 0 setzen.

5.7 FPDE

File Directory Primary Entry. Ersteintrag des Files.

Jeder File erhaelt, sobald er eroeffnet wird, einen sog. Ersteintrag in einem der FDE-Sektoren. Dieser Eintrag enthaelt folgendes :

1. Byte:

- Bit 7=0 entspricht einem FPDE (Ersteitrag), Bit 7=1 entspricht einem FXDE (Folgeeintrag).
- Bit 6=1 wenn der File ein Systemfile ist
- Bit 5=0 keine Bedeutung
- Bit 4=1 wenn ein Eintrag belegt ist
- Bit 3=1 wenn der File unsichtbar ist (invisible)
- Bits 2-0 = Zugriffslevel (siehe "ATTRIB" auf Seite 7)

2. Byte:

- Bit 7=0 sagt aus, dass dem File bei Bedarf mehr Platz zugeordnet werden darf.
- Bit 7=1 bedeutet, dass eine weiter Platzbelegung untersagr ist. Die Kommandos DIR, ATTRIB, CREATE, und die Routine fuer die Platzbelegung im DOS benutzen dieses Bit.
- Bit 6=0 bedeutet, dass beim Close der ueberfluessige Platz freigegeben werden darf.
- Bit 6=1 sagt aus, dass der nicht benutzte Platz beim Close nicht freigegeben werden darf. Die Kommandos DIR, ATTRIB, CREATE und die Close-Routine im DOS benutzen dieses Bit.
- Bit 5=1 sagt aus, dass mindestens ein Sektor dieses Files geschrieben wurde, seit dieses Bit das letzte Mal auf 0 gesetzt wurde. Die Kommandos DIR, ATTRIB, CREATE, PROT, COPY und die Schreibroutine im DOS benutzen dieses Bit.
- Bits 4-0 = z. Zt. nicht benutzt.

4. Byte : 0-255 die Position des End-Of-File Bytes im letzten Sektor des Files (0 entspricht 256).

5. Byte : 0-255 Recordlaenge des Files, 0 entspricht 256. Wenn ein File ueber die DOS-Routine 4420H erzeugt wird, wird der Wert des Registers B in diesem Byte abgespeichert. Wenn ein existierender File, auch wenn er als neuer Output-File deklariert ist, eroeffnet wird, so wird dieser Wert nicht veraendert. Dieser Wert wird im NEWDOS nicht benutzt (???).

6. bis 13. Byte : Filename, linksbuendig, aufgefuellt mit Leerzeichen falls notwendig.

14. bis 16. Byte : Filetyp, linksbuendig, aufgefuellt mit Leerzeichen falls notwendig.

17./18. Byte : Komprimiertes Update-Passwort.

19./20. Byte : Komprimiertes Zugriffs-Passwort.

21./22. Byte : Relative Sektornummer des letzten Sektors in diesem File. Wenn das 4. Byte ungleich 0 ist, dann ist dieser Wert die relative Sektornummer + 1. In NEWDOS/80 musste so verfahren werden, um die Files kompatibel zum TRSDOS zu halten. Nur dann, wenn ein File eroeffnet ist, ist auch dieser Wert unterschiedlich zum TRSDOS. Dieser Wert wird naemlich nur bei der Generierung eines neuen Files und beim CLOSE veraendert, nicht jedoch waehrend des Schreibens. Dies hat zur Folge, dass, wenn das Schreiben in einen File durch druecken der Reset-Taste unterbrochen wird, zwar der neue Output in diesem File steht, diese zwei Bytes jedoch noch den alten EOF- Sektor enthalten.

23. bis 30. Byte : 4 Byte-Paare (Erweiterungs-Elemente), von denen jedes einen Bereich auf der Diskette definiert, der diesem File zugeordnet ist.

- 1. Byte :
  - 255 (0FFH) = Ende der Erweiterungen
  - 254 (0FEH) = das 2. Byte enthaelt den DEC fuer den naechsten FXDE fuer diesen File.
  - 0-253 (0-0FDH) = relative "Lump"-Nummer auf der Diskette, wo der Bereich dieses Files liegt. Die Spurnummer ist aber begrenzt auf 0-191 bei der Version 2.

- 2. Byte (wenn das 1. Byte < 254) : Die linken 3 Bits dieses Bytes enthalten die Anzahl der Granules, die zwischen dem Start des "Lumps" und dem Start des File-Bereichs liegen. Bei einer kleinen Standartfloppy also immer 000 oder 001. Beim NEWDOS/80 liegt dieser Wert bei max. 7.

  Die rechten 5 Bits dieses Bytes enthalten die Anzahl der fortlaufenden Granules-1 fuer diesen File.

31./32. Byte : Enthalten ein Erweiterungselement fuer diesen File, wenn das erste dieser 2 Bytes entweder 255 oder 254 ist.

5.8 FXDE

File eXtended Directory Entry - Erweiterungseintrag

Wenn ein File mehr als 4 verschiedene Bereiche auf einer Floppy belegt, muessen die Adressen der Bereiche, die groesser als 4 sind, in einem FXDE untergebracht werden, da in dem FPDE nur 4 Bereiche untergebracht werden koennen. Der FXDE hat folgendes Format:

1. Byte Bit 7 und 4 sind 1 zur Erkennung des FXDE's.

2. Byte DEC (Directory Entry Code) des vorangegangenen FPDE's oder FXDE's. Dies ist eine sog.

Rueckwaertsverkettung. Im 31. Byte des vorangegangenen Eintrags steht 254, und im 32. Byte steht der DEC des momentanen FXDE's.

3.-22. Byte Werden nicht benutzt und sollten 0 sein.

23. bis 32. Byte Sind definiert wie die Bytes im FPDE.

5.9 FCB

File Control Block (wird auch als IOB-Input/Output Block bezeichnet)

Sobald auf einen File zugegriffen wird, entweder lesend oder schreibend, muss eine Verbindung zwischen dem Programm und dem hergestellt werden. Diese Verbindung wird im DOS durch das OPEN geschaffen. Solange es diese Verbindung gibt (bis zum CLOSE) stehen in einem 32 Bytes langen Block alle benoetigten Kontrollinformationen. Darin enthalten sind unter anderem Filenamen (nur beim OPEN), Adresse des Benutzerbuffers, Position usw. Dieser Kontrollblock darf, solange ein File eroeffnet ist, nicht fuer andere Zwecke benutzt werden, da dieser Buffer vom DOS benoetigt wird und nur einmal existiert.

Zum Eroeffnen eines Files (ueber die Routinen 4420H oder 4424H) muss der Benutzer im Registerpaar DE die Adresse eines 32 Bytes langen Blocks uebergeben, den dann das System als FCB belegt. Beim Eroeffnen muss der Filename in den ersten Bytes dieses Blocks liegen, die CLOSE-Routine schreibt diesen Filenamen wieder in diesen Block zurueck. Solange dieser File eroeffnet ist, haben die 32 Bytes folgende Bedeutung :

1. Byte Die einzelnen Bits haben folgende Bedeutung:

Bit 7 =1-File ist eroeffnet, =0-File ist geschlossen

Bit 6-3 =0-undefiniert

Bit 2 in Version 2 nicht benutzt. .go kap5a9a =-der Wert im 11. Byte dieses FCB's ist ein Sektor innerhalb einer Spur, der Wert des 12. Bytes ist eine Spur auf der Diskette. DOS nimmt keine Umwandlung von Spur und Sektor vor, da der Anwender dies bereits getan hat.

Bit 1 =1-der Wert im 3. NEXT-Feld dieses FCB's ist ein RBA innerhalb der Diskette und nicht in einem File. Die Umsetzung auf Spur- und Sektorwerte erfolgt durch entsprechende Division und benutzt nicht die File-Erweiterungen.

Bit 0 =1-die Sektoren in diesem File werden lese-geschuetzt. In diesem Mode schreibt das DOS in das Directory. Dieses Bit sollte nie 1 sein bei Ein- oder Ausgabe ueber die Routinen 0013H oder 001BH.

2. Byte Die einzelnen Bits haben folgende Bedeutung:

Bit 7

- = 1-die Operationen ueber diesen FCB werden Byteweise ausgefuehrt oder als Records. Das NEXT-Feld enthaelt den Zaehler fuer das naechste Byte, das zu schreiben oder zu lesen ist.

- = 0-die Operationen ueber diesen FCB werden sektorweise als 256-Byte-Bloecke abgewickelt, der NEXT-Wert wird immer um 256 erhoeht. Bit 7 wird beim OPEN dann auf 1 gesetzt, wenn das Register b ungleich 0 ist. Ebenso wird es auf 1 gesetzt, wenn eine der ROM-Routinen angesprochen werden, die byteweise Ein-Ausgabe vornehmen (DE muss die FCB-Adresse enthalten).

Bit 6

- = 0-EOF soll bei jedem Schreiben auf die Diskette auf den NEXT-Wert gesetzt werden.

- = 1-der EOF-Wert wird beim Schreiben nur dann veraendert, wenn der NEXT-Wert den EOF-Wert uebersteigt.

Bit 5

- = 0-der Buffer enthaelt den aktuellen Sektor

- = 1-der Buffer enthaelt nicht den aktuellen Sektor

Bit 4 Bufferstatus:

- = 0-der aktuelle Buffer enhaelt keine geaenderten Daten

- = 1-der aktuelle Buffer enthaelt geaenderte Daten, die noch nicht auf die Floppy geschrieben wurden. Wenn bei einem CLOSE dieses Bit auf 1 steht, wird der Buffer automatisch auf die Floppy geschrieben. Wenn dieses Bit gesetzt ist werden die veraenderten Daten auch auf die Floppy geschrieben, wenn ueber eine der Routinen 443FH, 4451H, 4442H, 4445H,4448H oder 444EH die Position auf einen anderen Sektor geaendert wird.

Bit 3 Dieses Bit ist in Version 2 des NEWDOS/80 die Kennung, ob ein File bzw. File-Control-Block im neuen oder im alten Format geschrieben ist. Bei Bit 3 = 0 handelt es sich um einen File im alten Format.

Bit 2 = 0 : Zugriffslevel (siehe "ATTRIB" auf Seite 7)

3. Byte Die Bits 7 - 5 haben die gleiche Bedeutung wie im 2. Byte des FPDE's. Die Bits 4 - 0 sind frei fuer spaetere Verwendung.

4. + 5. Byte Diese Bytes enthalten die Bufferadresse, die beim OPEN in Registerpaar HL stehen. Sektoren werden in diesen Buffer gelesen und daraus auf die Diskette geschrieben.

6. Byte Niedrigster Wert des NEXT-Feldes. In diesem Byte steht die relative Position innerhalb eines Sektors (siehe Beschreibung von Byte 11-12)

7. Byte Dieses Byte enthaelt die Laufwerksnummer des Files beim OPEN.

8. Byte Dieses Byte enthaelt den DEC des FPDE's des Files. Dieser DEC ist die Verbindung zwischen dem FCB und dem Directory-Eintrag nach dem OPEN, wenn die Directory-Information nicht mehr im FCB steht.

9. Byte Niedrigstes Byte von EOF. Siehe Beschreibung von Byte 13 und 14.

10. Byte Recordlaenge von diesem File, Wert zwischen 0 und und 256. Dieser Wert wird beim Eroeffnen eines Files im B-Register uebergeben. Ist dieser Wert ungleich 0 beim Eroeffnen, wird das Bit 7 des 1. Bytes auf 1 gesetzt. Beim Lesen oder Schreiben eines Sektors in DOS muss HL die Bufferadresse enthalten, wo der Record hingeschrieben werden soll (lesen) oder von wo der Record in den FCB-Buffer gelesen werden soll (schreiben).

11.+12. Byte Das 12., 11. und 5. Byte bilden zusammen die RBA-Adresse innerhalb eines Files. In ihnen steht, welches Byte bei Input oder Output als naechstes zu behandeln ist. Bei byteweisem Lesen oder Schreiben (Bit 7 des 2. Bytes = 1) wird dieser Wert von NEWDOS/80 exakt mitgezaehlt, wo hingegen TRSDOS diesen Wert um 256 hochzaehlt. Bei Sektor-Operationen zaehlen sowohl NEWDOS/80 als auch TRSDOS den NEXT-Wert um 256 hoch, ausgenommen der Fall, dass im NEWDOS/80 nur dann um einen vollen Sektor hochgezaehlt wird, wenn das 6. Byte des FCB's auf 0 steht. TRSDOS und NEWDOS/80 sind in der Behandlung dieses NEXT-Feldes nicht kompatibel.

13.+14. Byte das 14., 13. und 8. Byte bilden zusammen die RBA-Adresse von End-Of-File innerhalb dieses Files (das 1. Byte hinter dem letzten Datenbyte). Dieser Wert wird beim OPEN initialisiert und wird entsprechend dem Bit 6 von Byte 2 dieses FCB's beim sektor-, record- und byteweisen Schreiben veraendert. In NEWDOS/80 wird dieser Wert exakt behandelt, waehrend TRSDOS diesen Wert um 256 hochzaehlt, wenn das 6. Byte des FCB's nicht 0 ist. Aus diesem Grunde sind TRSDOS und NEWDOS/80 in der Behandlung dieses Feldes nicht kompatibel. Um die beschriebenen Disketten austauschbar zu halten, setzt das NEWDOS/80 bei einem CLOSE den Wert EOF im FPDE gleich wie TRSDOS.

15.-22. Byte Diese Bytes sind identisch mit den Bytes 23 bis 30 des FPDE.

23.-24. Byte Fuer den momentanen FXDE, dessen 4 Erweiterungselemente in den FCB-Bytes 25 bis 32 stehen, bedeutet die Zahl in diesem Feld die relative Anzahl von Granules, die dem ersten Erweiterungseintrag zugeordnet sind. Wenn der Wert dieser 2 Bytes 0FFFFH ist, su gibt es keinen FXDE in den Bytes 25 bis 32.

25.-32. Byte Identisch zu den Bytes 23 bis 30 im aktuellen FXDE, falls vorhanden.

Weitere Erlaeuterung der Bytes 17 bis 32:

Die Defintion der Bytes 17 bis 32 hat sich gegenueber der Version 1 von NEWDOS/80 und dem TRSDOS Modell I geaendert. Es wurde angenommen, dass nur wenige Anwenderprogramme diese Bytes benutzen, und dass man aus diesem Grunde diese Bytes dafuer hernehmen konnte, die Anzahl der Zugriffe durch das System auf das Directory zu optimieren. Solche aelteren Programme, wie z.

B. der in BASIC geschriebene SUPERZAP, muessen umgeschrieben werden auf die Verwendung des neuen FCB's. Fehler koennten eine katastophale Auswirkung haben! Aus diesem Grunde wurde das Bit 3 des 2. FCB-Bytes als Kennung fuer den neuen FCB benutzt.

Mit dieser Aenderung im FCB, der die Bytes 17 bis 32 betrifft, ist es jetzt moeglich, bis zu 8 Erweiterungselemente in direktem Zugriff, also ohne weiteren Directory-Zugriff, zu haben. Mit dem Kommando DIR und der Option A wird angezeigt, wie viele Erweiterungselemente ein File hat.

Wenn ein File nun mehr als 8 Erweiterungselemente hat, d. h. wenn mehr als ein FXDE dem File zugeordnet ist, dann enthaelt der FCB die Angaben fuer die ersten 4 Erweiterungselemente und ueber die letzten 4 Erweiterungselemente, die zuletzt angesprochen wurde. Das bedeutet allerdings auch, dass bei grossen Random-Files die Anzahl der Directory-Zugriffe gegenueber der Version 1 bedeutend hoeher sein kann.

6.0 ZUSAETZLICHE PROGRAMME AUF DER NEWDOS/80-DISKETTE

Die folgenden beschriebenen Programme werden auf jeder NEWDOS/80 Diskette mitgeliefert.

6.1 SUPERZAP

Das Programm SUPERZAP/CMD erlaubt es dem Benutzer, jeden Teil des Speichers und jeden Sektor der Floppy zu lesen und zu veraendern, ausgenommen natuerlich das ROM. Alle NEWDOS/80 Benutzer muessen mit diesem Programm umgehen koennen, da unter anderem auch alle noetigen Aenderungen am Betriebssystem oder Programmen in der Art vom Hersteller, der Fa. APPARAT, herausgegeben werden, dass sie ueber die Funktion 'DFS' und 'MOD..' eingegeben werden.

SUPERZAP arbeitet sowohl mit Gross- als auch mit Kleinbuchstaben.

SUPERZAP benutzt die in PDRIVE (siehe "PDRIVE" auf Seite 34) eingegebenen Standardwerte fuer jedes Laufwerk.

6.1.1 Funktionen

Das Startmenu zeigt alle aufrufbaren Funktionen. Sie werden aufgerufen durch Eingabe der entsprechenden Buchstabenkombination und Druecken der ENTER-Taste.

Folgende Funktionen stehen zur Verfuegung:

DD Anzeige eines Sektors der Floppy. SUPERZAP fraegt nach der Laufwerksnummer, der Spur und dem Sektor. Nach der Eingabe wird der entsprechende Sektor gelesen und angezeigt.

DM Anzeige eines Speicherbereichs. SUPERZAP fraegt nach der Startadresse und zeigt dann den Speicherinhalt an. Die Startadresse wird auf volle 256 Bytes abgerundet.

DFS Sektor eines Files anzeigen. Nach der Eingabe des Filenamens fraegt SUPERZAP nach der relativen Sektornummer innerhalb dieses Files, liest diesen Sektor und zeigt ihn an.

DTS Relativen Sektor anzeigen. Die Laufwerksnummer, die Spurnummer und der relative Sektor auf der Spur muessen eingegeben werden. Der Sektor wird gelesen und angezeigt.

DMDB Anzeige eines DUMP-Blocks. Nach der Eingabe des Filenamens eines DUMP-Files (erzeugt ueber das Kommando "DUMP" auf Seite 25) wird die Startadresse des Dumps angezeigt und dann nach der Adresse, die angezeigt werden soll, gefragt. Die Anzeigeadresse wird auf volle 256 Bytes abgerundet und in einem 256 Byte langen Block angezeigt.

VDS Ueberpruefung eines Disksektors. Nach der Eingabe von Laufwerksnummer, Spur und Sektor der zu ueberpruefenden Diskette wird nach der Sektoranzahl gefragt. Ferner muss dann eingegeben werden, ob das

Programm anhalten soll, sobald ein Fehler entdeckt wird. Die Ueberpruefung erfolgt durch einfaches Lesen der entsprechenden Sektoren. Auf diese Art kann sehr schnell ein Sektor mit Lesefehler gefunden werden.

ZDS Loeschen von Sektoren. Nach der Eingabe von Laufwerksnummer, Spur und Startsektor der zu loeschenden Sektoren wird nach der Sektoranzahl gefragt. Die entsprechenden Sektoren werden mit Nullen ueberschrieben. Der Sektorstatus wird nicht veraendert.

CDS Kopieren von Sektoren. Zuerst muessen Laufwerksnummer, Spur und relative Sektornummer auf dieser Spur angegeben werden, von wo die Daten kopiert werden sollen (Source). Dann werden Laufwerksnummer, Spur und relative Sektornummer auf dieser Spur angegeben, wohin die Daten geschrieben werden sollen (Destination). Nach Eingabe der Sektoranzahl werden diese Sektoren kopiert. Ebenso wird der Sektorstatus kopiert.

CDD Kopieren von Daten. Ebenso wie in CDS werden die Source-Angaben eingegeben, danach die Adresse des 1. zu kopierenden Bytes. Nach Eingabe der Destination-Werte muss die Anzahl der zu kopierenden Bytes eingegeben werden (1....65535). Der Sektorstatus wird jedoch nicht veraendert.

DPWE Codiertes Passwort anzeigen. SUPERZAP fragt nach dem Passwort, das codiert werden soll. Das Ergebnis wird in hexadezimal angezeigt. Der gleiche hexadezimale Code steht im FPDE eines Files.

DNTH Codierten Name/Typ anzeigen (HASH-Code). SUPERZAP fragt nach dem Namen eines Files, dann nach dem Typ (/xxx). Das Ergebnis wird in hexadezimal angezeigt. Der gleiche hexadezimale Code steht im HIT-Sektor des Directories.

EXIT Der SUPERZAP wird beendet, die Ausfuehrung kehrt zum DOS zurueck ueber 440DH ('DOS READY').

Bei den Funktionen ZDS, CDS und CDD wird der Anwender nochmals gefragt, ob er sicher ist, dass er die richtige Funktion aufgerufen hat, da diese Funktionen die Daten der Diskette veraendern.

In den Funktionen CDS und CDD erfolgt das Kopieren normalerweise in aufsteigender Reihenfolge. Wenn sich jedoch das letzte Byte, das kopiert werden soll, in dem Bereich 'NACH' befindet, wird von oben her kopiert, da sich die Daten ueberlappen.

Alle Ein/Ausgaben der Floppy werden ueber die normalen DOS-Routinen abgewickelt. Wenn also ein Lese- oder Schreibfehler auftritt, wird der unter der System-Option AK eingegebene Wert fuer die Anzahl der Versuche benutzt.

Wenn bei den Funktionen VDS, ZDS, CDS und CDD ein Lese- oder Schreibfehler auftritt, hat der Anwender die Moeglichkeit, den entsprechenden Sektor nochmals zu lesen, ihn zu ueberspringen oder diese Funktion abzubrechen. In vielen Faellen kann ein Fehler durch nochmaliges Lesen oder Schreiben behoben sein. Bei einem Lesefehler wird der Sektorpuffer auf die entsprechende Stelle geschrieben, egal was in dem Puffer steht.

Solange SUPERZAP auf eine numerische Eingabe wartet, kann durch Eingabe von 'X' diese Funktion abgebrochen werden. Das Startmenu wird dann angezeigt. Soll die Funktion abgebrochen werden, wenn SUPERZAP einen Filenamen erwartet, kann diese Funktion durch druecken der ENTER-Taste ohne sonstige Eingabe abgebrochen werden.

Die Anzeige bei den Funktionen DD, DM, DFS, DDRS und DMDB kann auch durch den Zusatz ',P' auf den Drucker ausgegeben werden.

Bei DD,P , DFS,P und DDRS,P wird nach der Anzahl der auszudruckenden Sektoren gefragt, bei DM,P und DMDB,P muss die Anzahl der auszudruckenden Bytes eingegeben werden.

ACHTUNG !!! Wenn der Drucker nicht READY ist oder waehrend des Ausdrucks einen Fehler meldet, wartet SUPERZAP, bis der Drucker READY ist, ohne dass auf dem Bildschirm etwas ausgegeben wird.

Durch Druecken der 'P'-Taste kann der Ausdruck unterbrochen werden, mit der ENTER-Taste wird er fortgefuehrt. Mit der 'H'-Taste wird der Ausdruck abgebrochen.

6.1.2 Anzeige

Waehrend bei den Funktionen DD, DM, DFS, DDRS und DMDB ein Sektor angezeigt wird, erwartet das NEWDOS eine Eingabe. Diese Eingabe wird nicht mit der ENTER-Taste abgeschlossen und auch nicht am Bildschirm angezeigt. Folgende Moeglichkeiten haben Sie:

X Die momentane Funktion wird abgebrochen und das Startmenu angezeigt.

R Der momentane Sektor oder die Speicherseite werden nochmals angezeigt.

+ oder ; der naechstfolgende Sektor oder die naechsthoehere Speicherseite werden angezeigt.

- Der vorhergehende Sektor oder die naechstniedrige Speicherseite wird angezeigt.

J Gleiche Funktion nochmals starten. Wenn nur der 1. Parameter eingegeben wurde, werden die vorherigen Werte der 2. und 3. Parameter benutzt, wenn nur 2 Parameter eingegeben werden, wird der vorangegangene Wert des 3. Parameters benutzt.

K Gleiche Funktion nochmals starten. Der erste vorher eingegebene Parameter wird wieder benutzt. Wird nur der 2. Parameter eingegeben, wird der 3. Parameter der vorangegangenen Eingabe benutzt.

SCOPY Nur in DD, DFS und DDRS: der momentane Sektor wird auf einen anderen Sektor, dessen Laufwerksnummer, Spur und Sektoradresse eingegeben werden muessen, kopiert. Dieser Sektor kann der Gleiche sein wie der gelesene. SUPERZAP liest den 'NACH'-Sektor und meldet den Status, dann wird der 'VON'-Sektor auf den 'NACH'-Sektor geschrieben. Dies kann vor allem dann nuetzlich sein, wenn ein Sektor ein falsches Paritybit hat und mit wenigen Ausnahmen korrekt gelesen wurde. Wird der Sektor auf sich selbst kopiert wird ein neues Paritybit generiert und der Sektor kann dann wieder hergestellt werden. Ebenso kann auf diese Art der Status eines Sektors geaendert werden.

F Solange der SUPERZAP im Anzeigemodus ist, kann man ueber dieses Kommando bis zu vier Bytes in einer Folge im Memory oder auf der Diskette suchen. Wenn die Suche erfolgreich verlaufen ist, wird der Memory-Block oder der Sektor mit den gesuchten Bytes angezeigt, die gesuchten Bytes werden durch den

blinkenden Cursor markiert. Der Cursor verschwindet wieder, sobald eine Taste gedrueckt wird.

Folgende Kommandos fuer die Suche stehen zur Verfuegung:

F,aa,bb,cc,dd Die 1 bis 4 hexadezimalen Bytes aa bis dd werden gespeichert und die Suche beginnt je nach Anzeigemodus im Memory, auf dem ersten Sektor der Diskette oder dem ersten Sektor eines Files.

F, Gleiche Funktion wie vorher, nur dass die bereits gesuchten Bytes vom letzten Kommando nochmals gesucht werden.

Fxx,aa,bb,cc,dd Die 1 bis 4 hexadezimalen Bytes aa bis dd werden gespeichert und die Suche beginnt je nach Anzeigemodus im Memory auf der angezeigten Seite beim Byte xx oder im momentan angezeigten File beim Byte xx.

Fxx, oder Fxx Gleiche Funktion wie vorher, nur dass die bereits gesuchten Bytes vom letzten Kommando nochmals gesucht werden.

F Die Suche beginnt bei den zuletzt gefundenen Bytes mit den bereits eingegebenen Bytes.

L,aa,bb,cc,dd Das Kommando L wird anstelle des Kommandos F,aa,bb,cc,dd benutzt, wenn es sich um einen File im Load-format wie z. B. SUPERZAP/CMD LMOFFSET/CMD etc. handelt. Dabei werden die Kontrollbytes von der Suche ausgenommen (Blocklaenge, Startadresse etc.). Dadurch werden auch Bytefolgen gefunden, die durch diese Kontrollinformation unterbrochen wurden. Die Kontrollinformation darf nicht veraendert werden. Sie beginnt meist mit dem Kontrollbyte 01H. Ausser dass die Kontrollinformation ignoriert wird, besteht kein Unterschied zwischen dem Suchmodus L und F. Fuer eine weitere Suche in einem Load-Modul kann man auch F, benutzen.

L, Gleiche Funktion wie vorher, nur dass die bereits gesuchten Bytes vom letzten Kommando nochmals gesucht werden.

MOD.. Nur bei DD, DM, DFS und DDRS. SUPERZAP geht in den Aenderungsmodus und zeigt den Cursor an der Stelle an, die mit .. (Wert von 00 bis FFH) eingegeben wurde.

6.1.3 Modifizieren

Wenn sich SUPERZAP im Aenderungsmodus nach dem Aufruf von MOD.. befindet, koennen einzelne Bytes im Sektor oder der Speicherseite geaendert werden. Folgende Vereinbarungen sind zu beachten:

Als hexadezimale Werte duerfen die Zeichen 0-9 und A-F eingegeben werden. Das angezeigte hexadezimale Digit wird durch das eingegebene Zeichen ueberschrieben und der Cursor geht eine Stelle weiter. Wenn der Cursor wieder auf den Seitenanfang

springt, wird ueberprueft, ob das naechste eingegebene Zeichen ein hexadezimales Digit ist oder nicht. Aenderungen im Speicher werden sofort ausgefuehrt, waehrend Aenderungen in einem Sektor gespeichert werden, bis der Sektor zurueckgeschrieben wird oder die Funktion mit 'Q' abgebrochen wird.

Leerzeichen oder rechter Pfeil	der Cursor rueckt eine Position vor
linker Pfeil	der Cursor geht eine Position zurueck
Shift rechter Pfeil	der Cursor geht vier Positionen vor
Shift linker Pfeil	der Cursor geht vier Positionen zurueck
Pfeil nach unten	der Cursor geht eine Zeile nach unten
Pfeil nach oben	der Cursor geht eine Zeile nach oben

ZT.. Dieses Kommando wird senkrecht in der 7. Spalte angezeigt und muss mit Druecken der ENTER-Taste angeschlossen werden. Alle Digits ab einschliesslich der momentanen Cursorposition bis einschliesslich dem 2. Digit der eingegebenen Zahl .. werden auf 00 gesetzt. Der Cursor wird auf das erste folgende Byte der eingegebenen Zahl .. positioniert.

RT..,xy Dieses Kommando ist aehnlich wie das Kommando ZT.., ausser dass die Bytes nicht auf 00 gesetzt werden, sondern das 1. Digit ersetzt wird durch das Digit x, das 2. Digit wird ersetzt durch y.

Q Gilt nur fuer Sektoroperationen. Die Modifizierung wird abgebrochen und SUPERZAP kehrt in den normalen Anzeigemodus zurueck, ohne die Aenderungen auf die Floppy zurueckzuschreiben.

ENTER Der Aenderungsmodus wird abgebrochen. Ist im Speicher geaendert worden, kehrt SUPERZAP in den Anzeigemodus zurueck. Bei Sektoraenderungen wird nachgefragt, ob der geaenderte Sektor auch wirklich auf die Floppy zurueckgeschrieben werden soll oder nicht. Wenn nein, springt SUPERZAP zurueck in den Aenderungsmodus, wenn ja, schreibt SUPERZAP den geaenderten Sektor auf die Floppy und kehrt dann in den Anzeigemodus zurueck.

Alle anderen Eingaben fuehren zu einem 'CHECK'.

Bei einer fehlerhaften Eingabe wird das Wort 'CHECK' senkrecht in der 7. Spalte angezeigt. Durch Eingabe von '*' kann die alte Funktion fortgefuehrt werden.

6.2 DISASSEM

Das Programm DISASSEM/CMD macht eine Rueckuebersetzung von Z80-Maschinencode in einen lesbaren Sourcecode. Dieser Sourcecode wird entweder auf dem Bildschirm oder auf Drucker und Bildschirm ausgegeben. Ferner kann dieses Programm eine Crossreference (Querverweis) erzeugen.

Dieses Programm ist identisch mit dem Programm frueherer NEWDOS-Versionen, es wurden nur die notwendigen Aenderungen eingebaut.

6.2.1 Object-Input

'OBJECT FROM MAIN MEMORY OR DISK?'

D oder nur ENTER Der Objectcode wird von der Diskette gelesen.

Auf die Frage 'FILESPEC?' wird der Name des zu uebersetzenden Programmes eingegeben.

Auf die Frage 'OFFSET OBJECT VIRTUAL ADDRESSES BY (HEX)?' gibt man entweder nichts ein, in diesem Falle benutzt der Disassembler die im Programm stehenden Adressen, oder man gibt eine hexadezimale Zahl mit 1-4 Stellen ohne 'H' am Ende als Offset zur Startadresse ein. Dies wird dann benoetigt, wenn das Programm zwar auf eine bestimmte Adresse geladen wird, aber dann in einem anderen Bereich laeuft. Beispiel: Das Objectmodul wird in den Bereich von C000H- OFFFFH geladen, wird aber sofort in den Bereich von 7000H- 0AFFFH geshiftet und dort ausgefuehrt. Als Startoffset gibt man in diesem Falle B000 ein, da C000 + B000 die Startadresse von 7000 ergibt.

Auf die Frage 'VIRTUAL RESTART LOCATION (HEX)?' kann entweder nichts oder aber eine hexadezimale Adresse (ohne H) eingegeben werden. Auf diese Art kann das Disassembly eines grossen Programmes an beliebiger Stelle fortgesetzt werden.

M Der zu uebersetzende Code soll aus dem Speicher gelesen werden.

Auf die Frage 'OBJECT VIRTUAL ADDRESS (HEX)?' wird in hexadezimal (1 bis 4 Zeichen ohne H) die Adresse angegeben, in der der Code normalerweise ausgefuehrt wird, als die Startadresse des Programmes. In der Liste ist das die erste Adresse.

Auf die Frage 'OBJECT REAL BASE ADDRESS (HEX)?' wird in hexadezimal (1 bis 4 Zeichen ohne H) die Adresse angegeben, in der der Code im Moment gerade steht (muss nicht die Startadresse sein).

6.2.2 Optionen

Eingaben auf 'ANY MORE OPTIONS?' (die ersten drei Eingaben koennen immer eingegeben werden, alle weiteren Eingaben nur dann, wenn der Object-Code von der Diskette kommt).

PTR Der Output geht auf den Printer.

BFSP Die normale Pause, die eingelegt wird, sobald der Bildschirm voll ist oder wenn die BREAK-Taste gedrueckt wird, wird uebergangen. Normalerweise kann man dann mit der ENTER-Taste fortfahren, Mit 'X' abbrechen oder mit 'V' vom Memory aus mit einer neuen Adresse fortfahren. Mit BFSP wird dies unterdrueckt, die Liste erscheint so schnell der

Disassembler uebersetzen kann. Diese Option wird automatisch mit der Option PTR aufgerufen.

nichts (nur ENTER) Es werden keine weiteren Optionen verlangt.

NCR Es wird keine Referenztabelle am Ende des Programmes erstellt und auch nichts ausgedruckt.

NIP Es erfolgt kein Ausdruck oder Anzeige des uebersetzten Programmes, dies hat aber keinen Einfluss auf die Referenztabelle.

STD Der rueckuebersetzte Code wird auf die Floppy geschrieben. Dieser Code kann von EDTASM verarbeitet werden.

FGN=xxx xxx darf aus 3 Zeichen bestehen und ist der Name, bei dem die Labels beginnen. Von diesem Namen an werden die Labels fuer Variable als Buchstaben hochgezaehlt. Wird er nicht eingegeben, erhaelt er den Defaultnamen AAA.

RTD Die Referenztabelle (Cross Reference Table) wird auf der Floppy abgespeichert. Nach dem normalen Uebersetzungsvorgang wird auf 'REFERENCE TABLE FILESPEC?' der Name der Referenztabelle eingegeben. Fuer zwei oder mehrere Programme koennen die Files mit der Referenzliste gemischt werden. Siehe Formatangabe am Ende dieses Abschnitts.

REA Die Referenztabelle wird komplett aufgebaut, alle Referenztypen werden eingesetzt.

RE& Die Referenztabelle enthaelt nur die Typen, die anstelle des '&' eingegeben werden. Dies kann entweder L, P, R, S, T, U, V, W oder X sein

RIA In der Referenztabelle werden keine Referenztypen angegeben.

RI& Die Referenztypen, die anstelle des '&' eingegeben werden, erscheinen nicht in der Referenzliste. Anstelle des '&' kann entweder L, P, R, S, T, U, V, W oder X eingegeben werden.

6.2.3 Funktion

Der Disassembler arbeitet in 4 Durchlaeufen:

1. Kommt der Objectcode von der Floppy und die Option 'NCR' wurde nicht angegeben, wird der Objectcode uebersetzt und dann die Referenztabelle erzeugt. Der uebersetzte Code wird zwar angezeigt, kann aber nicht unterbrochen werden. Sollte der Speicherplatz fuer die Referenztabelle nicht ausreichen, wird das Programm DISASSEM abgebrochen.

2. Wurde 'RTD' eingegeben, werden die Referenzen auf die Floppy geschrieben.

3. Uebersetzte Befehle anzeigen (Drucker oder Bildschirm). Je nach Eingabe werden nach jeder Seite Pausen gemacht. In Spalte 1 der Liste werden die Anzahl der Befehle angezeigt, die sich auf diese Zeile beziehen (0 = keine, F = 15 oder mehr). In Spalte 2 steht die Nummer des Bytes, auf welches sich die Referenz bezieht. Ist die Spalte 2 leer, beziehen sich alle anderen Befehle auf das 1. Byte dieser Zeile, andernfalls ist in binaer (hexadezimal ausgedruckt) angegeben, auf welches Byte Bezug genommen wird.

4. Wenn der Objectcode von der Diskette gelesen wird und nicht 'NCR' eingegeben wurde, wird die Referenztabelle gelistet. Zuerst wird die Art der Referenz gedruckt, und dann, in aufsteigender Reihenfolge, alle Adressen der Befehle, die sich auf diese Referenz beziehen. Der Zusatz zu jeder Adresse bezieht sich auf den Z80-Code, der diese Adresse anspricht.

Wenn der Disassembler einen Fehler im von der Floppy eingelesenen Code findet, wird die Meldung 'DISK OBJECT FILE FORMAT NOT AS EXPECTED' angezeigt und das Programm DISASSEM abgebrochen.

Wird waehrend des Ausdruckes die Taste 'P' gedrueckt, wird das Programm angehalten, mit der ENTER-Taste kann es fortgesetzt, mit der 'X'-Taste abgebrochen werden.

Wurde die Option PTR angegeben, muessen folgende Fragen beantwortet werden:

- Geben Sie auf die Frage '# OF LINES PER PAGE, EXCLUDING TOP AND BOTTOM MARGINS? (1-255)' ein, wieviele Zeilen auf einer Seite gedruckt werden sollen.
- Geben Sie auf die Frage '# LINES EACH FOR TOP AND BOTTOM MARGIN? (1-10)' die Anzahl der Zeilen an, die der Drucker am Anfang und am Ende jeder Seite freilassen soll. Gebn Sie 0 ein, wird kein Blattvorschub gemacht.
- Geben Sie auf die Frage 'REPLY "ENTER" WHEN PRINTER AT TOP OF PAGE' nur ENTER ein, sobald der Drucker am Seitanfang steht.
- Geben Sie auf die Frage 'HIGH ASCII CODE FOR PRINTER (5A-7F)' den hexadezimalen Wert ein, den Ihr Drucker verarbeiten kann.

Mit der STD-Option wird der uebersetzte Code auf die Floppy geschrieben im EDTASM-Format. Der Code kann dann vom EDTASM geladen werden (falls nicht zu gross) und ggf. geaendert. Es erscheint die Frage 'ASSEMBLER OUTPUT FILESPEC', auf die der Filename fuer den Source-Code einzugeben ist. Alle numerischen Werte werden ersetzt durch einen Namen mit 3 Zeichen Laenge, die Namen sind alphabetisch aufsteigend, beginnend von dem Namen, der bei FGN angegeben wurde, oder von AAA als Defaultwert. Wenn ein numerischer Wert auftritt, den es bereits gibt, so wird der entsprechende Name eingesetzt. Gibt es diesen Wert noch nicht, so wird der naechste Name vergeben und in eine Equate-Tabelle am Ende des Programms aufgenommen. ORG-Statements werden nach Bedarf aufgenommen, ein END-Statement am Ende der Liste eingefuegt.

Das Programm DISASSEM wird in den Speicher auf die Adresse 8200H geladen und auf die Startadresse von 5200H umgespeichert (siehe LMOFFSET) zur Ausfuehrung.

Das Format der Referenztabelle, die mit 'RTD' erzeugt wurde:

1. 1 Byte 0C0H (EOF) - zu ignorieren
2. 1 oder mehrere Eintraege in folgender Form:
   a. 2 Byte Adresse, 1. Byte die ersten 2 Stellen, 2. Byte die letzten 2 Stellen der Adresse.
   b. Kontrollbyte, Bits 7-0 (7 steht links):
      - 7-6 = 11 Letzter Eintrag in der Tabelle - alle anderen Bits und Bytes dieses Eintrags sind zu ignorieren.
      - 7-6 = 01 Referenzadresse. Die Bits 5-0 enthalten 0. Auf diese Adresse deuten eine oder mehrere der folgenden Adressen.

7-6 = 00 Adressen der Befehle, die sich auf die Adresse unter 1. beziehen. Die Bits 5-0 enthalten die Befehlsart dieser Adresse, wobei 0 = S, 1 = T, 2 = U, 3 = V, 4 = W, 5 = X, 8 = P, 9 = L und 10 = R bedeuteten.

6.3 LMOFFSET

Das Programm LMOFFSET/CMD liest LOAD-Module von der Kassette oder der Diskette, zeigt die Ladeinformationen an, aendert den Ladebereich, ermoeglicht es dem Programm, nach dem Laden sich in einem anderen Speicherbereich ausfuehren zu lassen und speichert das so geaenderte Programm unter einem neuen Namen auf der Diskette.

Das LMOFFSET hat unter NEWDOS/80 die gleichen Funktionen wie in frueheren NEWDOS-Versionen.

Funktion von LMOFFSET:

1. Lesen eines Assemblerprogrammes, das fuer die Kassette geschrieben wurde, von der Kassette, oder, wenn es fuer den Diskgebrauch geschrieben, von der Floppy. In diesem Falle muss der Filename eingegeben werden.

   - Ist das Programm auf der Floppy, so wird nach dem Filenamen gefragt.
   - Ist das Programm auf einer Kassette, so wird, sobald LMOFFSET bereit ist zum Einlesen, ein '*' angezeigt. Stellen Sie die Kassette auf den Anfang und druecken Sie PLAY. '***' werden angezeigt, sobald das Einlesen synchronisiert ist. 'C' wird angezeigt, wenn ein Checksum-Fehler aufgetreten ist, 'P' bei zusaetzlichen fuehrenden Bytes, 'I' bei sonstigen zusaetzlichen Bytes.

2. Am Bildschirm wird folgendes angezeigt:

   - Der Ladebereich des Programmes
   - Moegliche Konflikte mit dem Betriebssystem (Speicher)
   - Die Startadresse des Programmes

   Falls bereits ein Zusatz mit LMOFFSET am Anfang des Programmes gemacht wurde, die Ladeadresse des Programmes also bereits veraendert wurde, liegt die Startadresse in diesem 15 Bytes langen Zusatz.

3. Fragt nach der neuen Ladeadresse des Programmes. Es wird entweder die neue Ladeadresse eingegeben oder nur auf ENTER gedrueckt (falls die momentane Ladeadresse nicht geaendert werden soll). Falls eine neue Ladeadresse eingegeben wird, hat man die Moeglichkeit, den von LMOFFSET erzeugten Zusatz zum Verschieben zu unterdruecken. In diesem Falle kann das Programm jedoch nur ueber das DOS-Kommando 'LOAD' in den Speicher geladen werden, da der Zusatz zum Verschieben und der Sprung zur neuen Startadresse fehlen. Die eingesetzte Startadresse ist in diesem Falle 0. Ueber das Kommando 'LOAD' koennen dann ein oder mehrere Module wieder geladen und ueber 'DUMP' auf der Diskette gespeichert werden. Fuer eine neue Startadresse und Ladebereich kann man mit 2. fortfahren.

4. Wird nun eine neue Ladeadresse eingegeben und der 15 Bytes lange Zusatz nicht unterdrueckt, kann man noch

entscheiden, ob das Programm mit oder ohne Interrupts laufen soll. Bei einer Eingabe von 'Y' wird der Befehl 'DI' (Disable Interrupts) in den Zusatz eingefuegt, um die Interrupts von der CPU her zu stoppen.

5. Ein Filename muss eingegeben werden, unter dem dieses Programm auf der Floppy abgespeichert wird. Wurde eine neue Ladeadresse eingegeben

   a. wird diese Adresse in diesem Modul geaendert

   b. werden (falls nicht unterdrueckt) die zusaetzlichen Bytes generiert und vor den Record geschrieben, in dem die Startadresse enthalten ist.

   c. wurde der Zusatz unterdrueckt, wird die Startadresse auf 0 gesetzt.

6. Wurde bei Punkt 3 null eingegeben und danach ein neuer Ladepunkt angegeben und der Anhang nicht unterdrueckt wurde, fragt LMOFFSET ob das DOS aktiviert bleiben soll oder nicht.

7. LMOFFSET fragt als naechstes, ob auf das Kassettengeraet oder die Floppy ausgegeben werden soll.

   - Soll auf die Floppy ausgegeben werden, so muss der Filenamen eingegeben werden.

   - wird auf die Kassette ausgegeben, fragt das LMOFFSET nach dem Modulnamen und der Bandgeschwindigkeit (H oder L). Sobald die Kassette eingelegt ist, kann man ENTER druecken.

8. Das Modul wird jetzt auf die Floppy oder die Kassette geschrieben. Wurde ein neuer Ladepunkt vereinbart, wird

   a. die Ladeadresse jedes Records veraendert,

   b. wenn der Anhang nicht unterdrueckt wurde, werden die zusaetzlichen Bytes geschrieben, bevor der Eingangspunkt geschrieben wird,

   c. wenn der Anhang unterdrueckt wurde und eine neue Ladeadresse eingegeben wurde, wird der Eingangspunkt auf 0 gesetzt.

9. Sobald das Modul komplett geschrieben wurde oder wenn ein Fehler beim Schreiben aufgetreten ist, fragt LMOFFSET nach, ob das Modul nochmals abgespeichert werden soll. Die kann durchaus nochmals unter dem gleichen Namen sein.

10. Wenn alles fertig ist oder wenn ein Fehler (nicht beim Schreiben) aufgetreten ist, wird nach einem anderen Load-Modul gefragt.

Wenn ein derart geaendertes Programm mit Zusatz aufgerufen wird, startet dieses Programm mit der 1. Adresse des Zusatzes, die Interrupts werden abgeschaltet (falls nicht anders angegeben), der Objectcode in den alten Bereich transferiert und zur alten Startadresse gesprungen.

ACHTUNG !

Jedes Mal, wenn ein neuer Ladebereich ueber LMOFFSET angegeben wird, erhaelt das Programm den 15 Byte langen Zusatz ! LMOFFSET kann nicht erkennen, ob das Programm bereits schon einmal einen neuen Ladebereich erhalten hat !

LMOFFSET erzeugt im bearbeiteten Programm keine neuen Adressen, es verschiebt das Programm nur von dem neuen Ladebe-

reich auf die alte Adresse und startet die Ausfuehrung. Wenn das Programm im DOS-Bereich stand, ist groesste Vorsicht geboten beim Wiedereinschalten der Interrupts !

Hatte das Programm einen alten Ladebereich im Bildschirmspeicher (3C00H bis 3FFFH) ohne diesen zu ueberschreiten, wird keine neue Startadresse eingesetzt.

Wenn ein Programm in einem Teil des DOS-Bereiches ausgefuehrt wird (zwischen 4000H und 51FFH), muss man dafuer sorgen, dass entweder die Interrupts ausgeschaltet bleiben oder die Interruptvektoren entsprechend geaendert werden. Ferner muss die NEWDOS-Tastaturroutine abgeschaltet werden. In NEWDOS/80 und in den aelteren NEWDOS-Versionen konnte man das machen, indem man waehrend des Reset die Tasten Shift und Arrow Up gleichzeitig gedrueckt hat. In NEWDOS/80 kann auch der Systemparameter AJ=N eingegeben werden. Ferner muss auch beachtet werden, dass Systemroutinen im ROM verschiedene DISKBASIC-Vektoren zwischen 4100H und 41FFH aufrufen.

Wenn das zu aendernde Programm von LMOFFSET gelesen wird, wird es in einen Speicherbereich ab 6000H geschrieben. War das Programm auf der Diskette abgespeichert, steht genau die gleiche Information im Speicher, beim Laden von der Kassette wird es erst in das Disk-Format konvertiert. Der Benutzer kann sich das geladene Programm ueber DEBUG ansehen, es hat folgendes Format:

1. Object Code Record

   a. Kennbyte = 01H
   b. Anzahl der Objectbytes+2 (0-2 = 256-258)
   c. Ladeadresse (2 Bytes)
   d. Object-Bytes

2. Start Record

   a. Kennbyte = 02H
   b. Byte mit 02H
   c. Startadresse (2 Bytes)

3. Kommentar-Record

   a. Kennbyte = 0 oder zwischen 3 und 0FFH (bei Diskprogrammen) oder 0 bzw. zwischen 3 und 0FBH (bei Tapeprogrammen)
   b. Anzahl der Kommentarbytes
   c. Kommentarbytes

4. Record mit zusaetzlicher Information (nur wenn Programm von Tape geladen - es handelt sich um die Bytes, die auf der Kassette vor dem Synchronisationsbyte 55H (U) stehen).

   a. Kennbyte = 0FFH
   b. Byteanzahl (2 Bytes)
   c. Vorspann

5. Record mit Namen (nur wenn Programm von der Kassette geladen wurde) - es handelt sich um die Bytes, die auf der Kassette zwischen der Synchronisation 55H (U) und entweder 3CH (Kennbyte Objectcode) oder 78H (Kennung fuer eine Startadresse) stehen.

   a. Kennbyte = 0FEH
   b. Byteanzahl (2 Bytes)
   c. Name

6. Record mit den zusaetzlichen Bytes, die zwischen 2 Bloecken stehen (nur wenn Programm von der Kassette geladen wurde).

   a. Kennbyte = 0FDH
   b. Byteanzahl (2 Bytes)
   c. Zusaetzliche Bytes

7. Record mit falscher Pruefziffer (Checksum). Es folgt der Block mit der falschen Pruefziffer (nur wenn das Programm von der Kassette geladen wurde).

   a. Kennbyte = 0FCH
   b. Falsche Pruefziffer (1 Byte)
   c. Errechnete Pruefziffer (1 Byte)

Soll das Programm von der Kassette geladen werden, wird ein '*' von LMOFFSET angezeigt, sobald die Kassette gelesen werden kann. Nach der Bandpositionierung kann man am Recorder 'PLAY' druecken. '***' wird angezeigt, sobald die Synchronisation eingesetzt hat. Beim Lesen einer falschen Pruefziffer wird 'C' angezeigt, bei fuehrenden zusaetzlichen Bytes ein 'P' und bei zusaetzlichen Bytes zwischen zwei Bloecken ein 'I'.

6.4 DIRCHECK

Das Programm DIRCHECK/CMD testet und listet das Directory der eingegebenen Floppy. Werden Fehler gefunden, so druckt diese das Programm vor dem Directory-Inhalt aus. In der neuen Version kann auch das Directory bereinigt werden (aber nicht repariert). Es kann ebenfalls das Directory als zusaetzliche Option fuer den Austausch zwischen Modell I und III geschuetzt geschrieben werden.

Wird auf die Frage 'OUTPUT TO PRINTER' mit Y geantwortet, wird das Directory auf dem Drucker ausgegeben, bei N auf dem Bildschirm.

Auf die Frage 'WHICH DRIVE CONTAINS THE TARGET DISKETTE' gibt man die Laufwerksnummer (0...3) an.

DIRCHECK liest zuerst den BOOT-Sektor der Diskette und prueft, ob die ersten 2 Bytes = 00 FE sind. Trifft dies zu, so wird das 3. Byte als "Lump"-Nummer gelesen, in dem das Directory beginnt. Sind die ersten 2 Bytes nicht korrekt, so wird '***** DISKETTE 1ST SECTOR NOT "BOOT". ASSUMING DIRECTORY STARTS ON LUMP 17 DECIMAL' ausgegeben. angezeigt.

Erklaerungen und Fehler werden zuerst ausgedruckt. Die numerischen Werte in diesen Zeilen sind in hexadezimal, damit laesst sich das Directory leichter wieder ueber SUPERZAP herstellen. Bevor sie jedoch versuchen, ein defektes Directory zu reparieren, lesen sie sich genau die Beschreibung ueber das Directory durch!

DIRCHECK liest dann das Directory. In frueheren Versionen wurde dies abgebrochen, wenn das Directory nicht schreibgeschuetzt war. In der vorliegenden Version fuehrt dies wegen der Lesbarkeit von Modell I und III nicht mehr zum Abbruch, sondern nur noch zu 2 Fehlermeldungen, eine vor der Liste des Inhalts, eine nach dem Directory-Inhalt. Die Fehlermeldung lautet '***** AT LEAST ONE DIRECTORY SECTOR UNPROTECTED'. Wurden noch eine groessere Anzahl anderer Fehler gemeldet, kann man davon ausgehen, dass DIRCHECK das Directory nicht gefunden hat. In diesem Fall darf man das Kommando W nicht benutzen.

Folgende Angaben koennen dem 1. Teil der Fehlerliste entnommen werden:

FILE-Name Wenn entweder der erste oder der Erweiterungseintrag angegeben ist, handelt es sich bei dem angegebenen hexadezimalen Code um den Ersteintrag des Files (FPDE).

Extension-Eintrag	Wenn der Filename nicht angegeben ist, dann ist das hexadezimale Byte der DEC der File-Erweiterung
HIT-Sektor	Das angegebene Byte ist die relative Adresse im HIT-Sektor in hexadezimal
GAT-Sektor	Das angegebene Byte ist die relative Adresse im GAT-Sektor in hexadezimal
Granule	Das angegebene Byte ist die relative Adresse des Granules auf der Diskette in hexadezimal
Spur (Track)	Das angegebene Byte ist in hexadezimal die relative Spur auf der Diskette

Die dann ausgelisteten Files enhalten dezimale Werte und folgende Kennungen fuer den File-Typ:

S	Systemprogramme
I	Unsichtbare Files
P=nnn	File mit dem Zugriffslevel nnn, sowohl Update- als auch Zugriffspasswort sind ungleich Blank.
EOF=sss/bbb	End Of File, sss = Sektornummer im File, bbb = Bytenummer in diesem Sektor.
nnn EXTS	Anzahl der Fileerweiterungen, deren 1. Byte kleiner als 254 ist.
nnn Sectors	Anzahl der Sektoren fuer diesen File.

Zuletzt wird noch die Anzahl der freien Granules und die gesperrten Granules dieser Diskette ausgegeben. Hat die Diskette mehr als 60H (96 dezimal) "Lumps", oder ist das Byte 60H des GAT-Teiles 0FFH, so nimmt DIRCHECK an, dass keine Lock-Out-Table vorhanden ist.

Die Meldung 'FUNCTION COMPLETED' wird ausgegeben, danach folgende Frage:

```
REPLY
N      TO EXIT PROGRAM
Y      IF ANOTHER DISKETTE FOR SAME SPECS
I      FOR PROGRAM RE-INITIALIZATION
W      TO WRITE DIRECTORY SECTORS PROTECTED
C      TO CLEAN UP (NOT REPAIR) THE DIRECTORY
```

Geben Sie darauf folgendes ein:

N	Fuer Programmende, ueber einen Ruecksprung auf 402DH wird zum DOS zurueckgekehrt.
Y	Eine andere Diskette wird ueberprueft, jedoch muss diese die gleichen Daten betreffs Anzahl der Spuren, Output und Directory haben.
I	Eine andere Diskette mit anderen Daten soll soll ausgedruckt und getestet werden.
W	Die Directory-Sektoren werden gelesen und geschuetzt zurueckgeschrieben. Vergleichen Sie dazu das DOS-Kommando "WRDIRP" auf Seite 51 und die Option BN von "SYSTEM" auf Seite 46. Dieses Kommando hat nur dann Bedeutung, wenn Disketten zwischen Modell I und III getauscht werden.

C Alle nicht benutzten Directory-Eintraege werden komplett auf 0 gesetzt. Dies ist im Prinzip nur eine kosmetische Operation, die keine funktionelle Bedeutung hat. Alle geloeschten FDE's stehen noch im Directory, da nur das 4. Bit im ersten Byte geloescht wird beim KILL. Mit der restlichen Information kann dann ein geuebter Benutzer einen versehentlich geloeschten File wieder erzeugen, wenn jedoch zu viele schon geloeschte Files im Directory stehen, wird das Auffinden unnoetig erschwert.

Waehrend die Liste auf den Bildschirm oder den Drucker ausgegeben wird kann folgende Eingabe gemacht werden:

BREAK Die Liste wird nach der naechsten Zeile oder Zeilengruppe unterbrochen

ENTER Der Druck bzw. das Listen auf Bildschirm wird fortgefuehrt

UP-ARROW Das Listen bzw. Drucken wird abgebrochen.

6.5 EDTASM

Editor/Assembler fuer das Floppy-System

Vor ca. 2 Jahren hat die Fa. APPARAT den Editor/Assembler fuer das TRS80-Kassettensystem folgendermassen modifiziert:

1. Source-Programm sowohl von Diskette als auch von Kassette lesen
2. Source- und/oder Object-Programm sowohl auf die Diskette als auch auf die Floppy schreiben. Nach Abschluss wird der File auf der Floppy nochmals zur Kontrolle gelesen.
3. Mit Down-Arrow die naechsten 15 Zeilen weiterblaettern auf dem Bildschirm.
4. Symbole in alphabetischer Reihenfolge in einer Referenzliste ausgeben.
5. Unterdrueckung des zusaetzlichen Object-Codes bei dem Befehl 'DEFM'. Nur das 1. Byte wird gelistet.
6. Kleinbuchstaben in Grossbuchstaben konvertieren.
7. Im 'A'-Kommando Anzeige der noch zur Verfuegung stehenden Bytes im Textpuffer nach der Anzeige der Fehler ('TOTAL ERRORS').

Es wurde stark angenommen, dass die Fa. RADIO SHACK (TANDY) bald einen Floppy-orientierten Editor/Assembler herausbringen wird, die diese Version von APPARAT ueberfluessig machen wird. Da diese Version von APPARAT nur aus der alten TRS80 Version abgewandelt wurde, sind damit die Urheberrechte der Fa. TANDY betroffen. Aus diesem Grunde verlangt die Fa. APPARAT in jedem Falle den Kauf des Orginal-Editors/Assembler mit Kassetteneinund ausgabe vor Benutzung dieses Programms. Es werden in dieser Dokumentation nur die von TANDY unterschiedlichen Befehle beschrieben. Die Fa. APPARAT hat bisher nicht und wird auch in Zukunft nicht auskunft geben ueber die Standardbefehle aus dem Bereich der Kassettenversion von TANDY.

Dieser EDTASM ist gleich wie der EDTASM frueherer NEWDOS-Versionen.

Folgende zusaetzliche Befehle gibt es:

1. Laden des Source-Moduls in den Textpuffer.

   - L D=nnnnnnnn/ttt.ppppppp:d - laden von der Floppy
   - L T=nnnnnn - laden von der Kassette (gilt nur fuer Modell I, siehe dazu Aenderung #035)

   Enthaelt der Textpuffer bereits einen Text, wird 'TEXT IN BUFFER. ARE YOU CONCATENATING???' ausgegeben. Antworten Sie mit 'N', wenn der Puffer zuerst geloescht werden soll, bei der Eingabe von 'Y' wird der neue Text an den alten angehaengt. Es wird keine Meldung ausgegeben, wenn sich die Zeilennummern ueberlappen, aus diesem Grunde sollte nach Beendigung des Ladens das Kommando 'N' ausgefuehrt werden fuer ein Renumber.

2. Sichern des Textpuffers.

   - W D=nnnnnnnn/ttt.ppppppp:d - auf Floppy
   - W T=nnnnnn - auf Kassette (gilt nur fuer Modell I, siehe dazu Aenderung #035)

3. Auf die Frage beim A-Kommando 'OBJECT FILE TO DISK OR TAPE? REPLY D OR T?' kann man eingeben:

   T Der Objectcode wird auf die Kassette geschrieben, der Name wird dem A-Kommando entnommen.

   D Der Objectcode wird auf die Diskette geschrieben, auf die Frage 'OBJECT FILESPEC?' wird der Filename nnnnnnnn/ttt.ppppppp:d eingegeben. Der File wird zwar sofort eroeffnet, es wird jedoch erst dann auf die Floppy geschrieben, wenn der Listoutput beendet ist. Der beim A-Kommando eingegebene Name wird ignoriert.

4. Wenn ein Source- oder Object-File eroeffnet wird, wird in jedem Falle eine der folgenden Fragen angezeigt:

   - 'FILE ALREADY EXISTS. USE IT????'. Antworten Sie entweder mit 'Y' falls der vorhandene File ueberschrieben werden soll, oder druecken Sie die BREAK-Taste um das 'A'- oder 'W'-Kommando zu beenden.
   - '********** FILE NON-EXISTENT. REPLY 'C' TO CREATE IT'. Einen File mit diesem Namen gibt es noch nicht, durch Eingabe von 'C' wird er neu erstellt. Wird das nicht gewuenscht, kann man mit der BREAK-Taste die Funktion 'A' oder 'W' beenden.

5. Im Gegensatz zum Orginal-DOS laeuft dieser EDTASM mit einschalteten Interrupts (ausser wenn diese durch die Floppyein- bzw. Ausgabe ausgeschaltet werden), die BREAK-Taste kann normal benutzt werden.

6. Dieser EDTASM kann mit dem normalen TRSDOS arbeiten.

7. Dieser EDTASM benoetigt ein TRSDOS-aehnliches Betriebssystem und benutzt die Standard-Peripherie von Level II wie Tastatur, Bildschirm und Drucker-Routinen und Kontrollbloecke.

6.6 CHAINBLD/BAS

Das BASIC-Programm CHAINBLD/BAS ist ein recht einfaches Programm zum Erzeugen und Veraendern von CHAIN-Files.

CHAINBLD arbeitet im Record-Mode, d. h., dass jede Zeile mit einem EOL-Character abschliessen muss (0DH). Jede Zeile bildet einen eigenen Record im File. Jedes Einfuegen, Loeschen, Ersetzen und Kopieren erfolgt in Records.

Ferner werden im CHAINBLD keine Character in Steuerzeichen umgesetzt. Die einfache Regel ist, dass alles, was im BASIC-Kommando LINE INPUT akzeptiert wird, auch in den File aufgenommen wird. Die einzige Ausnahme bildet der EOL-Character. Falls der Benutzer unbedingt bestimmte Steuerzeichen im CHAIN-File benoetigt, muessen diese mit einem anderen Programm eingefuegt werden. Zur Not steht auch noch der SUPERZAP zur Verfuegung.

Nach dem Programmstart wird von CHAINBLD zuerst aller zur Verfuegung stehende Platz fuer die Strings initialisiert. Das dauert ca. 16 Sekunden. Bei Benutzung der Break-Taste zur Unterbrechung des Programmes muss darauf geachtet werden, dass bei der Fortsetzung des Programmes der Platz wieder reserviert wird.

Nach der Initialisierung wird das Hauptmenu angezeigt (nicht Verwechseln mit dem Edit-Menu). Folgende Eingaben sind moeglich:

1. DELETE ALL TEXT LINES

   Alle Textzeilen im String-Buffer werden geloescht, die Edit-Maske wird danach angezeigt. Beim Aufruf des Programms ist der Speicher leer.

2. LOAD EXISTING TEXT FROM DISK

   Diese Option benutzt man, wenn man einen bereits existierenden Textfile editieren will. Falls bereits Textzeilen im Buffer stehen wird gefragt, ob diese Zeilen geloescht werden koennen. Falls man N eingibt, kehrt das Programm in das Hauptmenu zurueck, bei Y wird der Buffer zuerst geloescht.

   Danach wird nach dem Filenamen des zu edtitierenden Files gefragt. Wird kein Zusatz zum Filenamen eingegeben (/xxx), so nimmt das Programm den Defaultwert /JCL an. Der File darf nicht groesser sein als der reservierte String-Platz und darf nicht mehr als 1000 Zeilen haben. Der File muss, wie oben erwaehnt, in Records unterteilt sein. Nach dem Laden wird das Edit-Menu angezeigt.

3. SAVE TEXT TO DISK

   Wenn man das Editieren oder die Neueingabe eines Files beendet hat, kann man diesen mit dieser Option auf der Diskette abspeichern. Fuer den Zusatz beim Filenamen gelten die gleichen Voraussetzungen wie beim Laden. Wenn der Text-Buffer keinen Text enthaelt kann man auch einen leeren File auf die Floppy schreiben. Als naechstes wird gefragt, ob der File im NEWDOS/80 Version 1 Format auf die Diskette geschrieben werden soll. In diesem Falle werden die Zeilen, die mit /./0 bis /./3 beginnen, umgesetzt in Zeilen mit dem Startcharacter 80H bis 83H. Der restliche Text wird nicht veraendert.

4. EDIT TEXT

   Mit dieser Option wird nur die Edit-Maske angezeigt.

5. EXIT PROGRAM

   Falls im String-Bereich des Programmes noch ein veraenderter Text steht, der noch nicht auf die Diskette geschrieben wurde, gibt das Programm eine Warnung heraus. Sie haben dann noch die Moeglichkeit, den Text auf die Floppy zu schreiben. Falls Sie das nicht wollen, wird der String-Platz auf 50 Bytes reduziert und das Programm beendet.

Wird das Edit-Menu angezeigt, haben Sie folgende Moeglichkeiten:

L oder ; Sie koennen den Text listen. Der Text wird in aufsteigender Reihenfolge durchnummeriert, unabhaengig davon, welche Aenderungen Sie darin machen. Die Zeilennummer erscheint jedoch nicht im Text-File. Damit kann aber eingegeben werden, ab wo Sie listen wollen oder wo eingefuegt, geloescht, ersetzt etc. werden soll. Mit L und ; kann der Text angezeigt werden. L; zeigt die erste Zeile an, L/ die letzte. Mit L 52 kann beispielsweise die 52. Zeile angezeigt werden. In jedem Falle werden die nachfolgenden Zeilen, falls vorhanden, mit angezeigt. Mit ; kann man vorwaerts blaettern.

I Mit dem Kommando I koennen Zeilen an einer bestimmten Stelle eingefuegt werden. Mit I 10 wird nach der 10. Zeile eingefuegt, mit I 0 vor der ersten Zeile, mit I / nach der letzten Zeile. Mit der Eingabe von /.// wird dieser Modus unterbrochen und ein neues Kommando kann eingegeben werden.

R Mit dem Kommando R kann eine Zeile an einer bestimmten Stelle ersetzt werden. Mit R43 kann eine neue Zeile 43 eingegeben werden.

D Mit dem Kommando D koennen eine oder mehrere Zeilen geloescht werden. Mit D34 wird die Zeile 34 geloescht, mit D 20 41 werden die Zeilen 20 bis 43 geloescht.

X Mit dem Kommando X kann an eine Zeile etwas angefuegt werden. Dies ist die einzige Moeglichkeit, wie in einer Zeile direkt etwas veraendert werden kann. Weitere Moeglichkeiten haben Sie nicht.

C Mit C koennen Zeilen kopiert werden. C 20 30 5 bewirkt, dass die Zeilen 20 bis 30 hinter der Zeile 5 eingefuegt werden. Die alten Zeilen 20 bis 30 bleiben bestehen und haben jetzt die Zeilennummer 31 bis 41.

M Mit M koennen Zeilen kopiert werden. C 20 30 5 bewirkt, dass die Zeilen 20 bis 30 hinter der Zeile 5 eingefuegt werden. Die alten Zeilen 20 bis 30 werden geloescht.

U Mit dem Kommando U wird das Edit-Menu angezeigt.

Q Mit dem Kommando Q wird das Haupt-Menu angezeigt.

Die beste Methode, das Programm CHAINBLD kennenzulernen, ist die, das Programm auszutesten. Auf der NEWDOS/80-Diskette befindet sich der File CHAINTST/JCL, mit ihm koennen Sie ueben. Speichern Sie ihn aber nicht unter dem gleichen Namen zurueck, benutzen Sie einen anderen Namen. Sie koennten ihn noch brauchen.

6.7 SPOOLER-PROGRAMM

6.7.1 Beschreibung

Das Objectmodul ASPOOL wird auf jeder NEWDOS/80 Diskette mitgeliefert und enthaelt das automatische Spooler-Programm von H. S. Gentry, das von APPARAT fuer die Benutzung unter NEWDOS/80 modifiziert wurde. Dieses Programm leitet alle Druckerausgaben automatisch auf die Floppy um und druckt dann auch automatisch. Das Spoolerprogramm arbeitet im Background, waehrend ein Anwenderprogramm im Foreground ausgefuehrt wird.

Es wird jede Sekunde ueberprueft, ob entweder eine Eingabe gemacht wird oder ob gedruckt werden kann.

Dieses Spoolerprogramm wurde erst in letzter Minute auf die NEWDOS/80 Diskette aufgenommen als freies Programm fuer die Anwender. Es unterliegt nicht der Wartung von APPARAT. Bei der Verwendung des Spoolers mit NEWDOS/80-Systembefehlen (z. B. PRINT, DIR P, JKL) ist groesste Vorsicht geboten. NEWDOS/80 nimmt an, dass die Ausgaben direkt auf einen Drucker und nicht auf die Floppy gehen. Es kann vorkommen, dass es manchmal problemlos geht, manchmal wird aber auch restlos alles zerstoert.

Mit diesem Spoolerprogramm kann eine Ausgabe nicht wiederholt werden. Ist ein Ausdruck erst einmal erfolgt, wird der Wert EOF auf 0 gesetzt und der File geschlossen, um Platz zu sparen. Das Spoolerprogramm kann auch nicht, nachdem es einmal abgebrochen oder beendet wurde, beim erneuten Start alte Ausgaben drucken, da es beim Aktivieren neu initialisiert wird. Der Anwender wird auch davor gewarnt, solange das Spoolerprogramm aktiv ist, Reset oder das Kommando BOOT zu benutzen um wieder auf DOS READY zu kommen. Nehem Sie statt dessen (es gibt keinen anderen Weg) DFG, um ins MINI-DOS zu kommen und brechen Sie es mit MDBORT ab, oder geben Sie im DEBUG (mit 123 aufzurufen) G402D ein. Nur so kommen Sie in den Status DOS READY.

6.7.2 Initialisierung

Bevor Sie mit dem Spoolerprogramm arbeiten koennen, muessen Sie eine lauffaehige Kopie des Objektmoduls ASPOOL erzeugen. Fuer unterschiedliche Anwendungen muessen Sie neue Kopien machen. Wenn Sie den FILENAME1 eingeben, muss dieser immer ASPOOL/MAS sein, bei FILENAME2 duerfen Sie nie ASPOOL/MAS eingeben !

Die Initialisierung geschieht folgendermassen: im DOS READY Status FILENAME1.1 eingeben (ASPOOL/MAS:0,I), das Programm fragt dann nach den einzelnen Parametern:

- Soll der Driver fuer den Drucker benutzt werden, dessen Adresse in 4026H und 4027H steht (Eingabe Y oder N). Bei N wird ein Treiber im Spooler benutzt, zur Auswahl muessen folgende Fragen beantwortet werden:
  - Hat der Drucker Parallel- oder Seriellschnittstelle (Eingabe P oder S). Bei Eingabe von S kann man noch eingeben, ob es sich um einen H14 handelt (Y oder N).
- Soll zwischen den einzelnen Seiten ein Formfeed (FF) ausgegeben werden (Y oder N), bei Y kann man eingeben,
  - wieviele Zeilen auf eine Seite gedruckt werden sollen (Zahl zwischen 10 und 99 eingeben).
- Soll der Drucker ein Hard- oder ein Soft-Formfeed bekommen (bei Soft-Formfeed wird die Anzahl der Zeilen mitgezaehlt, und mit CR/LF auf die naechste Seite hochgezaehlt, bei Hard- Formfeed wird nur das Zeichen FF gesendet). Eingabe hier H oder S. Bei S koennen Sie erneut eingeben,
  - wieviele Zeilen eine Seite hat (Zahl zwischen 10 und 99 eingeben).
- Soll nach einem Return automatisch ein Linefeed gesendet werden oder nicht (Eingabe Y oder N). Einige Drucker benoetigen das Linefeed nach dem Return, der Rechner liefert aber kein Linefeed. Andere Drucker erzeugen automatisch ihr Linefeed, z. B. der TRS80-Drucker.

- Das Spoolerprogramm fragt nach dem Laufwerk, auf dem die Daten gespoolt werden sollen. Geben Sie eine Zahl zwischen 0 und 3 ein.

- Das Spoolerprogramm fragt nach der Zeit nach der letzten Eingabe die es warten soll, bis der Ausdruck gestartet wird. Geben sie die Zeit in Sekunden 2-stellig ein (00-59). Wenn eine Zeit > 00 eingegeben wird heisst das, dass das Keyboard Prioritaet gegenueber dem Drucker hat. Wenn eine Taste an der Tastatur gedrueckt wird, waehrend ein File gerade ausgedruckt wird, so wird dieser Ausdruck solange unterbrochen, bis die eingegebene Zeit keine Taste mehr gedrueckt wird.

- Das Spoolerprogramm fragt, ob der Drucker mit dem 25 msec Interrupt betrieben werden soll oder nicht (Eingabe Y oder N). Wenn die Interrupts benutzt werden koennen vom Spoolerprogramm bedeutet das, dass waehrend ein Foregroundprogramm ausgefuehrt wird, das nicht staendig die Tastatur abfraegt, gedruckt werden kann. Der Nachteil mit den Interrupts ist der, dass der Output zum Drucker auch in einer Zeile unterbrochen werden kann. Wenn die Interrupts benutzt werden, wird der Ausdruck unterbrochen, solange nichts vom Anwenderprogramm ausgegeben wird oder die Tastatur abgefragt wird. Dies kommt daher, weil der naechste Sektor erst dann eingelesen lesen wird, wenn etwas ausgegeben oder die Tastatur abgefragt wird.

- Die letzte Eingabe, die zum Intialisieren benoetigt wird, ist, ob ein Buffer fuer die Eingaben zu verwenden ist oder nicht (Y oder N). Mit der gepufferten Eingabe wird verhindert, dass Eingabedaten verloren gehen. Wenn der 25 msec Interrupt zugelassen ist (s. vorangegangene Frage), benutzt das Programm den Keyboard-Treiber aus dem ROM und schaltet alle anderen Treiber aus (wie NEWDOS/80 Entprellung, Kleinschreibung etc.). Wenn der Interrupt nicht benutzt wird, bleiben die vorhandenen Treiber aktiv (Adresse in 4016H und 4017H). Diese Treiber bleiben ebenfalls aktiv, wenn die gepufferte Eingabe nicht benutzt wird.

Nach dieser letzten Eingabe beginnt die eigentliche Initialisierung, das Programm im Speicher wird entsprechend der Eingabedaten veraendert. Die letzte zu beantwortende Frage ist, unter welchem Filenamen das veraenderte Programm abgespeichert werden soll. Antworten Sie auf keinen Fall mit ASPOOL/MAS!!!

6.7.3 Aktivierung

Wenn Sie das Spooling benutzen wollen, geben Sie FILENAME2,A ein (FILENAME2 = der Name, unter dem das Programm nach der Initialisierung abgespeichert wurde, z. B. SPOOLER,A). Ist der Spooler bereits gestartet erscheint die Fehlermeldung FILE ALREADY EXISTS.

Das Modul wird in den Bereich zwischen 5200H und 52FFH geladen und shiftet sich selbst nach HIMEM-Programmgroesse + 1 und setzt HIMEM in 4049H und 404AH auf den neuen Wert. Dann wird der Keyboardtreiber in 4016H-4017H und der Druckertreiber in 4026H und 4027H auf den Wert des Spoolerprogramms gesetzt. Wenn die Interrupts benutzt werden, setzt sich der Spooler in die Interruptkette des Systems. Die Meldung SPOOLER ACTIVE wird noch angezeigt, dann wird das Spoolerprogramm ueber 402DH ins DOS verlassen.

Der Spooler ist jetzt aktiv. Alle Daten, die auf den Drucker ausgegeben werden sollen, werden in einen der 5 Diskfiles POOL1, POOL2, POOL3, POOL4 oder POOL5 geschrieben. Die 5 ver-

schiedenen Files aus folgendem Grunde: wurden genuegend Daten in den ersten File geschrieben und man wuenscht, dass sie jetzt auf den Drucker ausgegeben werden, gibt man End Of File aus (03H). Dies geschieht entweder ueber das DOS-Kommando *ASP,W (von BASIC aus ueber CMD"*ASP,W") oder durch drucken von 03H (von BASIC aus LPRINT CHR$(3)). Der File, auf den bisher gespoolt wurde, wird geschlossen fuer den Ausdruck. Kommen jetzt weitere Daten fuer den Drucker, so werden die auf den naechsten File geschrieben, solange der erste File gedruckt wird. Dies kann fuenfmal wiederholt werden. Wird es ein sechstes Mal versucht, so kommt die Meldung SPOOL FULL, WAITING ON PRINTER, und das Programm wartet, bis der erste File gedruckt ist. Alle Daten werden gedruckt, waehrend ein Programm laeuft oder das System auf ein Kommando wartet. Jedes Mal, wenn *ASP,W eingegeben wird oder ein Byte mit 03H ausgegeben wird, interpretiert dies das Spoolerprogramm als EOF und wechselt den File. Dies ist dafuer gut, dass der Drucker dadurch staendig laufen kann und es keine Platzprobleme im Spoolfile geben kann.

Um den Spooler zu stoppen gibt man *ASP,S ein (von BASIC aus CMD"*ASP,S") oder gibt ein Byte mit 04H aus. Diese Prozedur loescht den momentanen Spoolfile und verhindert, dass neue Spoolfiles angelegt werden. Die Meldung SPOOL STOPPING wird ausgegeben. Das Hauptprogramm wird fortgesetzt, und alle Zeichen, die noch an den Drucker geschickt werden, werden ignoriert. Der Spooler gibt noch alle beschriebenen Files aus. Wenn alle Files gedruckt sind, fuehrt der Spooler das Kommando *ASP,P aus. Beachten Sie, dass wenn der Drucker ohne ersichtlichen Grund stoppt, die eventuelle Ursache nur die ist, dass der Spooler auf das Druecken einer Taste wartet, damit er wieder einlesen kann.

Der Spooler kann jederzeit voll abgebrochen werden durch Eingabe von *ASP,P (von BASIC aus CMD"*ASP,P"). Alle gespoolten und noch nicht ausgegebenen Daten gehen dabei verloren. Falls eine Interruptroutine aktiv war, wird sie geloescht. Alle Treiberadressen werden auf den alten Wert zurueckgesetzt, der Wert HIMEM wird dann zurueckgesetzt, wenn er inzwischen nicht veraendert wurde. Die Meldung SPOOLER PURGED wird angezeigt, ueber 402DH wird ins DOS zurueckgesprungen.

Sie koennen durch die Eingabe von *ASP,C (von BASIC aus CMD"*ASP,C") den momentanen Print-File loeschen. Der Spooler antwortet mit CLEAR BACKLOG OR PRINT (B/P) ? Geben Sie B ein, wenn nicht nur der momentane File, sondern auch alle anderen noch zu druckenden Files geloescht werden sollen. Bei einer Eingabe von P wird nur die Ausgabe vom momentanen File gestoppt. Bei einer Eingabe von P werden nicht die alten Files geloescht, und bei einer Eingabe von B wird nicht der momentane File geloescht.

Eine Statusmeldung ueber den Spooler erhalten sie bei Eingabe von *ASP (CMD"*ASP" in BASIC). Es wird eine Liste aller Files ausgegeben, die auf das Drucken warten (BACKLOG) und der File, der auf Input steht fuer Print oder Spool. Wenn der Spooler gestoppt, aber nicht geloescht ist, kommt die Meldung SPOOL STOPPING, wenn der Spooler geloescht oder noch nicht aktiviert wurde, wird FILE NOT IN DIRECTORY ausgegeben.

7.0 BASIC TEIL 1 - OHNE EIN/AUSGABE

7.1 ALLGEMEINES

Fuer das NEWDOS/80 wurde der Teil des BASIC, der im RAM steht, neu geschrieben (Zusatzteil 'Disk-BASIC'). An den meisten Schnittstellen wurde nichts geaendert, sie sind groesstenteils wie im 'Disk-BASIC' unter TRSDOS 2.3 . Die Anwender von NEWDOS/80 werden darauf hingewiesen, dass sie sich sowohl im Level II BASIC Handbuch als auch im TRSDOS - Handbuch auskennen sollten. Dieses Kapitel und auch die naechsten folgenden befassen sich fast ausschliesslich mit dem Unterschied zwischen TRSDOS und NEWDOS/80.

Sollten sie noch kein TRSDOS - Handbuch haben, wird der Kauf dringend angeraten.

7.2 ALLGEMEINE HINWEISE

1. Tritt waehrend eines BASIC-Programmes ein Syntax-Fehler auf, geht das System nicht sofort in den Edit-Modus. Die Zeile mit dem Fehler wird nur als momentane Zeile gespeichert, kann also sofort bei Bedarf gelistet (Eingabe .) oder editiert (Eingabe ,) werden. Die Werte der Variablen bleiben dadurch erhalten und koennen auf diese Art kontrolliert werden.

2. BASIC-Programme koennen die Break-Taste sperren ueber das Kommando CMD"BREAK,N" oder freigeben ueber CMD"BREAK,Y".

3. Das Kommando CLOAD verliert die Synchronisation, wenn das BASIC mit mehr als 3 Filebereichen gestartet wurde.

4. Wenn keine Fehler-Routine im Programm vereinbart wurde (ON ERROR GOTO xxx), werden bei einem DOS-Fehler sowohl der DOS- als auch der zugehoerige BASIC-Fehler angezeigt.

5. Im BASIC gibt es jetzt 8 zusaetzliche Overlay-Programme, die bei Bedarf nachgeladen werden. Der Benutzer wird bemerken, dass diese Programme bei den Kommandos RUN, STOP, BREAK oder beim Auftreten eines Fehlers nachgeladen werden.

6. Im NEWDOS/80 Disk-BASIC ist es nicht erlaubt eine Zeile durch einfaches Eingeben der Zeilennummer zu loeschen. Es muss das Delete-Kommando D oder DELETE benutzt werden.

7.3 DISK - BASIC

Das Disk - BASIC kann mit folgenden Kommandos vom DOS aufgerufen werden :

```
BASIC
BASIC *
BASIC n
BASIC m
BASIC kmd
BASIC n,m,kmd
BASIC m,n,kmd
BASIC n,m
BASIC m,n
BASIC n,kmd
BASIC m,kmd
```

wobei die Abkuerzungen folgende Bedeutung haben :

M bedeutet, dass das BASIC nach seinem Aufruf keine Initialisierung durchfuehrt. Die vorher eingegebenen Parameter n oder m werden dem Memory entnommen. Ein eventuell vorhandenes BASIC - Programm wird nicht geloescht. Ist dieser Ruecksprung erfolgreich verlaufen, wird sofort die Funktion 'LIST' aufgerufen und das BASIC - Programm gelistet. Wurde vorher das BASIC-Programm von einem anderen Anwender- oder Systemprogramm zerstoert, ist ein Ruecksprung nicht mehr moeglich. Es erfolgt ein Sprung ins DOS, die Meldung 'DOS READY' erscheint.

BASIC * funktioniert nicht, wenn weniger als 2 I/O - Bereiche (n) definiert wurden oder das im Speicher stehende Anwenderprogramm kuerzer als 3 Zeilen war.

n Anzahl der zur Verfuegung stehenden I/O-Bereiche. Dies entspricht der maximalen Anzahl der gleichzeitig zu eroeffnenden Files. Der Defaultwert ist 3, das Maximum 15. Wird bei der Ausgabe auf die Floppy eine andere Recordlaenge benutzt als 256, muss nach der Anzahl der Filebereiche ein V eingegeben werden.

m Memory - Groesse. Diese Eingabe entspricht der Frage nach Memory-Size beim BASIC Level II. Der Defaultwert wird den Memory-Adressen 4049H (LSB) und 404AH (MSB) entnommen. Er kann dort ueber das DOS - Kommando 'HIMEM' veraendert werden.

kmd BASIC-Kommando oder -Befehl. Dieser String wird dem BASIC nach der Initialisierung uebergeben und vom BASIC ausgefuehrt.

Der komlette BASIC - Aufruf darf maximal 80 Zeichen haben, wenn das BASIC ueber das Kommando 'AUTO' aufgerufen wird maximal 32 Zeichen.

Jeder Fehler bei der Initialisierung des BASIC fuehrt zu einem Ruecksprung ins DOS.

Wenn das DOS im 'RUN-Mode' ist (siehe "SYSTEM" auf Seite 46), muss BASIC mit dem Parameter kmd (RUN) aufgerufen werden.

Beispiele:

1. BASIC

   Dieses Kommando laedt das BASIC mit 3 Filebereichen, der Wert von HIMEM wird als Obergrenze festgelegt und 'READY' angezeigt. Der Benutzer kann Kommandos eingeben.

2. BASIC,RUN"XXX/BAS"

   Dieses Kommando laedt das BASIC mit 3 Filebereichen, der Wert von HIMEM wird als Obergrenze festgelegt, das Programm XXX/BAS geladen und gestartet.

3. BASIC,9,48152,LOAD"XXX/BAS"

   Dieses Kommando laedt das BASIC mit 9 Filebereichen, der Wert 48151 wird als Obergrenze festgelegt, das Programm XXX/BAS wird geladen. Es wird 'READY' angezeigt, der Benutzer kann jetzt Kommandos eingeben.

4. BASIC,3V

   Dieses Kommando ist im Prinzip gleich wie das erste Beispiel, nur dass zu den 3 Filebereichen jeweils ein zusaetzlicher Buffer von 256 Bytes freigehalten wird. Diesen zusaetzlichen Buffer benoetigt das BASIC zur Abarbeitung von Files mit einer Recordlaenge von ungleich 256 Bytes.

5. BASIC,CLEAR3000:A=1:RUN"XXX",V

Mit diesem Kommando wird das BASIC mit 3 Filebereichen geladen, die Memory Obergrenze von HIMEM uebernommen, es werden 3000 Bytes reserviert fuer den Stringbereich, das Programm XXX geladen und gestartet ohne dass die Variablen geloescht werden.

7.4 ZUSAETZLICHE DIREKTE KOMMANDOS

Das NEWDOS/80 BASIC erlaubt folgende direkte Kommandos:

. (Punkt)	Listen der momentanen Textzeile
Down - Arrow	Listen der naechsten Textzeile. Bei Textende listen der letzten Zeile.
Up - Arrow	Listen der vorhergehenden Textzeile. Am Textanfang listen der 1. Zeile.
o. Shift-Up-Arrow	erste Textzeile im Programm listen
/ o. Shift-Down-Arrow	letzte Textzeile listen. Benutzer mit dem neuen ROM haben keinen eigenen Character fuer 'Shift-Down-Arrow'. Sie muessen '/' benutzen.
, (Komma)	editieren der momentanen Textzeile

Es ist nur eines dieser Kommandos pro Eingabezeile erlaubt, dieses Zeichen muss auch an der ersten Stelle der Eingabezeile stehen (keine Zeilennummern oder Backspace erlaubt).

7.5 ABKUERZUNGEN

Im NEWDOS/80 Disk - BASIC sind fuer die Kommandos AUTO, DELETE, EDIT und LIST folgende Abkuerzungen erlaubt:

A	(AUTO)
D	(DELETE)
E	(EDIT)
L	(LIST)

Folgende Voraussetzungen muessen aber erfuellt sein :

1. Die Abkuerzung steht an 1. Stelle der Zeile
2. Nach der Abkuerzung muss eine Dezimalzahl oder ein Punkt stehen
3. Die Eingabezeile darf kein '=' enthalten

7.6 ZUS. EDIT-KOMMANDOS

Zwei zusaetzliche Edit-Kommandos wurden mit dem NEWDOS/80 ein-

gefuehrt :

```
DI aaaaa,bbbbb
DI.,bbbbb
DU aaaaa,bbbbb
DU .,bbbbb
```

aaaaa ist die alte, existierende Zeilennummer,

bbbbb ist die neue Zeilennummer, an der die Zeile eingefuegt werden soll.

Wird anstelle von aaaaa ein '.' verwendet, wird die letzte editierte Zeile nach bbbbb verschoben.

DI bedeutet Delete and Insert (loeschen und einfuegen),

DU Duplicate (duplizieren).

7.7 RUN UND LOAD

Mit diesen Kommandos kann man jetzt bestimmen, dass die Variablen bei Abbruch oder Ende nicht geloescht werden, ebenso bleiben Filebereiche gueltig, es wird also kein CLOSE gemacht. Diese Option wird folgendermassen eingegeben:

```
RUN"fname1",V
LOAD"fname1",V
```

wobei fname1 der Filenamen des zu ladenden bzw. startenden Programmes ist. LOAD mit der Option V arbeitet gleich wie RUN mit der Option V. RUN mit V loescht die Variablen nicht beim Start, ausser der DEFFN Variablen. Die Variablen haben also nach Programmende noch den gleichen Wert wie in dem Programm. Alle Files, die vor dem RUN eroeffnet waren, bleiben auch beim Start des Programmes eroeffnet und darauf kann zugegriffen werden, auch wenn das Programm beendet ist. Mit der Option V kann nicht gleichzeitig R angegeben werden.

7.8 MERGE

Das Kommando MERGE wurde so erweitert, dass auch BASIC-Files benutzt werden koennen.

MERGE kann sowohl als Kommando als auch als Statement ausgefuehrt werden.

Wenn MERGE im Programm auftritt, darf es nicht Bestandteil eines DEFFN Statements sein, einer Subroutine oder eines FOR...NEXT Loops. Es muss das letzte Statement einer Zeile sein, die naechste auszufuehrende Zeile muss dieser Zeile folgen nach dem MERGE, und der einzufuegende File darf keine Zeilennummer enthalten die kleiner ist als die des MERGE-Statements. Benutzen Sie das Kommando CMD"F",DELETE um die Zeilen zu loeschen, die mit dem Kommando MERGE in Konflikt kommen.

Alle Variablen bleiben erhalten beim MERGE. Der Benutzer muss sicherstellen, dass genuegend Platz im Memory fuer das hinzu-

zufuegende Programm ist, da ein Fehler sonst nicht behoben werden kann.

Beispiel:

```
100 MERGE"XXX/BAS"
110 X = 1
```

Die Ausfuehrung dieses Programms geht nach dem MERGE-Statement mit der Zeile 110 weiter.

7.9 RENUM

Neues Durchnummerieren des Basicprogramms. Das Kommando RENUM hat verschiedene Formate :

```
1.  RENUM sssss,iiiii,ppppp,qqqqq<,U><,X>
2.  RENUM ,
3.  RENUM U
4.  RENUM X     * entfaellt lt.
5.  RENUM U,X   * Aenderung #029
```

Das momentan im Speicher stehende BASIC-Programm wird neu durchnummeriert und bleibt dabei im Speicher. Mit der Option U wird keine neunummerierung vorgenommen, es wird nur der Text auf fehlende Zeilennummern und Fehler durchsucht. Der Benutzer kann, bei richtiger Angabe der neuen Zeilennummern, einen Teil des Programmes in einen anderen Bereich verlagern.

Folgende Fehlermeldungen koennen beim Parameter U auftreten:

sssss/U Textzeile sssss fehlt

sssss/X Textzeile sssss hat einen Syntaxfehler

sssss/S Textzeile sssss hat eine falsche Zeilennummer

RENUM sssss,iiiii,ppppp,qqqqq hat folgende Funktion :

Alle Zeilen, deren Zeilennummern groesser oder gleich ppppp und kleiner oder gleich qqqqq sind, erhalten die neue Zeilennummer sssss, die sich mit jeder Zeile um den Faktor iiiii erhoeht.

sssss und iiiii muessen im Bereich zwischen 1 und 65529 sein, ihr Defaultwert ist 10. ppppp muss ebenso im Bereich zwischen 1 und 65529 liegen, der Defaultwert ist 0. qqqqq muss ebenso zwischen 1 und 65529 liegen, der Defaultwert ist 65529.

Die Defaultwerte koennen wie folgt benutzt werden :

```
RENUM ,
RENUM sssss
RENUM ,iiiii
RENUM ,,ppppp
RENUM ,,,qqqqq
RENUM sssss,iiiii
RENUM sssss,,ppppp
RENUM ,iiiii,ppppp
RENUM sssss,,,qqqqq
RENUM ,iiiii,,qqqqq
RENUM ,,ppppp,qqqqq
RENUM sssss,iiiii,ppppp
```

```
RENUM sssss,iiiii,,qqqqq
RENUM sssss,,ppppp,qqqqq
RENUM ,iiiii,ppppp,qqqqq
```

Wenn ein Fehler auftritt bevor der Text geaendert wird, also bevor die Zeilennummern geaendert werden, geht das RENUM .... automatisch in den Modus RENUM U ueber. Alle Zeilen mit einem Zehler werden angezeigt, der Text wird nicht veraendert.

Wenn der Fehler nach dem Start der Zeilenaenderung auftritt, kehrt das Renumber-Programm zum DOS zurueck und gibt die Fehlermeldung 'FATAL ERROR. TEXT NOW BAD' aus. Der Text wurde in diesem Fall nur teilweise geaendert, eine Rueckkehr ins BASIC mit BASIC * ist nicht zu empfehlen.

Wenn sich die Systemprogramme SYS11/SYS oder SYS13/SYS beim Aufruf von RENUM nicht im System befinden, kehrt das BASIC automatisch zum DOS zurueck.

Beispiel:

1. RENUM U

   Der BASIC-Text wird durchsucht nach undefinierten Zeilennummern. Ferner werden die Fehler angezeigt, die sonst auch noch beim RENUM auftreten koennen. Der Text wird nicht veraendert.

2. RENUM

   Der BASIC-Text wird neu durchnummeriert, der neue Zeilenabstand betraegt 10. Die erste Zeile hat die Nummer 10, die zweite Zeile 20 usw.

3. RENUM 100,100

   Der BASIC-Text wird neu durchnummeriert, der neue Zeilenabstand betraegt 100. Die erste Zeile hat die Nummer 100, die zweite Zeile 200 usw.

4. RENUM 2050,,2050,3160

   Alle Textzeilen von und inclusive der Zeilennummer 2050 bis und inclusive der Zeilennummer 3160 werden mit dem Erhoehungsfaktor 10 neu durchnummeriert. Die erste Zeile hat dann die Zeilennummer 2050, die zweite Zeile die Nummer 2060 usw.

5. RENUM 30000,5,15365,18112

   Alle Textzeilen von und inclusive der Zeilennummer 15366 bis und inclusive der Zeilennummer 18112 werden mit dem Erhoehungsfaktor 5 neu durchnummeriert. Die erste Zeile hat dann die Zeilennummer 30000, die zweite Zeile die Nummer 30005 usw. Der Text mit der neuen Zeilennummer wird an die der Nummer entsprechende Stelle verschoben.

7.10 REF

Dieses Programm erzeugt eine Referenzliste eines BASIC-Programmes mit Zeilennummern und Variablennamen.

Folgende Aufrufe sind moeglich :

REF*	(1) Komplette Referenzliste erzeugen und auf den Bildschirm ausgeben.
REF$	(2) Komplette Referenzliste erzeugen und auf den Drucker ausgeben.
REFnn	(3) Referenzliste von allen Variablen mit dem Namen nn. Hat der gewuenschte Variablenname nur einen Buchstaben, muss dieser ergaenzt werden mit einem Leerzeichen. nn darf nicht laenger als 2 Buchstaben sein.
REFsssss	(4) Referenzliste von allen Zahlen mit dem Wert sssss. sssss darf maximal 5 Ziffern lang sein und liegt zwischen 1 und 99999. Hexadezimale oder oktale Zahlen werden nicht angezeigt.
REF*nn	(5) 1 und 3
REF$nn	(6) 2 und 3
REF*sssss	(7) 1 und 4
REF$sssss	(8) 2 und 4
REF	(9) Die naechste Textzeile mit mindestens einer Referenz auf eine Zahl oder einen Variablenname des letzten REFnn oder REFsssss wird angezeigt. Falls keine weiteren Zeilen mehr vorhanden sind, wird 'TEXT END' angezeigt. Wird REF nochmals eingegeben, wird die erste Textzeile angezeigt.
REF=xxx	(10) Die Zeichen xxx werden ueber die Standardroutine im BASIC gepackt. Dieser Text wird dann im Programm verglichen und von allen Zeilen, die diesen Text enthalten, die Zeilennummer ausgedruckt. Folgen keine Textzeilen mehr, wird 'TEXT END' ausgedruckt. Wenn der gepackte Wert laenger ist als 16 Bytes, so wird nur nach den ersten 16 Bytes gesucht. Diese Art von REF koennen Sie benutzen, wenn Sie wissen wollen, wo z. B. ein bestimmter Befehl im Text auftritt, z. B. GOTO, LPRINT etc.
REF"xxx	(11) Dieses Kommando arbeitet gleich wie das vorherige, mit dem Unterschied allerdings, dass der gesuchte Text xxx nicht gepackt wird. In diesem Fall koennen Sie also nach Kommentaren oder Strings suchen.
REF@sssss	(12) Dieses Kommando arbeitet gleich wie das Format 9, ausser dass erst ab einer Zeilennummer von groesser sssss gesucht wird.

Die bestehende Referenzliste wird normalerweise (nicht immer) geloescht, wenn eine neue Zeile oder ein Befehl eingegeben wird. Die Anzeige der Liste kann durch Druecken der 'BREAK'-Taste unterbrochen werden, mit 'ENTER' wird sie fortgesetzt und mit 'Up-Arrow' kann sie abgebrochen werden. Bei den Formaten 5 bis 8 werden, falls vorhanden, die eingegebenen Variablen oder Zahlen angezeigt, wenn diese Namen oder Zahlen nicht vorhanden sind, jeweils die naechst hoeheren.

Wenn sich das Systemprogramm SYS12/SYS beim Aufruf von REF nicht im System befindet, kehrt das BASIC automatisch zum DOS zurueck.

7.11 KLEINSCHREIBUNGSUNTERDRUECKUNG

Benutzer, die keine Kleinschreibung installiert haben, oder solche, die es zwar installiert haben, es aber nicht benutzen, wundern sich manchmal darueber, warum manchmal ein String nicht gefunden wird oder warum ein Syntax-Fehler auftritt, obwohl der eingegebene Text in Ordnung zu sein scheint. Das kann daher kommen, dass zwar Kleinbuchstaben als Eingabe in Strings akzeptiert werden, ebenso das kleine ä in Statements (bei PRINT ä .), jedoch als Grossbuchstaben angezeigt werden. Leider kann dagegen nichts anderes getan werden, als die Systemoption AS=Y einzugeben, damit alle Kleinbuchstaben in Grossbuchstaben umgewandelt werden. Sie sparen sich aber dadurch viele Probleme.

7.12 RUN-MODE

Im Disk-BASIC gibt es 2 Moeglichkeiten, das BASIC in den sogenannten 'RUN ONLY MODE' zu versetzen :

- Als Systemoption AB wird 'Y' eingegeben,
- Der BASIC-File wird ueber ein Passwort mit dem Level 'EXEC' geschuetzt.

Wenn bei der Systemoption AB=Y von DOS her das BASIC aufgerufen wird, muss dies mit der gleichzeitigen Eingabe eines RUN-Kommandos oder ueber LOAD (mit RUN) erfolgen. Der Anwender hat keine Moeglichkeit, ein Statement einzugeben.

Im 'RUN-ONLY-MODE' ist die BREAK-Taste gesperrt, das BASIC akzeptiert keine direkten Statements (nur DATE-Eingabe) vom Benutzer. Das geladene Programm bleibt immer aktiv und muss ausgefuehrt werden. Ueber ein Menu-Programm hat man die Moeglichkeit, in den Statements ueber 'RUN' oder 'LOAD' (mit Option RUN) verschiedene andere Programme aufzurufen. Es sei darauf hingewiesen, dass das 'LOAD' keine eroeffneten Files schliesst, und dass weder LOAD noch RUN etwas an den File - Puffern veraendern. In NEWDOS/80 kann man 'FIELD' auch bei geschlossenen Files ausfuehren. Auf diese Art kann man Daten von einem Programm an das nachfolgende uebergeben ohne sie auf Floppy zu schreiben.

7.13 CMD"X"

Das BASIC-Statement CMD"xx" wurde erweitert. Wenn xx mehr als ein Zeichen lang ist und am Ende ein $ hat wird angenommen, dass es sich um eine Stringvariable handelt. Der Inhalt dieser Stringvariablen wird als Text fuer xx interpretiert.

Wenn xx nur aus einem Zeichen besteht wird die normale Verarbeitung fuer CMD"S", CMD"D", CMD"E", CMD"R" und CMD"T" gestartet.

7.13.1 CMD"A"

Nicht benutzt im NEWDOS/80, benutzen Sie CMD"S".

7.13.2 CMD"B"

Nicht benutzt im NEWDOS/80, benutzen Sie CMD"S".

7.13.3 CMD"C"

Dieses Kommando hat folgende Funktionen:

1. Alle Leerzeichen aus dem Text entfernen, ausser innerhalb von Strings.

2. Alle Bemerkungen (REM und ') aus dem Text entfernen, auch in solchen Zeilen, wo die Bemerkung als Folgestatement steht.

Das Kommando CMD"C",S loescht alle Leerzeichen im Text, ausser in Strings, jedoch auch im DATA-Statement. Mit dem Kommando CMD"C",R werden alle Bemerkungen (also REM's und ') geloescht.

- In einigen Faellen kann es vorkommen, dass ein GOTO oder GOSUB eine Zeilennummer enthalten, die nur aus einer Bemerkung besteht, z. B. den Kopf mit einer Erlaeuterung einer Subroutine. In solchen Faellen muss das Programm umgeschrieben werden, damit dieses GOTO oder GOSUB auf eine Zeile mit einem 'echten' Statement geht. Nach einem CMD"C" koennen die fehlerhaften Statements mit RENUM U leicht gefunden werden.

- Das BASIC wurde so entwickelt, dass Leerzeichen in einer Zeile ausser bei Strings ueberlesen werden. Aus diesem Grunde macht es nichts aus, wenn die Leerzeichen entfernt werden. Es kann jedoch unter gewissen Umstaenden zu Konfliktsituationen kommen. Z. B. die Statements

```
10 FIELD 1,20 AS C$
20 IF F OR D THEN 10
```

  werden umgewandelt in

```
10 FIELD1,20ASC$
20 IFFORDTHEN10
```

  Beide Statements bekommen bei der Ausfuehrung einen Sysntax-Fehler, wenn sie

  - einmal im ASCII-Format abgespeichert und zurueckgeladen werden, oder
  - einmal editiert werden.

  Um diesen Fehler zu vermeiden, werden alle Statements zuerst gepackt, dann nochmals entpackt und das Ergebnis mit der Orginal-Zeile verglichen. Kommt es nicht zur Uebereinstimmung, so wird die Zeilennummer ausgegeben und die Zeile bleibt unveraendert.

7.13.4 CMD"D"

Die eigentliche Bedeutung vom TRSDOS ist im NEWDOS/80 nicht implementiert fuer das Modell III. Hier koennen Sie die Funktion ueber CMD"doscmd" aufrufen. Im NEWDOS/80 fuer Modell I ruft das Kommando CMD"D" das DEBUG ueber '123' auf.

7.13.5 CMD"E"

Zeigt die letzte DOS-Fehlermeldung an, die an das BASIC uebergeben wurde. 'NO ERROR' bedeutet, dass noch kein Fehler aufgetreten ist.

7.13.6 CMD"F"

Wurde nicht benutzt im TRSDOS. Im NEWDOS gibt es zwei verschiedene Formate.

1. CMD"F",fc

   Diese Art wird benutzt, wenn fc (Funktions-Code) von den Kommandos RENUM und REF veraendert werden kann und soll, also eine Zeilennummer enthaelt.

2. CMD"F=fc"

   Diese Art wird benutzt, wenn fc (Funktions-Code) von den Kommandos RENUM, REF und CMD"C" nicht veraendert werden kann oder soll, da sowohl REF als auch RENUM, wie auch CMD"C" Zeichen innerhalb eines Strings ignorieren.

Die verschiedenen Funktionscodes (fc) werden in "CMD"F=POPx"" auf Seite 123 bis "CMD"F=SS"" auf Seite 125 naeher erlaeutert.

7.13.7 CMD"I"

Wird beim Modell I weder im TRSDOS noch im NEWDOS/80 benutzt. Beim Modell III benutzen Sie bitte im NEWDOS/80 die Funktion CMD"doscmd".

7.13.8 CMD"J"

Datumskonvertierung mit folgender Syntax:

CMD"J",datum1,datum2

Dieses Kommando hat folgende Funktion:

Der Ausdruck datum1 wird in ein anderes Format (siehe unten) konvertiert und in die Stringvariable datum2 geschrieben. Wenn datum1 das Format mm/tt/jj hat (beachten Sie das amerikanische Format, bei dem zuerst der Monat kommen muss), erhaelt man in datum2 den entsprechenden Tag im Jahr in der Form ddd.

Ist datum1 im Format -jj/ddd erhaelt man in datum2 das Datum in der Form mm/tt/jj, wobei

tt ist der Tag, 2 Character lang und liegt zwischen 01 und 31.

mm ist der Monat, 2 Character lang und liegt zwischen 01 und 12.

jj ist das Jahr, 2 Character lang und liegt zwischen 00 und 99. Fuer die Konvertierung benoetigt das System die Jahresangabe, um Schaltjahre erkennen zu koennen.

Wegen der Erkennung von Schaltjahren muss jj im 20. Jahrhundert sein, also zwischen 1900 und 1999.

7.13.9 CMD"L"

Die Funktion dieses Kommandos von TRSDOS fuer Modell III ist nicht implementiert im NEWDOS/80. Benutzen Sie dafuer CMD"LOAD,fname". Fuer Modell I hat dieses Kommando keine Bedeutung.

7.13.10 CMD"O"

Sortieren von Feldern. Siehe Beschreibung "CMD"O"" auf Seite 126.

7.13.11 CMD"P"

Nicht benutzt im Modell I. Benutzen Sie im Modell III den Befehl PEEK(&H37E8) fuer die Erkennung des momentanen Printerstatus.

7.13.12 CMD"R"

Die urspruengliche Bedeutung des TRSDOS im Modell III ist nicht implementiert. Benutzen Sie dafuer CMD"CLOCK,Y". Im Modell I werden die Interrupts wieder eingeschaltet.

7.13.13 CMD"S"

Das BASIC wird verlassen und das NEWDOS/80 wird aufgerufen. Auf dem Bildschirm wird DOS READY angezeigt.

Ist das Kommando in der Form CMD"S=doscmd", wird folgendes ausgefuehrt :

- Der Rest des Kommandos, ausser 'S=' wird in den DOS-Eingabepuffer gestellt wie ein DOS-Kommando.
- Das BASIC wird verlassen wie bei CMD"S".
- Das im Puffer stehende Kommando wird von DOS sofort ausgefuehrt, ohne dass 'DOS READY' vorher angezeigt wird.
- Ist das Kommando ausgefuehrt, wird die Kontrolle dem DOS uebertragen und nicht mehr dem BASIC.

7.13.14 CMD"T"

Die urspruengliche Bedeutung des TRSDOS im Modell III ist nicht implementiert. Benutzen Sie dafuer CMD"CLOCK,N". Im Modell I werden die Interrupts ausgeschaltet.

7.13.15 CMD"X"

Die urspruengliche Bedeutung des TRSDOS im Modell III ist nicht implementiert. Benutzen Sie dafuer das REF-Kommando. Wird nicht benutzt im Modell I.

7.13.16 CMD"Z"

Die urspruengliche Bedeutung des TRSDOS im Modell III ist nicht implementiert. Benutzen Sie dafuer CMD"ROUTE,...". Wird nicht benutzt im Modell I.

7.14 CMD"DOSCMD"

Wenn xx mehr als ein Zeichen hat und die ersten zwei Zeichen von xx ungleich "S=" oder "F=" sind, dann wird angenommen, dass xx ein DOS-Kommando ist. BASIC schreibt den String xx in den DOS-Eingabepuffer, setzt DOS in den MINI-DOS Status und macht einen Aufruf ueber die Routine 4419H, den DOS-CALL. beim Ruecksprung in das BASIC wird dieser Status geloescht. Wurde das aufgerufene Kommando von DOS zurueckgewiesen, weil es beispielsweise nicht zulaessig war im MINI-DOS, wird folgendes ausgefuehrt:

Wenn weniger als 8000 Bytes zwischen dem BASIC-Stack und dem Variablenbereich frei sind im Speicher, wird der Fehler 'OM' (OUT OF MEMORY - Memory Ueberlauf) angezeigt und das Kommando abgebrochen. Wenn ein groesserer Platz als 8000 Bytes in diesem Bereich frei ist, wird der ganze Inhalt von 5200H bis 70FFH in diesen Bereich geschrieben, der Stack wird auf den Bereich von 7000H - 70FFH gesetzt. Dann wird die Checksum (Quersumme) ueber den Bereich von 7100H bis zum Ende des von BASIC benutzten Speicherplatzes errechnet (dauert ca. 2 Sekunden). Dann erst wird das DOS aufgerufen, um dieses Kommando auszufuehren. Beim Ruecksprung von DOS nach Ausfuehren dieses Kommandos ins BASIC wird der Bereich 5200H - 70FFH wieder zurueckgeschrieben, die Checksum nochmals errechnet (nochmals ca. 2 Sekunden), und mit der alten Checksum verglichen. Stimmen diese Quersummen nicht miteinander ueberein, was bedeutet, dass das DOS einige Bytes des BASIC zerstoert hat, bedeutet das, dass das BASIC nicht fortgesetzt werden kann, es meldet an DOS den Fehler 'BAD MEMORY' und kehrt in das DOS zurueck.

In jedem Falle wird, egal was fuer ein Kommando im DOS ausgefuehrt wurde, der Returncode ueberprueft. Wurde ein Fehler festgestellt und dieser bereits angezeigt, wird das Kommando CMD"xx" abgebrochen ueber die Routine 'PREVIOUSLY DISPLAYED ERROR'. Wurde ein DOS-Fehler entdeckt, wird von BASIC die Routine 4409H aufgerufen um diese Fehlermeldung anzuzeigen und das Kommando CMD"xx" mit 'DOS ERROR' abgebrochen. Wurde kein Fehler entdeckt, faehrt BASIC mit der normalen Verarbeitung fort.

Jedes DOS-Kommando, das in dem Bereich zwischen 5200H und 6FFFH (und einigen anderen Bereichen) ausgefuehrt wird, kann aufgerufen werden ueber CMD"xx". Unter anderem gehoeren die Programme 'SUPERZAP' und 'DIRCHECK' dazu. Ebenso 'FORMAT' und die meisten Arten von 'COPY', es koennen jedoch keine ganzen Floppies kopiert werden, auch darf der Parameter 'UBB' nicht benutzt werden, da diese Arten das gesamte Memory benoetigen.

Es sei noch einmal darauf hingewiesen, dass das DOS-Kommando maximal 80 Zeichen haben kann, einschliesslich des Returns, das BASIC an den String anfuegt im DOS-Eingabepuffer.

In frueheren NEWDOS-Versionen wurde ebenfalls der Bereich von 4080H bis 41FFH gerettet. Dies wird nicht mehr beim NEWDOS/80 getan. Benutzer werden also davor gewarnt, ueber ein Anwenderprogramm etwas in diesem Bereich zu aendern, es sei denn, dass diese Aenderung bewusst ausgefuehrt wird, um die Funktion des BASIC oder eines BASIC-Programmes zu beeinflussen.

Der Befehl CMD"BASIC" sollte nie ausgefuehrt werden. Wenn man, aus welchem Grunde auch immer, das BASIC neu starten will, sollte man dies ueber CMD"S=BASIC" tun.

Einige Beispiele von Kommandos, die auf diese Art ausgefuehrt werden koennen:

1. CMD"DIR 1" - Directory listen
2. CMD"COPY XXX:0 XXX:1" - Kopieren
3. CMD"COPY 0 1 01/01/81 FMT" - Kopieren einer ganzen Diskette
4. CMD"SUPERZAP" - SUPERZAP ausfuehren
5. CMD"DO CHAINFIL" - Chainfile abarbeiten

7.15 CMD"F=POPX"

Wenn die ersten 2 Zeichen nach dem " ein F= sind wird folgendes ausgefuehrt :

Bei der Eingabe von CMD"F=POPS" werden alle Returnadressen und alle Kontrollbloecke fuer 'FOR-NEXT'-Loops geloescht. Das naechste Statement wird ausgefuehrt. Dies erlaubt es dem Benutzer, alle bisher nicht benutzten Ruecksprueng e und alle abgebrochenen 'FOR-NEXT'-Loops zu loeschen, da es manchmal, speziell bei rekursiver Programmierung, zu Fehlern kommen kann.

Bei der Eingabe von CMD"F=POPR" werden alle Returnadressen des momentanen 'GOSUB'-Levels inclusive der Kontrollbloecke fuer 'FOR-NEXT'-Loops dieses Levels geloescht. Dies hat den gleichen Effekt wie ein Return, jedoch mit dem Unterschied, dass das Programm nicht mit dem Statement fortfaehrt, das auf das 'GOSUB' folgt, sondern mit dem naechsten Statement nach CMD"F=POPR".

Bei der Eingabe bzw. dem Statement CMD"F=POPF" xx (xx ist ein Variablenname) wird der Kontrollblock des 'FOR-NEXT'-Loops mit der Variablen xx geloescht. Es hat den gleichen Effekt, als waere CMD"F=POPF" xx das letzte ausgefuehrte 'NEXT xx' Statement dieses Loops.

7.16 CMD"F=SASZ"

Mit diesem Kommando kann man die Groesse des Stringbereiches aendern, ohne dass die eventuell schon belegten Stringvariablen ihren Wert aendern. Das Kommando wird aufgerufen mit folgender Syntax:

CMD"F=SASZ",expl

Dieses Kommando kann benutzt werden, wenn sich beispielsweise waehrend eines Programmes herausstellt, dass der Stringbereich zu klein ist oder zu gross (wenn er sich mit den numerischen Variablen ueberlappt).

Beispiel:

7.17 CMD"F=ERASE" UND CMD"F=KEEP"

Mit diesem Kommandos kann man einzelne Variable loeschen

- CMD"F=ERASE",vn1,vn2,vn3....

  Die Variablen mit dem Namen vn1, vn2, vn3 ... werden geloescht. Sind diese Variable Bestandteil eines Feldes (vereinbart mit DIM..), kann man dies machen, weil z. B. das Feld nicht mehr benoetigt wird oder weil man das Feld neu definieren will mit einer anderen Groesse. Dieses Statement kann ueber mehrere Zeilen gehen wie beim folgenden KEEP.

- CMD"F=KEEP",vn1,vn2,vn3....

  Dieses Kommando bewirkt, dass alle Variablen geloescht werden ausser den im Kommando angegebenen Variablen vn1, vn2, vn3 etc. und den Variablentypen, die ueber DEFFN vereinbart wurden. Ist eine angegebene Variable ein Feldname, so wird das gesamte Feld von der Loeschung ausgenommen. Es muessen alle Variablen, die nicht geloescht werden sollen, in einem Statement angegeben werden. Aus diesem Grunde darf das Statement auch ueber mehrere Zeilen gehen, die fortzusetzende Zeile muss nur mit einem Komma abschliessen. Dazwischenstehende Bemerkungen (REM's) werden ignoriert.

  Beispiel:

```
CMD"F=KEEP",A$,B%,C,D@,      ' ERSTE ZEILE DES STATM.
E!,F,G$,                     ' ZWEITE ZEILE DES STATM.
REM - WIRD UEBERLESEN
H!,I                         ' DRITTE ZEILE VON KEEP
```

7.18 CMD"F",DELETE

Mit diesem Kommandos kann man Zeilen im Programmtext loeschen.

CMD"F",DELETE ln1-ln2

Mit diesem Kommando kann man alle Textzeilen im Programm loeschen, die groesser und gleich ln1 sind, bis zu der Zeile, die kleiner und gleich der Zeilennummer ln2 sind. Alle Variablen bleiben wie vorher, ausser denen, die mit DEFFN im geloeschten Textteil vereinbart wurden. Die Groesse des Stringbereichs wird nicht veraendert.

CMD"F",DELETE muss das letzte Statement in einer Zeile sein, es darf nicht als direktes Kommando eingegeben werden, darf nicht ein Teil eines DEFFN-Statements sein, darf nicht eine Subroutine oder Bestandteil eines FOR...NEXT-Loops sein und das Statement, das als naechstes ausgefuehrt werden soll, muss dem DELETE in der naechsten Zeile folgen.

Beispiel:

```
100 CMD"F",DELETE 10500-15000
110 X = 1 '            WEITER MIT DIESER ZEILE
```

7.19 CMD"F=SWAP"

Mit diesem Kommandos kann man den Wert zweier Variablen tauschen.

CMD"F=SWAP",vn1,vn2

Mit diesem Kommando wird der Wert der Variablen vn1 mit der Variablen vn2 vertauscht. Beide Variable muessen absolut gleich sein, d. h., man kann nur Integer mit Integer, String mit String, Gleitkomma mit Gleitkomma etc. vertauschen.

Beispiel:

```
CMD"F=SWAP",A$,B$
```

7.20 CMD"F=SS"

Mit diesem Kommandos kann man die Einzelschrittausfuehrung ein- oder ausschalten.

1. CMD"F=SS" - Einzelschritt einschalten
2. CMD"F=SS",ln1 - Einzelschritt ab Zeile ln1 einschalten
3. CMD"F=SS",N - Einzelschritt ausschalten

Mit dem NEWDOS/80 Version 2 kann der Programmierer jetzt auch Programme schrittweise ausfuehren. Mit Format 1 oder 2 wird die Einzelschrittausfuehrung eingeschaltet. Bei Format 2 beginnt sie nicht sofort, sondern erst bei Erreichen der Zeilennummer ln1. Mit jedem Schritt wird eine Zeile bearbeitet. Sobald die Zeile abgearbeitet ist, wird in der rechten oberen Ecke des Bildschirmes die naechste Zeilennummer im Format ânnnnn angezeigt als Zeichen, dass das System auf eine Eingabe wartet. Mit dem Druecken der ENTER-Taste wird diese Zeile bearbeitet, mit Druecken der BREAK-Taste wird das Programm wie sonst auch abgebrochen. Wenn nichts am Text veraendert wird, kann mit CONT das Programm fortgesetzt werden. In diesem Falle wird wieder ânnnnn angezeigt, ohne dass die Zeile bearbeitet wird. Dies geschieht erst, wenn man wieder ENTER drueckt. Nach dem BREAK kann aber auch z. B. der Einzelschritt-Modus abgeschaltet werden und das Programm ueber CONT fortgesetzt werden.

Einzelschrittausfuehrung oder die beabsichtigte Einzelschrittausfuehrung ab einer bestimmten Zeile bleiben wirksam, bis entweder das Kommando CMD"F=SS",N eingegeben wird zum Abbruch, oder wenn ein CMD"F=SS",ln1 eingegeben wird bis die

Zeilennummer lnl erreicht ist. Die Ausfuehrung der Kommandos RUN, LOAD, NEW etc. beeinflussen den Status der Einzelschritt-ausfuehrung.

7.21 CMD"O"

Mit diesem Kommandos kann man BASIC-Felder sortieren. Es hat 2 verschiedene Formate:

1. CMD"O",n,av1<,av2,...> direkter Sort

2. CMD"O",n,*iav1,av2<,av3,...> indirekter Sort

In dieser Beschreibung wird der Ausdruck REN oefter benutzt und bedeutet Relative Element Nummer. Die Elementnummer in einem BASIC-Feld ist, unabhaengig von der Dimension des Feldes, eine Integerzahl von 0 an aufwaerts. Wenn es sich um ein eindimensionales Feld handelt, so ist REN nur die Indizierung. Falls Sie nur eindimensionale Felder benutzen brauchen Sie in diesem Abschnitt nicht weiterzulesen. Bei mehrdimensionalen Feldern sollten Sie aber ueber die Reihenfolge des Sortiervorganges Bescheid wissen. Die Regel fuer das Errechnen von REN in mehrdimensionalen Feldern ist ziemlich komplex und laesst sich am besten an einem Beispiel darstellen:

```
DIM A(R1,R2,R3)
Y = A(X1,X2,X3)
```

Der Wert von REN errechnet sich w e folgt:

REN = X1+X2*(R1+1)+X3*(R1+1)*(R2+1).

Waere das Feld nur nur zweidimensional, wuerde sich REN errechnen aus X1+X2(R1+1). Wenn das Feld nur eindimensional waere, so haette REN natuerlich den Wert X1.

Werden im CMD"O" mehr als ein Feld angegeben (ausser iav1), so muss der Wert REN fuer alle Felder gleich sein.

Jedes der eingegebenen Felder hat einen Sortierlevel (ausser iav1), der Sortierlevel fuer das erste Feld ist der Hoechste, fuer das letzte eingegebene Feld der Niedrigste. Innerhalb dieses Levels werden die Stringvariable in aufsteigender Reihenfolge ihres numerischen Wertes des ASCII-Characters sortiert, und zwar in aufsteigender Reihenfolge. Numerische Variable werden von negativ nach positiv in aufsteigender Reihenfolge sortiert. Wird im CMD-Statement ein Minuszeichen vor die Variable gesetzt, so wird in absteigender Reihenfolge sortiert (z. B. -A$(0)). Hat ein String eine Laenge von 0, so wird ein negativer Wert angenommen.

Normalerweise werden bei Stringvariablen alle Character fuer den Sort benutzt. Wenn man allerdings eine Begrenzung in der Form (x,y) eingibt (z. B. A$(1)(5,4)), so faengt der Vergleich fuer den Sort beim x. Character an und ist nur y Character lang.

n ist die Anzahl der Elemente eines Feldes, die sortiert werden sollen. Elemente ueber oder unter n werden nicht mitsortiert. Wenn n=0 ist, werden entsprechend viele Elemente sortiert, wie das erste Feld Elemente minus der Startadresse hat.

Wenn die Anzahl der Elemente in einem Feld geringer ist als angegeben, also geringer als Startadresse plus n, so wird der Fehler FC gemeldet.

Es duerfen maximal 9 Felder angegeben werden. Alle Indizes aller Variablen muessen den gleichen REN-Wert haben.

In Format 1 ist der direkte Sort von 1 - 9 Feldern dargestellt. Die Felder werden sortiert in der beschriebenen Reihenfolge.

- av1 muss angegeben werden, av2 und mehr sind optional
- Die entstehende Reihenfolge in den n Elementen jedes Feldes ist gleich. Die Felder werden nicht nacheinander sortiert. Wenn z. B. in Feld av1 das x. Element an die m. Stelle verschoben wird, so werden in allen nachfolgenden Feldern das x. Element an die m. Stelle geschoben.
- Das Format 1 ist gleich wie das CMD"O" im TRSDOS wenn
  - nur eine Variable angegeben wird
  - dies eine Stringvariable ist
  - und n eine Integer-Variable ist

Das Format 2 ist ein indirekter Sortiervorgang. In diesem Sort werden ebenfalls nur n Elemente des Feldes iav1 sortiert, alle anderen Felder werden nicht veraendert. Der Grundgedanke bei diesem Sortieren war der, dass das zu sortierende Feld nicht veraendert wird.

- Format 2 wird unterschieden durch den Stern vor iav1 vom Format 1.
- iav1 muss ein Integer-Feld sein
- av2 muss angegeben werden, av3 und weitere koennen angegeben werden.
- Die n aufeinanderfolgenden Elemente von iav1 erhalten den REN-Wert der n sortierten Elemente des Feldes av2 (und av3 etc. falls vorhanden.
- Beim Sortieren wird das Feld iav1 veraendert, alle anderen Felder bleiben unangetastet.
- Nach Beendigung des Sortiervorganges sind die n Elemente des Feldes iav1 sortiert in der Reihenfolge, dass der Inhalt des Elements des Feldes iav1 den REN-Wert der sortierten Variablen av2 etc. ergibt. Bei mehrdimensionalen Feldern ist die Berechnung des Indexes daraus sehr komliziert, man sollte geuebten Programmierern ueberlassen.

Beispiele fuer einige Sorts:

```
10 DIM NM$(200),AM!(100),LN$(100),IX%(100),ZC!(50),1$(50)
30 X = 150
40 CMD"O",x,NM$(0)
60 CMD"O",X,-NM$(25)
70 CMD"O",0,-AM!(1),LN$(1)(5,3)
80 CMD"O",100,*IX%(0),ZC!(1),L$(1)
```

- In Zeile 40 werden die ersten 150 Elemente des Feldes NM$ (von Element NM$(0) bis NM$(149)) in aufsteigender Reihenfolge sortiert. Ist eines der Stringelemente 0 (""), so kommt dies an die erste Stelle. Die restlichen 51 Elemente

des Feldes NM$ (Element NM$(150) bis NM$(200)) bleiben unveraendert und nicht sortiert.

- In der Zeile 60 werden die Elemente NM$(25) bis NM$(174) in absteigender Reihenfolge sortiert, wenn ein Nullstring mit enthalten ist, so wird dieser am Ende des sortierten Feldes erscheinen. Die ersten 25 und die letzten 26 Elemente des Feldes werden nicht mitsortiert.

- In Zeile 70 werden die Felder AM! und LN$ in absteigender Reihenfolge des Feldes AM! sortiert. Haben mehrere Elemente von AM! den gleichen Wert, so wird nach aufsteigender Reihenfolge der Stringvariablen LN$ sortiert, wobei fuer den Vergleich nur der 5., der 6. und der 7. Character herangezogen wird. Falls ein Element in LN$ weniger als 5 Character hat, so wird ein Nullstring fuer die Sortierung angenommen. AM!(0) und LN$(0) werden nicht in den Sortiervorgang mit einbezogen. Da die Anzahl der zu sortierenden Elemente mit 0 angegeben war, werden 100 Elemente sortiert (die Groesse von AM! minus 1).

- Die Zeile 80 enthaelt einen indirekten Sort. In diesem Sort werden die ersten 100 Elemente des Feldes IX% initialisiert mit REN-Werten von 1 bis 100, wobei IX%(0)=1 ist und IX%(99)=100. Mit diesen Werten wird das eigentliche Sortierfeld ZC! und L$ indiziert. Diese Sortierfelder werden, ohne dass sie in ihrer Reihenfolge veraendert werden, in aufsteigender Reihenfolge sortiert. Haben 2 oder mehrere Elemente des Feldes ZC! den gleichen Wert, so entscheiden die entsprechenden Elemente des Feldes L$ fuer die aufsteigende Reihenfolge. ZC! und L$ bleiben, wie schon erwaehnt, in ihrer Reihenfolge unveraendert, nur der entsprechende REN-Wert wird veraendert. Wenn der Sortiervorgang beendet ist, kann der Wert von IX%(0) benutzt werden als Index fuer das kleinste Element von ZC! und L$, in IX%(99) steht der Index fuer das groesste Element von ZC! und L$. Die Elemente IX%(100), ZC!(0) und L$(0) wurden ignoriert bei diesem Sort.

7.22 RENEW

Mit diesem Kommando kann ein mit NEW geloeschtes Programm wieder aktiviert werden. Alles was RENEW im Prinzip macht, ist das erste Byte des Text-Bereiches auf ungleich 0 zu setzen, die Vorwaertspointer wieder herstellen und CLEAR aufzurufen. Der alte Text im Textbereich muesste danach wieder verfuegbar sein fuer RUN oder EDIT. Wurde aber nach dem NEW auch nur eine Zeile eines neuen Programmes eingegeben, kann der alte Text nicht mehr aktiviert werden. Ferner kann ein grosser Unsinn entstehen, wenn der Text-Bereich vorher ueberhaupt keinen Text enthalten hat.

8.0 BASIC TEIL 2 - EIN/AUSGABE

8.1 ALLGEMEINES

Dieses Kapitel behandelt die zusaetzlichen Elemente des Disk-BASIC's von NEWDOS/80 Version 2 gegenueber der Version 1, den aelteren NEWDOS-Versionen (NEWDOS/21-/35) und der Orginal-TRSDOS Version 2.3 . Die Aussagen von Abschnitt 7.1 gelten auch hier.

Dieser Teil mit der Ein-/Ausgabe auf die Diskette ist sehr komplex und auch schwierig zu verstehen. Es handelt sich hier aber nur um die Spezifikation, im Anhang A steht die Beschreibung dieses Teiles, den ein Kunde von APPARAT geschrieben hat, ferner stehen jetzt in der Version 2 im Anhang B noch 18 Beispiele zur Ein- und Ausgabe zur Verfuegung. Das Programm SAMPLE01/BAS ist dafuer nicht mehr auf der Diskette.

Sollte es zu widerspruechlichen Aussagen zwischen den Anhaengen A/B und diesem Kapitel kommen, so hat in jedem Falle dieses Kapitel den Vorrang.

In diesem Kapitel sind etliche Abkuerzungen benutzt, die in Kapitel 10 erklaert sind. Bitte schlagen Sie dort nach.

8.2 DISK-FILES

Zu den bisher existierenden Disk-BASIC-Filetypen, den sequentiellen mit PRINT/INPUT und den Randomfiles mit GET und PUT (aber nicht mit variabler Recordlaenge), wurden zwei neue Filetypen hinzugefuegt: die sog. 'Marked Item'- mit 3 Untergruppen ("MI", "MU" und "MF") und die 'Fixed Item'-Files.

PRINT/INPUT-Files (sequentiell) und Field-Item-Files (Random-Files) sind im TRSDOS-Manual fuer Modell I in Kapitel 7 und im TRSDOS-Manual Teil II: fuer Model! III ausfuehrlich beschrieben. Der Anwender sollte sich diese Kapitel vorher gut durchlesen und gegebenenfalls auch einige Beispiele durcharbeiten, bevor er hier in Kapitel 8 fortfaehrt.

- Ein Field-Item-File (bekannt im TRSDOS als Random-File) besteht aus Records, die alle gleich lang sind. Die Recordlaenge kann zwischen 1 und 256 Bytes betragen. Wenn die Recordlaenge ungleich 256 Bytes ist, so muss beim Aufruf von BASIC die Anzahl der Filebereiche angegeben werden mit dem Zusatz 'V'.

  ```
  BASIC,3V
  ```

  Damit wird das BASIC mit 3 Filebereichen initialisiert. Jeder Filebereich besteht aus 2 Buffern. Ein Buffer wird fuer das FIELD-Statement benoetigt, der 2. Buffer wird als Sektorbuffer benutzt. Denken Sie bitte daran, dass der Zusatz V nur bei einer anderen Recordlaenge als 256 bei Field-Item-Files angegeben wird. Andernfalls ist eine zusatzliche Belegung fuer einen zweiten Buffers sinnlos. Das OPEN-Statement fuer einen File mit unterschiedlicher Recordlaenge sieht folgendermassen aus:

  ```
  OPEN,"R",fan,fname,lrecl
  ```

  wobei lrecl einen Wert zwischen 1 und 256 hat.

8.3 UNTERSCHIED ZWISCHEN DEN FILETYPEN

- PRINT/INPUT-Files koennen nur sequentiell benutzt werden; Field-Item, Fixed-Item und Marked-Item-Files koennen sowohl sequentiell als auch random benutzt werden.

- PRINT/INPUT-Files werden als ASCII-Character abgespeichert. Alle numerischen Werte werden auch in ASCII-Character umgewandelt. In Field-Item, Fixed-Item und Marked-Item-Files wird direkt der binaere Wert abgespeichert, was Umwandlungszeit und Platz spart.

- PRINT/INPUT-Files werden mit dem PRINT-Statement auf die Floppy geschrieben. Dies ist aber sehr unpraktisch, da 2 Strings mit der Sequenz ;"," abgeschlossen werden muessen zur Trennung. Field-Item, Fixed-Item und Marked-Item-Files werden mit dem PUT-Statement geschrieben, die Trennung zwischen den einzelnen Strings geschieht bei den Fixed-Item-Files durch das FIELD-Statement, bei Fixed-Item-Files durch die Laengenangabe im IGEL und bei Marked-Item-Files durch die interne Markierung.

- PRINT/INPUT-Files werden mit dem INPUT-Statement von der Floppy gelesen. Field-Item, Fixed-Item und Marked-Item-Files mit einem GET-Statement.

- In Field-Item-Files muessen die Daten vor dem Schreiben mit einem LSET oder RSET in den Buffer geschrieben werden, numerische Daten mit der MKD$-, MKI$- oder MKS$-Funktion. Diese explizite Umwandlung ist bei PRINT/INPUT-Files, bei Fixed- und Marked-Item-Files nicht noetig.

- In Field-Item-Files muessen die Daten nach dem Lesen vor der Benutzung von ASCII in binaere Werte umgewandelt werden mit der CVI-, CVS- oder CVD-Funktion. Diese explizite Umwandlung ist bei PRINT/INPUT-Files, bei Fixed- und Marked-Item-Files nicht noetig.

- PRINT/INPUT-Files duerfen jede Recordlaenge haben, Field-Item-Files haben ein Maximum von 256 Bytes, Fixed- und Marked-Item Files haben eine maximale Laenge von 4096 Bytes.

- PRINT/INPUT-Files werden ohne Aenderung in den File uebertragen, es werden nur fuehrende Leerzeichen abgeschnitten beim einlesen aus dem File. Strings in Field-Item-Files werden entweder rechts- oder linksbuendig in den Record gesetzt, jeweils entsprechend dem LSET oder RSET-Kommando. Strings in Marked-Item-Files werden rechtsbuendig geschrieben mit fuehrenden Leerzeichen. Strings im Marked-Item-File werden exakt uebertragen, nur wenn die maximale Laenge fuer einen Bestandteil eines Records ueberschritten wird, wird dieses Bestandteil abgeschnitten. Der Marked-Item-File ist der einzige, in dem Strings komplett unveraendert uebernommen werden.

Zuvor muessen aber die 3 verschiedenen Faelle der Anwendung von GET und PUT etwas naeher erlaeutert werden, da 2 davon neu sind.

8.4 GET UND PUT

GET und PUT arbeiten in 2 verschiedenen Phasen in folgender Reihenfolge:

1. Die Positionierungsphase. In dieser Phase wird die Position innerhalb des Files entsprechend dem Positionierungsparameter eingenommen. Dies ist der 2. Parameter bei jedem GET- und PUT-Statement.

2. Die Datenuebertragungsphase. In dieser Phase werden die Daten vom Speicher in den File uebertragen (beim PUT) bzw. vom File in den Speicher (beim GET).

Zuvor muessen aber die 3 verschiedenen Faelle der Anwendung von GET und PUT etwas naeher erlaeutert werden, da 2 davon neu sind.

8.4.1 FP

File Positionierung - fuer jedes GET und PUT ist die Ein- und Ausgabe intern entsprechend der FP-Angabe positioniert. FP kann folgende Form haben:

8.4.1.1 Null

Wenn die interne Record-Adresse (REMRA) gueltig ist und der File segmentiert ist wird er auf den naechsten Record positioniert. Andernfalls bleibt er auf dem gleichen Record stehen wie bei FP=*.

Beispiel: PUT 1,,1000

8.4.1.2 *

Die interne Position des Files bleibt auf dem gleichen Wert. Es wird also nicht auf den naechstän Record wie bei Null weitergeschaltet.

Beispiel: GET 1,*,1000

8.4.1.3

Der File wird positioniert auf die interne Record-Adresse (REMRA). Der gleiche Record kann nochmals behandelt werden. Ein Fehler wird gemeldet, wenn REMRA keinen gueltigen Wert hat.

Beispiel: PUT 1,#,1000

8.4.1.4 $

Der File wird positioniert auf die interne Byte-Adresse (REMBA). Der gleiche Record kann nochmals ueber die gleiche Positionierung des letzten GET- und PUT-Befehls (Null, *, RN oder !RBA) angesprochen werden.

Beispiel: GET 1,$,1000

8.4.1.5 %

Siehe "Pseudo Field - Funktion" auf Seite 144.

8.4.1.6 &

Siehe "PUT fan,&" auf Seite 143 - Befehl PUT,fan,&

8.4.1.7 &&

Siehe "PUT fan,&" auf Seite 143 - Befehl PUT,fan,&&

8.4.1.8 !RBA

RBA (Relative Byte Address) - ist eine relative Byteadresse innerhalb eines Files. Der Wert kann zwischen 0 und 16.777.215 liegen. GET und PUT lesen bzw. schreiben ab der angegebenen Position. Bei segmentierten Files ist !RBA die Position des Records.

ACHTUNG: Falsche Anwendung bzw. falsche Werte koennen einen File zerstoeren !

ACHTUNG: Der Ausdruck fp darf keine Funktion wie z. B. LOC enthalten, in der auf den Filebereich bezogen wird.

8.4.1.9 !%

Prinzipiell gleiche Funktion wie !RBA, jedoch mit dem Unterschied, dass !% den EOF-Wert (End-Of-File) als RBA benutzt.

Beispiel: GET 1,!%,1000

8.4.1.10 !$RBA

Position relativ zur momentanen Fileposition. Siehe "GET" auf Seite 139

Beispiel: GET 1,!$1354

8.4.1.11 !$%

Gleiche Funktion wie !$RBA, nur dass der EOF-Wert als RBA benutzt wird.

Beispiel: GET 1,!$%

8.4.1.12 !#RBA

RBA wird auf den neuen EOF-Wert gesetzt. Siehe "PUT mit !#RBA" auf Seite 144.

Beispiel: PUT 1,!#1354

8.4.1.13 RN

RN (Record Nummer) muss ein ganzzahliger (Integer) Ausdruck sein im Bereich von 1 (!) bis 32767 und gibt die Nummer des zu behandelnden Records an. Der File wird positioniert auf den ersten Wert des Records RN, der Record musste eroeffnet werden mit 'M' = I, R oder D, wenn 'FT' angegeben wurde, muss 'FT' = FF oder MF sein.

Beispiel: GET 1,30

8.4.2 IGEL

Item Group Expression List (kurz: Variablenliste) Eine Reihe von beliebigen Variablen (bei PUT auch Ausdruecke), die durch Komma getrennt sein muessen. Diese Variablen ent- entsprechen dem File-Inhalt, beginnend von der momentanen File-Position. Der 'IGEL' muss mit einem Semikolon (;) abschliessen. Kommas und Semikolon innerhalb von Hochkomma ("....,...;...") werden nicht beruecksichtigt, ebenso Kommentarzeilen mit REM (die Variablenliste kann ueber mehrere Zeilen gehen !).

Beispiel:

```
1000  A$(X(Y)),B(2+X),C!,(25)E$  ' ENDE DER 1. ZEILE
1100  ,"MUELLER,HUBER;", ' NOCHMALS: INNERHALB "..." ZUGEL.
1200  REM - AUCH KOMMENTARZEILEN WERDEN UEBERLESEN
1300  0.935,AA$+"ZZZZ "+"DDDDD "+CC$
1400  ;    ' <--- DIESES ; IST DAS ENDE DES IGELS
```

Im Fehlerfalle wird die Zeilennummer des GET/PUT-Statements angezeigt, nicht die Zeile, wo der Fehler tatsaechlich war!

8.4.3 IGEL - Ausdruck

Ein Ausdruck ist ein Bestandteil der 'IGEL Group Expression List'. In einem PUT-Statement wird der Wert dieses Ausdrucks dem entsprechenden Teil des Files zugewiesen. In einem GET-Statement besteht ein solcher Ausdruck nur aus einer Variablen, die den Wert des entsprechenden Teiles des Files erhaelt. Dieser IGEL- Ausdruck kann eine der folgenden Formen haben:

1. Ausdruck (Expression)
2. (len) exp (len = Laengenangabe)
3. (len) $ (nur Fixed Item Files)
4. (len) #
5. Null-exp

wobei

8.4.3.1 exp

expression (Ausdruck) - normalerweise der Name einer BASIC-Variablen, bei einem PUT auf einen 'MI'-File ist jedoch alles zugelassen, was rechts vom Gleichheitszeichen bei einem 'LET'-Statement stehen kann. Bei einem 'FI'-File muss der Variablenname seine Typkennung enthalten, also mit einem $, %, ! oder # abschliessen.

Es sollte beispielsweise geschrieben werden

```
A$,B%(X,Y),C!,D#;
```

anstelle von

```
A,B(X,Y),C,D;
```

Diese Regel gilt nicht fuer die Indizierung (im Beispiel (X,Y)).

8.4.3.2 (len) exp

Laenge der Variablen, kann zwischen 0 und 255 liegen und darf nur bei Stringvariablen auftreten.

1. Bei 'MI'-Files stellt 'len' die maximale Laenge der zu schreibenden (bei PUT) bzw. der zu lesenden (bei GET) Zeichen einer String- Variablen dar. Ist diese String-Variable kuerzer, wird nur die tatsaechliche Anzahl der Zeichen uebertragen. 'len' kann auch weggelassen werden.

2. bei 'FI'-Files stellt 'len' die tatsaechliche Laenge der Variablen und des entsprechenden Teils des Files dar. 'len' muss angegeben werden bei String-Variablen.

   Beispiel:

   ```
   (30)LN$,(20)FN$,AN%,DP$,(2)CD$(X)
   ```

8.4.3.3 (len)$

'len' gibt die Anzahl der zu ignorierenden Bytes an. Bei einem GET-Statement werden 'len' Bytes ueberlesen, bei einem PUT auf einen bestehenden Record werden diese Bytes nicht veraendert. Bei einem neuen Record wird wie bei (len)# verfahren. (len)$ darf nur bei 'FI'-Files benutzt werden.

Beispiel:

```
(10)$,AN%,(10)ST$,(17)$,DP$;
```

8.4.3.4 (len)#

Bei 'FI'-Files wird bei GET-Operationen genauso wie bei (len)$ verfahren, bei PUT-Operationen werden entsprechend viele Bytes als 0 in den Record geschrieben. Bei 'MI'-Files wird genau wie bei 'FI'-Files verfahren.

Beispiel:

```
(10)#,AN%,(10)ST$,(17)#,DP$;
```

8.4.3.5 null exp

Ein Null-Ausdruck kann nur beim GET von Marked Item Files benutzt werden. Er bewirkt, dass die Bestandteile des Files ignoriert werden. Wenn man beispielsweise den ersten, zweiten und vierten Record lesen Will, gibt man ein:

```
GET 1,,,,,X!,,A$;
```

Wenn beim Abarbeiten eines IGEL's ein Fehler bei einer Variablen auftritt, so wird die Nummer der Variablen in der Fehlermeldung vor den Fehler gesetzt. Wenn beispielsweise bei der 4. Variablen der Fehler auftritt, so erscheint vor der Fehlermeldung eine 4.

8.5 FIXED ITEM FILES

Sie haben folgende Eigenschaften:

1. Sie enthalten entweder keine, eines oder mehrere Bestandteile.
2. Der Typ und die Laenge jedes Bestandteils wird durch den Ausdruck bzw. die Ausdrucksliste des GET- und PUT-Statements festgelegt, nicht jedoch im File selbst. Dies ist der grundsaetzliche Unterschied zu den Marked-Item-Files.
3. Ein File kann unterteilt werden in Records von gleicher Laenge.
4. Die maximale Recordlaenge betraegt 4095 Bytes.
5. Die Anzahl und die Art eines Bestandteils eines Records haengt alleine von der Recordlaenge und dem 'IGEL' ab, der bei GET und PUT benutzt wird.
6. Auf einen Fixed-Item-File kann in BASIC nach einem OPEN mit FT = "FI" oder "FF" zugegriffen werden.
7. Der Inhalt von Bestandteilen eines Fixed-Item-Files wird in BASIC mit einem GET-Statement in eine Variable gelesen.
8. Bestandteile eines Fixed-Item-Files werden in BASIC erzeugt bzw. geaendert mit einem PUT-Statement.
9. Mit dem BASIC-Statement CLOSE wird die Verbindung zwischen BASIC-Programm und dem File unterbrochen.
10. Zwischen den Records eines Files bzw. den Bestandteilen eines Records wird auf der Floppy kein Platz freigelassen.

8.6 MARKED ITEM FILES

Sie haben folgende Merkmale:

1. Sie enthalten entweder keine oder mehrere Bestandteile.
2. Das Bestandteil eines Marked-Item-Files faengt immer mit einem oder mehreren Steuerzeichen an, dem entweder keine (Null- String), ein oder mehrere Datenbytes folgen.
3. Das Format der Bestandteile eines Marked-Item-Files haengt vom 1. Steuerzeichen dieses Bestandteils ab:

80H - 0FFH	es folgen 0-127 Bytes binaerer String
70H	SOR (Start Of Record). In einem Marked-Item-File mit Variabler Recordlaenge ('MU'-File) faengt jeder Record mit 70H an.
00	Fuellzeichen bei fester Recordlaenge oder Fuellzeichen beim Update variabler Recordlaenge.

71H — das naechste Byte enthaelt die Laenge (0- 255) des folgenden Strings.

72H — die naechsten 2 Bytes enthalten das Zweierkomplement einer Integer-Zahl (normales Level II BASIC Format).

73H — die naechsten 4 Bytes enthalten eine Realzahl (Floatingpoint-Zahl, einfache Genauigkeit, BASIC Format).

74H — die naechsten 8 Bytes enthalten eine Realzahl (Floatingpointzahl, doppelte Genauigkeit, BASIC Format).

4. Ein File kann unterteilt werden in Records, alle Records haben entweder die gleiche oder unterschiedliche Laenge.

5. Ein Record hat eine maximale Laenge von 4095 Bytes, einschliesslich aller Steuer- und Datenbytes.

6. Wenn die Records eines Files variable Laenge haben, fuegt das BASIC beim Start eines Records automatisch ein 'SOR' ein.

7. Records mit variabler Laenge koennen aus beliebig vielen und unterschiedlich vielen Bestandteilen bestehen. Bei Records mit fester Laenge muss der Benutzer selbst auf die Einhaltung der Laenge achten.

8. Die relative Position eines Bestandteiles eines Records kann von Record zu Record unterschiedlich sein aufgrund der verschiedenen Laenge verschiedener Typen. Grundsaetzlich ist es vorteilhaft, immer den gleichen Datentyp zu benutzen.

9. Auf einen Marked-Item-File kann in BASIC nach einem OPEN mit FT = "MI", "MU" oder "MF" zugegeriffen werden.

10. Der Inhalt von Bestandteilen eines Marked-Item-Files wird in BASIC mit einem GET-Statement in eine Variable gelesen.

11. Bestandteile eines Marked-Item-Files werden in BASIC erzeugt bzw. geaendert mit einem PUT-Statement.

12. Mit dem BASIC-Statement CLOSE wird die Verbindung zwischen BASIC-Programm und dem File unterbrochen.

13. Zwischen den Records eines Files bzw. den Bestandteilen eines Records wird auf der Floppy kein Platz freigelassen. In jedem Falle werden 'SOR' und Fuellzeichen nach Bedarf eingefuegt.

8.7 OPEN

Das OPEN-Statement wurde auf folgende Formatarten modifiziert:

1. OPEN m,fan,FNAME
2. OPEN m,fan,FNAME,len
3. OPEN m,fan,FNAME,ft
4. OPEN m,fan,FNAME,ft,len

8.7.1 FAN / Filespec

Siehe Anhang fuer die Erlaeuterung. Hier aber noch einige Beispiele fuer das OPEN:

```
OPEN "I",1,"XXX/DAT:1"
OPEN "R",2,"XXX/DAT",128
OPEN "O",1,"XXX/DAT:0","MU"
OPEN "D",3,"XXX/DAT","MF",71
```

8.7.2 Anwendung

Das 1. Format wird benutzt bei PRINT/INPUT-Files und Field Item Files. Das 2. Format benutzt man bei Field Item Files. Das 3. Format kommt bei FI, MI und MU-Files zur Anwendung, das 4. Format bei MU, MF und FF-Files.

8.7.3 m

m gibt den Operationsmodus an, mit welchem dann auf ein File zugegriffen werden kann. m wird als String angegeben (".") und kann folgende Werte haben:

I — Der File wird eroeffnet und ist nur fuer INPUT zugelassen. Auf den File kann mit INPUT (wenn FT nicht angegeben wurde) oder mit GET (wenn FT angegeben wurde) zugegriffen werden. Mit dem ersten GET oder INPUT erhaelt man automatisch den ersten Satz.

O — Falls der angegebene File mit dem namen FNAME noch nicht existiert, wird er erzeugt. Auf diesen File kann nur geschrieben werden mit PRINT (wenn FT nicht angegeben wurde) oder mit PUT (wenn FT angegeben wurde). EOF wird auf 0 gesetzt, der erste geschriebene Satz steht automatisch an erster Stelle.

E — Gleiche Funktion wie "O", nur dass des EOF-Byte nicht veraendert wird. Ein Satz, der auf einen bereits existierenden sequentiellen File geschrieben wird, wird automatisch angehaengt.

R — Falls der angegebene File mit dem Namen FNAME noch nicht existiert, wird er erzeugt. Auf diesen File kann nur mit GET und/oder PUT zugegriffen werden. EOF wird nicht veraendert, der File ist auf dem Anfang positioniert. Wenn ein PUT hinter das momentane EOF schreibt, wird der File automatisch erweitert.

D — Gleiche Bedingungen wie bei "R", nur dass der File bereits existieren muss und das PUT nach dem EOF als Fehler zurueckgewiesen wird. Der File kann also nicht erweitert werden.

8.7.4 FT

Folgendes geschieht, wenn FT angegeben wird:

1. Wenn ein GET benutzt wird fuer die Datenuebertragung von einem File in eine BASIC-Variable, muss entweder ein IGEL oder eine IGEL-Statementnummer (IGELSN) angegeben werden.

2. Wenn ein PUT benutzt wird fuer die Datenuebertragung von einer BASIC-Variablen in einen File, muss entweder ein IGEL oder eine IGEL-Statementnummer (IGELSN) angegeben werden.

3. Das BASIC-Statement FIELD darf nicht benutzt werden.

4. Das BASIC-Programm darf weder den Ein/Ausgabebuffer noch die Werte LRECL, NEXT oder EOF im FCB veraendern.

8.7.5 Mit FT

ft gibt den Filetyp an und wird als String eingegeben ("..."). ft kann folgende Werte haben:

FI Fixed Item File, der nicht in Records unterteilt ist. 'len' darf nicht angegeben werden.

FF Fixed Item File, der in Records gleicher Laenge unterteilt ist. 'len' muss angegeben werden.

MI Marked Item File, dieser Filetyp ist nicht in Records unterteilt. 'len' darf nicht angegeben werden. Ein "MI"-File kann nicht veraendert werden.

MU Marked Item Files, unterteilt in Records mit variabler Laenge. Der naechste Record wird jeweils anhand des SOR-Bytes gefunden. 'len' kann angegeben werden, es ist dann aber nur die maximale Laenge eines Records und darf dann nicht ueberschritten werden. Ein "MU"-File darf veraendert werden, der veraenderte Record darf aber nicht laenger sein als der alte Record. Ist der neue Record kuerzer als der alte, werden entsprechende Fuellzeichen (00H) eingefuegt.

MF Marked Item Files, unterteilt in Records mit gleicher Laenge. 'len' muss angegeben werden..

8.7.5.1 Ohne FT

Wenn der Eroeffnungsmodus eines Files "R" oder "D" ist gilt folgendes:

1. Es handelt sich um einen Field Item File (wahlfreier Zugriff) wie im TRSDOS, falls nicht anders angegeben.

2. Das FIELD-Statement muss benutzt werden. FIELD kann 256 Bytes lange Records bearbeiten, die Laenge der Strings im FIELD ist jedoch begrenzt auf 255 Bytes.

3. GET und PUT Statements duerfen keinen IGEL oder IGELSN enthalten.

8.7.5.2 len

Ein Ausdruck, der eine Integerzahl zwischen 1 und 4095 ergibt. Wenn FT = FF oder MF ist, entspricht len der Standardlaenge fuer alle Records des Files. Wenn FT = MU ist, dann entspricht len der maximalen Laenge der Records. Im Moment enthaelt der FPDE nicht die korrekte Recordlaenge (LRECL). Bei GET's und PUT's wird len ueberprueft. Bei FT = MF muessen fuer die Recordlaenge die Markierungsbytes beruecksichtigt werden. Dies sind

- 1 Byte fuer jedes Bestandteil des Records (Markierungsbyte)
- 1 Byte fuer jeden String von mehr als 127 Zeichen Laenge

Bei FT = MU muss in jedem Record Platz fuer das SOR-Byte (Start of Record).

Die Anzahl der Datenbytes in einem Marked Item File entspricht der Anzahl, die auch das BASIC benoetigt fuer einen String oder einen numerischen Wert. Dies sind:

Strings	aktuelle Stringlaenge, die ggf. entsprechend der Laengenangabe im IGEL gekuerzt werden kann.
Integer	2 Bytes
Gleitkomma	4 Bytes
Gleitkomma mit doppelter Genauigkeit	8 Bytes

Die Anzahl der Datenbytes in einem Fixed Item File entspricht der Anzahl, die im IGEL angegeben ist:

- Strings, sowie (len)$ und (len)#: die Laengenangabe aus dem IGEL
- Integer: 2 Bytes
- Gleitkomma: 4 Bytes
- Gleitkomma mit doppelter Genauigkeit: 8 Bytes

8.8 GET

Das GET-Statement im Disk-BASIC wurde so modifiziert, dass es folgende Formate zulaesst:

1. GET fan (FP ist Null)
2. GET fan,FP
3. GET fan,FP,IGELSN
4. GET fan,FP,,IGEL

wobei

8.8.1 fan und IGELSN

Sie sind erlaeutert in der Beschreibung, FP wird definiert in "FP" auf Seite 131 und der IGEL in "IGEL" auf Seite 133 Beispiele fuer die 4 verschiedenen Formate:

```
GET 1
GET 1,30
GET 1,!X,1000
GET 1,,,X%,Y!,Z#,(20)A$;
```

8.8.2 Ende

Wenn das GET-Statement abgeschlossen ist, ist der Filebereich positioniert auf

- bei Marked Item Files auf das naechste Bestandteil des Files
- bei Fixed Item Files auf das naechste Byte des Files
- bei Field Item Files auf den naechsten 256 Bytes Block

8.8.3 EOR oder EOF

- Bei Field Item Files wird der Filebuffer auf 0 gesetzt, so dass von da an alle Daten 0 sind. Eine Fehlermeldung erfolgt nicht.
- Bei Marked und Fixed Item Files erfolgt eine Fehlermeldung.

8.8.4 Fehlerfall

Wenn ein Fehler bei einem GET auftritt, wird der Filebereich an der gleichen Stelle wie vor dem GET positioniert. Die Daten der Variablen im IGEL oder FIELD-Statement sind ungueltig. Nach der Korrektur des Fehlers kann das gleiche Statement nochmals ausgefuehrt werden.

8.8.5 GET mit IGEL oder IGELSN

Wenn im GET ein IGEL oder IGELSN angegeben ist, werden die Bestandteile eines Records in die Variablen uebertragen, die im IGEL angegeben sind. Bei Marked Item Files:

1. Wenn ein IGEL-Ausdruck Null ist (fehlt), wird das sprechende Bestandteil im Record ueberlesen.
2. Ein Vorsatz vor dem Variablennamen bei Stringvariablen wird benutzt, um den einzulesenden String in der Laenge zu begrenzen. Wenn der im Record stehende String kuerzer ist als die angegebene Laenge, so wird die Laenge der Variablen auf den tatsaechlichen Wert gesetzt.
3. Stehen im Record Fuellzeichen, werden diese ignoriert.
4. Stimmen die Variablentypen (String/numerisch) nicht ueberein, wird der Fehler TM (TYPE MISMATCH) gemeldet.
5. Bei Fixed Item Files kann auf ein GET fuer das erste (oder die ersten) Bestandteil eines Records ein PUT auf die darauffolgenden Bestandteile erfolgen.
6. Bei einem Record-segmentierten File erfolgt eine Overflow-Fehlermeldung, wenn ein undefiniertes Bestandteil gefunden wird.
7. Ausser bei einem Vorsatz vor dem Variablennamen, der die uebertragende Laenge der Variablen begrenzt, wird der Inhalt der Variablen so uebertragen, wie er im Record steht. Fuehrende Leerzeichen werden nicht unterdrueckt.
8. Wenn noch eine Variable im IGEL angefordert wird, das naechste Zeichen im Record jedoch ein SOR-Byte ist, erfolgt die Fehlermeldung RECORD OVERFLOW.

Bei Fixed Item File Operationen:

1. Bei jeder Stringvariablen werden die Anzahl von Zeichen uebertragen, die als Vorsatz vor dem Variablennamen stehen.

2. Bei einem Record-segmentierten File erfogt eine Overflow-Fehlermeldung, wenn ein undefiniertes Bestandteil gefunden wird.

3. GET's und PUT's koennen aufeinander folgen. Folgendes ist dabei zu beachten:

   a. Der Benutzer muss selbst auf die richtige Positionierung innerhalb des Files achten.

   b. Die Recordgrenzen werden bei Record-segmentierten Files auf Einhaltung ueberwacht.

Bei Marked Item File Operationen:

1. Der Input von Bestandteilen eines Records kann sich ueber 2 oder mehr GET's erstrecken.

8.8.6 GET mit !$..

Die GET-Statements in der Form

```
GET FAN,!$RBA
GET FAN,!$%
```

erlauben es dem Programmierer, den File fuer den naechsten Zugriff mit GET, PUT, INPUT oder PRINT zu Positionieren, ohne dass eine Datenuebertragung stattfindet. Dabei bedeutet !$%, dass der aktuelle Wert von EOF als naechste Adresse benutzt wird. Statements in dieser Art bewirken, dass die Werte REMRA und REMBA ungueltig werden.

8.9 PUT

Das PUT-Statement im Disk-BASIC wurde so modifiziert, dass es folgende Formate zulaesst:

```
1.  PUT fan            (FP ist Null)
2.  PUT fan,FP
3.  PUT fan,FP,IGELSN
4.  PUT fan,FP,,IGEL
```

wobei

fan und IGELSN sind erlaeutert in der Beschreibung, FP wird definiert in "FP" auf Seite 131 und der IGEL in "IGEL" auf Seite 133). Noch einige Beispiele fuer PUT:

```
PUT 2
PUT 1,X
PUT 3,,1000
PUT 1,RN!,,(20)A$,B%,C!,D#;
```

8.9.1 Ende

Wenn das PUT-Statement abgeschlossen ist, ist der Filebereich so positioniert wie beim GET.

8.9.2 Fehler

Wenn ein Fehler bei einem PUT auftritt, wird der Filebereich an der gleichen Stelle wie vor dem PUT positioniert. Die Daten im File sind ungueltig, es kann beim naechsten GET auf diese Daten ein Fehler auftreten. Aus diesem Grunde sollte nach einem Fehler ein neues PUT auf die gleiche Stelle erfolgen. Um diese Fehler auf ein Minimum zu beschraenken, sollte nach einem OPEN mit dem Mode "R" oder "D" ein IGEL einmal ausgefuehrt werden ohne Datentransfer, damit andere Fehler (ausser I/O) ausgeschlossen werden koennen.

8.9.3 PUT mit IGEL oder IGELSN

Wenn im PUT ein IGEL oder IGELSN angegeben ist, werden die Bestandteile des IGEL's an die entsprechende Stelle im File geschrieben.

Bei Marked Item Files gilt:

1. SOR und Fuellzeichen werden automatisch vom System an der Stelle eingefuegt, wo sie notwendig sind.

2. Das Bestandteil eines IGEL darf alles enthalten, was auch rechts vom Gleichheitszeichen in einem LET-Statement erlaubt ist. Ausnahme: es darf keine Funktion sein, die sich auf einen Filebereich bezieht.

3. Bei Stringvariablen werden die Strings unveraendert in den File geschrieben, ausser wenn sie durch einen Vorsatz vor dem Filebestandteil begrenzt werden.

4. Numerische Zeichen oder Ausdruecke werden in der BASIC-internen Schreibweise in den File geschrieben. Es erfolgt keine Konvertierung.

5. Bei Files mit fester oder variabler Recordlaenge wird bei jedem PUT der Teil eines Records veraendert, bei dem das PUT beginnt bis zum Recordende. Wenn der uebertragene Teil nicht ausreicht, werden Fuellzeichen eingefuegt: ACHTUNG !!! Jedes Bestandteil eines Records, das hinter dem geschriebenen Teil dieses Records steht, wird geloescht, da der Record an dieser Stelle dann nur noch Fuellzeichen enthaelt !

6. Die maximale Summe aller Bytes in einem Record (Steuerbytes, Daten und Strings) kann die im OPEN angegebene Laenge len ueberschreiten, solange nicht mehr als 'len' Bytes bei einem PUT uebertragen werden.

Bei Fixed Item Files gilt:

Von einer Stringvariablen werden soviele Bytes in den File uebertragen, wie es im Vorsatz vor der Variablen angegeben ist. Die Variable wird ggf. abgeschnitten oder der Record mit Leerzeichen aufgefuellt.

8.9.4 Marked und Fixed Item Files

1. Ein Record kann mit mehreren PUT's in den File geschrieben werden.

2. Die Daten werden zwar in den Filebuffer geschrieben, dieser wird jedoch erst dann auf die Floppy geschrieben, wenn

   a. der File geschlossen wird,
   b. der Buffer einen anderen Teil des Files enthalten muss oder
   c. ein PUT fan,& ausgefuehrt wird.

3. Der Fehler 'RECORD OVERFLOW' wird gemeldet, wenn mehr Daten in den Record geschrieben werden, als die Recordlaenge es erlaubt.

4. Siehe 'OPEN' fuer die Anzahl der Bytes bei numerischen Daten.

8.9.5 PUT fan,&

Mit diesem Statement kann der Programmierer veranlassen, dass der bereits veraenderte, aber noch nicht geschriebene Buffer auf die Floppy geschrieben wird. Enthaelt der Buffer keine veraenderten Daten, wird das Statement ignoriert. Zur Erinnerung sei gesagt, dass der Buffer nicht unbedingt bei einem PUT auf die Floppy geschrieben wird, da noch weitere PUT's auf den gleichen Teil erfolgen koennen. Mit PUT fan,& kann dies umgangen werden. Es sollte in folgenden Faellen erfolgen:

1. Es dauert einige Zeit, bis wieder Daten auf die Floppy geschrieben werden, und der File soll nicht so lange geschlossen werden.

2. Eine einwandfreie Verbindung mit anderen Filebereichen ist nur gewaehrleistet, wenn die aktuellsten Daten in diesem File stehen.

3. Es handelt sich um sehr wichtige Daten.

8.9.6 PUT fan,&&

Mit diesem Statement kann der Programmierer veranlassen, dass der neue EOF-Wert in das Directory geschrieben wird. Auf diese Art kann der Programmierer verhindern, dass fuer die aktuelle EOF-Position der File geschlossen werden muss und wieder eroeffnet. Denn nur so kann ueber LOC(fan)! der EOF-Wert erfragr werden. Bevor der EOF-Wert in den File geschrieben wird, ruft diese Funktion das PUT fan,& auf und schreibt den aktiuellen Buffer auf den File.

Beispiel:

```
PUT 2,&&
```

8.9.7 PUT mit !$..

Das PUT-Statement in der Form !$RBA und !$% funktionieren auf die gleiche Art wie beim GET.

8.9.8 PUT mit !#RBA

Das PUT-Statement in der Form !#RBA setzt den EOF-Wert des Files auf den momentanen RBA-Wert. Sonst wird nichts geaendert im Filebereich. Der neue EOF-Wert wird allerdings erst dann auf die Floppy geschrieben, wenn ein CLOSE erfolgt.

Beispiel:

```
PUT 2,!#2000
```

8.10 REMRA UND REMBA

Innerhalb des File-Control-Blocks werden von BASIC zwei zusaetzliche Werte geschrieben mit der relativen Fileposition. Dies sind:

1. REMRA - REMembered Record Address (letzte Recordadresse)
2. REMBA - REMembered Byte Address (letzte Byteadresse)

wobei

1. Sowohl REMBA als auch REMRA im RBA-Format sind.
2. Jedes OPEN und jedes GET/PUT mit dem Parameter !#RBA diese zwei Werte auf ungueltig setzt.
3. Jedes PRINT und INPUT setzt den Wert REMRA auf den Recordanfang. REMBA wird nicht benutzt bei PRINT/INPUT.
4. Jedes GET/PUT mit der Positionierung Null, RN, !RBA, !% oder * (* nur, wenn der REMRA ungueltig ist oder der File nicht Record-segmentiert ist) setzt REMRA auf den Wert der File- Positionierung.
5. Jedes GET/PUT mit der Positionierung Null, RN, !RBA, !% oder * setzt REMBA auf den Wert der Filepositionierung.
6. Der Wert REMRA wird benutzt, wenn der File mit # positioniert wird.
7. Der Wert REMBA wird benutzt, wenn der File mit # positioniert wird.
8. Siehe "LOC-Funktion" auf Seite 145 fuer die Benutzung von LOC(fan)# fuer REMRA.

8.11 PSEUDO FIELD - FUNKTION

Bei Fixed Item Files ist das FIELD-Statement nicht erlaubt. Die Zuweisung von Variablen an einen bestimmten Platz im Record wurde ueber den IGEL gemacht und kann ueber eines der folgenden Statements aufgerufen werden:

1. GET fan,%,IGELSN
2. GET fan,%,,IGEL
3. PUT fan,%,IGELSN
4. PUT fan,%,,IGEL

wobei:

1. fan und IGELSN sind erlaeutert im Anhang, IGEL ist erlaeutert in "IGEL" auf Seite 133

2. fan wird nur fuer das Protokoll angegeben, ob der File eroeffnet ist oder welcher File eroeffnet ist, spielt keine Rolle fuer diese Pseudo-FIELD-Operation. Diese Funktion initialisert nur den IGEL und hat keine Auswirkung auf den Filebereich.

3. Der IGEL wird folgendermassen abgearbeitet:

   a. Numerische Variable bleiben unveraendert.

   b. Bestandteile in der Form (len)$ und (len)% werden ignoriert.

   c. Stringvariable werden auf die Laenge initialisiert, wie sie im Vorsatz vor dem Variablennamen im IGEL angegeben ist. Die Variable wird entweder abgeschnitten oder mit Leerzeichen aufgefuellt. Vom Kuerzen oder Auffuellen der Variablen abgesehen wird diese nicht veraendert. Wenn sich der String nicht im Variablenbereich befindet (z. B. nach READ/DATA), wird er dahin uebertragen. Er sollte dann nur noch mit LSET oder RSET veraendert werden.

8.12 LOC-FUNKTION

In NEWDOS/80 DISK BASIC wurde die Funktion LOC(fan) wie folgt definiert:

LOC(fan) fan ist die Nummer eines Filebereichs (1...15), der als File mit fester Recordlaenge eroeffnet wurde. Diese Funktion gibt einen Integerwert zwischen 1 und 32767 zurueck, die der Anzahl der vorhergehenden bearbeiteten Records entspricht. 0 bedeutet, dass noch kein Zugriff erfolgt ist.

LOC(fan)$ Bei Record-segmentierten Files gibt diese Funktion -1 zurueck (bei Abfrage mit IF = TRUE), wenn der Anfang des naechsten Records (wenn REMRA gueltig ist) oder die momentane Fileposition (wenn REMBA gueltig ist) groesser oder gleich EOF ist. Ist die Position kleiner als EOF, ist LOC(fan)%=0 (bei IF-FALSE). LOC(fan)% unterscheidet sich dadurch von der Funktion EOF, als dass diese nur den exakten EOF-Wert ueberprueft.

LOC(fan)% Diese Funktion gibt den EOF-Wert des Files im RBA-Format zurueck.

LOC(fan)! Bei Record-segmentierten Files gibt diese Funktion folgendes zurueck:

- Wenn REMRA gueltig ist, die Adresse des naechsten Records.
- Wenn REMRA un gueltig ist, die Adresse der momentanen Fileposition.

Bei nicht Record-segmentierten Files und bei PRINT/INPUT-Files erhaelt man immer den RBA- Wert der momentanen File-Position.

LOC(fan)# Diese Funktion gibt den RBA-Wert von REMRA zurueck. Wenn REMRA ungueltig ist kommt es zu einer Fehlermeldung.

Die Benutzung von LOC(fan)! und LOC(fan)# erlaubt es dem Programmierer, fuer eine Gruppe von Daten in einem nicht Record-segmentierten File oder fuer einen Record in einem Record-segmentierten File die genaue Byteadresse abzufragen und sie in einer Indextabelle festzuhalten. Mit dieser Indextabelle

kann man spaeter direkt ueber die Werte !RBA oder !#RBA als Fileposition auf die Daten zugreifen.

8.13 I/O - FEHLER

Die Funktion der Statements PRINT, PUT, INPUT und GET wurde im DISK-BASIC so abgeaendert, dass im Falle eines Fehlers bei der Abarbeitung eines dieser Statements der Filebereich nicht veraendert wird. Dadurch erhaelt der Programmierer mehr Moeglichkeiten, wenn ein Fehler auftritt.

1. Wenn beispielsweise das Programm beim sequentiellen OUTPUT eines PRINT/INPUT-Files den Fehler 'DISK SPACE FULL' meldet, kann man ueber EOF die EOF-Adresse erfragen und den File schliessen. Wenn kein anderer File auf dieser Diskette eroeffnet ist, kann man eine andere Diskette anstelle der vollen in das Laufwerk legen, einen File mit dem gleichen Namen und der gleichen Filenummer eroeffnen und das Programm mit dem fehlerhaften PRINT fortsetzen. Beim spaeteren INPUT kann ebenfalls mit dem EOF-Wert der ersten Diskette der File auf der 2. Diskette weitergelesen werden.

2. Wenn beim Schreiben auf einen "MU"-File mit mehreren PUT's ein Record geschrieben wird, und beim 2. PUT der Fehler 'DISK SPACE FULL' auftritt, wird der EOF-Wert nicht auf Recordanfang zurueckgesetzt, sondern auf den Anfang des 2. PUT's, also mitten im Record. Bevor man jetzt die Diskette wechselt muss der EOF-Wert auf Recordanfang gesetzt werden. Dies geschieht mit den folgende 2 Statements:

   ```
   X! = LOC(fan)# : PUT fan,!#X!
   ```

   Dann kann dieser File geschlossen werden, die neue Diskette in das Laufwerk geschoben werden und ein neuer File mit der Nummer eroeffnet werden. Das Schreiben auf den neuen File muss am Recordanfang beginnen, nicht mit dem fehlerhaften PUT ! Dies ist deshalb, weil ein Record im MU-File immer mit einem SOR-Byte anfangen muss und, wenn mehrere Datenbytes mit mehreren PUT's in einen Record geschrieben werden, der erste Teil in einem halben Record stehen darf und der 2. Teil den Rest des Records enthalten darf.

```
**************************************************************
***  Der Anwender und Programmierer muss vor allem darauf  ***
***  achten, dass nur der eine, fehlerhafte File auf der   ***
***  Diskette beim Wechsel eroeffnet war !                 ***
**************************************************************
```

Es sei nochmals darauf hingewiesen, dass die Daten beim fehlerhaften PUT und GET undefiniert sind, also nicht zu gebrauchen.

Um den eigentlichen Filebereich auch bei einem Fehler unveraendert zu lassen, benutzt das DISK-BASIC beim GET, PUT und anderen Fileoperationen einen GETrennten Kontrollbereich, bis das Statement ohne Fehler ausgefuehrt wurde.

8.14 ANMERKUNGEN ZUR DISK I/O

1. Bei Marked Item und Fixed Item Files kann man mit GET und PUT eine Gruppe von beliebigen Daten in einen File schreiben bzw. lesen. Die Anzahl der Daten wird nur begrenzt durch die Filegroesse und, wenn vorhanden, die

Recordlaenge. Logische Records koennen jede Laenge zwischen 1 und 4095 Bytes haben. Der Programmierer sollte niemals direkt auf den Filebuffer zugreifen. Wenn ein Programm Daten im Filebuffer eines Files, der fuer eine andere Zugriffsart als mit dem FIELD-Statement eroeffnet wurde, direkt veraendert, sind diese Daten nicht korrekt und koennen den File zerstoeren ! Extrem vorsichtig muss man sein, wenn der File mit einem FIELD-Statement veraendert, dann geschlossen und fuer eine andere Zugriffsart erneut eroeffnet wurde. Es duerfen dann keine LSET- oder REST-Statements mehr ausgefuert werden !

2. Die speziellen Funktionen fuer die Field Item Files MKD$, MKI$, MKS$, CVD, CVI, CVS, LSET und RSET arbeiten wie zuvor. Die Benutzung von MKD$, MKI$, MKS$, CVD, CVI, CVS, LSET und RSET sollte aber trotzdem eingestellt oder auf ein absolut notwendiges Minimum eingeschraenkt werden.

3. Bei GET- und PUT-Statements mit einer IGELSN wird im Fehlerfall nicht die Statementnummer des IGEL's, sondern nur die Statementnummer des GET's oder PUT's angezeigt, das diesen IGEL benutzt.

4. Um die Fehlererkennung in einem GET -oder PUT-Statement mit IGELSN leichter zu machen, sollte ein solches Statement das einzige in einer Zeile sein.

5. Ein File kann nur im Mode R oder D veraendert werden. MI und PRINT/INPUT-Files koennen nicht veraendert werden, nur erweitert. Ein Record im MU-File kann veraendert werden, jedoch darf die neue Recordlaenge die urspruengliche nicht ueberschreiten. Dies gilt nicht fuer den letzten Record eines Files.

6. Files, die fuer einen Zugriff mit PRINT/INPUT eroeffnet wurden, duerfen auch mit GET/PUT gelesen oder beschrieben werden, wenn der Positionierungsparameter in der Form !$RBA, !$%, !&RBA, & oder % ist.

7. BASIC-Funktionen, die eine Filenummer haben (z. B. EOF, LOC, LOF etc.), duerfen nicht innerhalb eines IGEL's oder im OPEN, GET, PUT, CLOSE, PRINT (auf einen File) oder INPUT (von einem File) auftreten. Diese Einschraenkung gibt es nur im NEWDOS.

8. Bei Files, deren Records ueber zwei oder mehr Sektoren auf der Floppy gehen (PRINT/INPUT, Fixed Item und Marked Item Files), kann die Anzahl der Diskettenzugriffe um mehr als 200 % nach oben gehen (verglichen mit Field Item Files), wenn ein Record oder eine Datengruppe gerade ueber 2 oder mehr Sektoren geht. Die Zunahme der Zugriffe errechnet sich aus (LEN/256)*200, wobei LEN die durchschnittliche Recordlaenge und kleiner als 256 ist. Fuer Recordlaengen von > 256 gibt es keine Annaeherung.

9.0 FEHLERMELDUNGEN

9.1 DOS - FEHLERMELDUNGEN

Folgende Fehlercodes werden im A-Register bei einem CALL oder JUMP zu der Routine 4409H auf dem Bildschirm angezeigt. Die Fehler sind aufgelistet in dezimal und hexadezimal.

dez.	hex.	Fehlermeldung
00	00	Kein Fehler
01	01	Parity - Fehler beim Lesen des Start - Satzes
02	02	Seek - Fehler beim Lesen
03	03	Daten verloren beim Lesen
04	04	Parity - Fehler beim Lesen
05	05	File nicht gefunden beim Lesen
06	06	Leseversuch auf geschuetzten Record
07	07	Leseversuch auf System - Record
08	08	Geraet nicht verfuegbar
09	09	Undefinierter Fehlercode
10	0A	Seek - Fehler beim Schreiben
11	0B	Daten verloren beim Schreiben
12	0C	Parity - Fehler beim Schreiben
13	0D	File nicht gefunden beim Schreiben
14	0E	Schreibversuch nicht erfolgreich
15	0F	Schreibversuch auf geschuetzten File
16	10	Geraet nicht verfuegbar
17	11	Lesefehler Directory
18	12	Schreibfehler Directory
19	13	Falscher File - Name
20	14	Spurnummer zu hoch
21	15	Falsche Funktion im DOS-CALL
22	16	Undefinierter Fehlercode
23	17	Undefinierter Fehlercode
24	18	File nicht im Directory
25	19	File-Zugriff zurueckgewiesen
26	1A	Directory voll
27	1B	Diskette voll
28	1C	End of File erreicht
29	1D	Nach End of File
30	1E	Directory voll bei Erweiterung
31	1F	Programm nicht gefunden
32	20	Falsche oder keine Diskettennummer
33	21	Kein Platz auf Diskette
34	22	Fehler im Load - File
35	23	Fehler im Memory
36	24	Zugriff auf ROM
37	25	Zugriff auf geschuetzten File
38	26	File nicht eroeffnet
39	27	Diskette nicht oder falsch initialisiert
40	28	Falsche Spur auf Diskette
41	29	Nicht erlaubte File - Nummer
42	2A	Nicht erlaubte DOS-Funktion
43	2B	Nicht erlaubte Funktion bei Chaining
44	2C	Falsche Daten im Directory
45	2D	Falsche FCB Daten
46	2E	Systemprogramm nicht gefunden
47	2F	Falsche Parameter
48	30	Falsche File - Bezeichnung
49	31	Falscher Record - Typ
50	32	Fehler im Boot - Lader
51	33	DOS - Fehler
52	34	Falsches Kommando
53	35	Doppelter File
54	36	Kommando zu lang
55	37	Nicht erlaubter Zugriff

56	38	Nicht erlaubte MINI-DOS Funktion
57	39	Abbruch einer Funktion
58	3A	Falscher Datentyp
59	3B	Memory nicht ausreichend
60	3C	Diskette falsch spezifiziert
61	3D	ASE=N, keine Erweiterung moeglich
62	3E	Keine Erweiterung ueber READ

Wenn ein Fehlercode nicht definert ist, wird UNKNOWN ERROR CODE angezeigt.

SYS4/SYS ist das Modul, das die DOS-Fehlermeldungen anzeigt.

9.2 BASIC - FEHLERMELDUNGEN

Zusaetzlich zu den normalen Fehlermeldungen des Level-II BASIC gibt es folgende Fehlercodes:

dez.	hex.	Fehlermeldung
51	33	Feld - Ueberlauf
52	34	Interner Fehler
53	35	File - Nummer falsch
54	36	File nicht gefunden
55	37	Falscher File - Mode
56	38	File bereits eroeffnet
58	3A	DOS - Fehler
59	3B	File existiert bereits
62	3E	Diskette voll
63	3F	Input nach End of File
64	40	Falsche Record - Nummer
65	41	Falscher File - Name
66	42	Falscher File - Mode
67	43	Kommando im File
68	44	Zu viele Files
69	45	Diskette mit Schreibschutz
70	46	Filezugriff nicht erlaubt
71	47	Ueberlauf Sequenz - Nr.
72	48	Record - Ueberlauf
73	49	Fileerweiterung verboten
75	4B	Bereits angezeigter Fehler
76	4C	Zeile 0 nicht zu verarbeiten
77	4D	Falscher File - Typ
78	4E	IGEL Syntax - Fehler
79	4F	IGEL Item Syntax - Fehler
80	50	Falsches oder fehlendes IGEL Item
81	51	IGEL item ohne Typangabe
82	52	Falsche Record - Laenge
83	53	Statement mit 2 Files
84	54	Falscher Positionierungs-Parameter

Der File SYS13/SYS zeigt normalerweise alle Disk-BASIC und ROM- BASIC-Fehler im Klartext an. Dieser File ist normalerweise nicht im Memory und wird erst bei einem Fehler angezogen. Wenn dieser File, z. B. aus Platzgruenden, geloescht wird, wird nur die Fehlernummer angezeigt.

Wird ein Fehler gemeldet, der nicht in dieser Liste ist, wird die Meldung 'UNPRINTABLE ERROR' angezeigt.

10.0 ZUSAMMENFASSUNG

Dieses Kapitel enthaelt die Erklaerung der Definitionen, die in diesem Handbuch benutzt wurde. In manchen Faellen ist eine bessere Erlaeuterung in diesem Handbuch, es wird dann hier nur auf diese Stelle verwiesen.

Bit Die kleinste Einheit sowohl im Speicher als auch auf der Floppy, auf die man zugreifen kann. Ein Bit hat entweder einen Wert von 0 (aus) oder 1 (an). Eine Gruppe von 4 Bits zusammen nennt man Digit, eine Gruppe von 8 Bits ist ein Byte. Die Bits in einem Byte werden von rechts aufsteigend nach links gezaehlt, beginnend bei 0. Das ganz linke Bit ist also Bit 7. Die Bits in einem Digit sind dementsprechend 3 ... 0.

BOOT/SYS Einer der zwei Kontrollfiles, die auf jeder NEWDOS/80 Diskette vorhanden sein muessen. Siehe "BOOT/SYS" auf Seite 77.

Buffer Ein Bereich im Speicher, der die Daten beim sektorweisen Lesen von der Floppy oder die neuen Daten beim Schreiben auf die Floppy enthaelt. Jeder FCB eines eroeffneten Files hat einen Buffer von 256 Bytes Laenge.

Byte Die kleinste adressierbare Einheit sowohl im Speicher als auch auf der Floppy. Es werden abwechselnd sowohl Byte als auch das Wort Zeichen (Character) benutzt. Ein Byte besteht aus 8 Bit und sein Wert wird in 2 hexadezimalen Digits angegeben. Im NEWDOS/80 wird von Zeichen, Byte und Charactern gesprochen. Diese Ausdruecke bedeuten zwar alle das Gleiche, sind aber manchmal etwas genauer.

Character Wird anstelle von Byte benutzt, ebenfalls die Bezeichnung fuer ein Byte, das ein druckbares Bitmuster enthaelt.

DEC - Directory Entry Code Eine 1 Byte lange Information ueber einen Directory Entry. Dieser DEC wird vom System benutzt, einen Eintrag im Directory oder eine freie Stelle schnellstmoeglich zu finden. Ist ein FCB eroeffnet, so enthaelt sein 8. Byte den DEC des FPDE's. Jeder FXDE enthaelt in seinem 2. Byte den DEC des dem File zugehoerenden FDE, und jeder FPDE oder FXDE, dessen 31. Byte = 255 (0FEH) ist, enthaelt auf dem 32. Byte den DEC des naechsten FXDE's. Das Format des 8 Bit langen DEC's ist :

rrrssssss

wobei sssss+2 die relative Sektornummer innerhalb des Directory ist, in dem der FDE steht, und rrr*32 die relative Byteadresse innerhalb dieses Sektors ist.

DIR/SYS Einer der zwei Kontrollfiles, die auf jeder NEWDOS/80 Diskette vorhanden sein muessen. Siehe "DIR/SYS" auf Seite 77 und "Struktur des Directory" auf Seite 81

EOF - End Of File Der RBA - Wert des letzten Bytes im File plus 1. Er definiert das erste Byte, das benutzt werden kann, wenn der File erweitert wird, und das Ende der Daten. Einige Files haben ein End Of File Byte, das das Ende eines Files kennzeichnet, z. B. Assembler

Source Files benutzen 1AH, BASIC-Files (keine BASIC-ASCII-Files) haben 3 aufeinanderfolgende Nullen etc. Die meisten Files benutzen aber den Wert aus dem FCB als EOF.

EOL - End Of Line Das Ende einer Zeile. Wenn es sich um eine Eingabe ans System handelt, ist es normalerweise das ENTER - Zeichen (0DH). Im BASIC - Text beendet eine Null die Zeile. Wenn eine Zeile keinen speziellen EOL - Character hat, so wird als EOL immer der letzte Character + 1 benutzt.

EOR - End Of Record Kennzeichnet das Ende eines Records. Es ist die Adresse des letzten Bytes + 1 im Record.

EOS - End Of Statement Das Ende eines BASIC - Statements. Das BASIC benutzt das Zeichen ":" um das Ende der Zeile zu definieren.

Extent Element - Erweiterungselement Ein 2 Byte langer Eintrag im FPDE/FXDE, der entweder die Adresse eines Elementes eines Files oder die Adresse eines FXDE's im Directory angibt.

FAN - FileArea Number Ein BASIC - Ausdruck, der einen Wert von 1 bis incl. 15 hat. Er steht fuer den Filebereich, der einem bestimmten File zugewiesen wird (bei OPEN) oder zugewiesen wurde.

FCB - File Control Block Siehe "FCB" auf Seite 86.

FDE - File Directory Entry Siehe "FDE-Sektoren" auf Seite 83.

FF File Ein Fixed Item BASIC File, dessen Records alle die gleiche Laenge haben.

FI File Ein Fixed Item BASIC File, der nicht in Records unterteilt ist.

File oder Disk-File bzw. Diskettenfile Eine Ansammlung von Daten auf einer Platte oder Diskette. Ein File enthaelt entweder Kontroll-Informationen wie z. B. BOOT/SYS oder DIR/SYS, oder ein Maschinenprogramm wie SYS0/SYS bis SYS13/SYS, BASIC/CMD und SUPERZAP/CMD, oder ein BASIC-Programm wie CHAINTST/BAS und SAMPLE01/BAS, oder er enthaelt Benutzerdaten wie z. B. eine Adressenliste. Die Kontrolldaten fuer alle Files stehen in DIR/SYS (siehe "DIR/SYS" auf Seite 77), wo jeder File einen FPDE und zwischen 0 und mehreren FXDE's hat. Ein File muss zuerst auf der Diskette existieren. Der Platz fuer den File wird ihm nach Bedarf in Granules zugewiesen. Die nicht benoetigten Granules werden beim CLOSE wieder freigegeben.

Filearea Ein Kontrollbereich des BASIC, der ungefaehr 300 Bytes lang ist und den FCB sowie den Buffer eines Files enthaelt. Dieser Bereich schafft die Verbindung zwischen BASIC - Programm und dem angesprochenen File und ist gueltig, solange der File geoeffnet ist. Dieser Bereich wird benutzt und angesprochen bei den Statements INPUT, PRINT, GET, PUT, FIELD, LOC, LOF und EOF. Alle Bereiche von Files arbeiten unabhaengig voneinander, auch dann, wenn mehrere gleichzeitig geoeffnet sind. Die Anzahl der verfuegbaren Filebereiche wird beim Aufruf von BASIC festgelegt (siehe "Disk - BASIC" auf Seite 110).

Field Item File Dieser Name wird im NEWDOS/80 fuer die Files benutzt, die im TRSDOS - BASIC Random Files genannt werden (Files mit wahlfreiem Zugriff). Es koennen jedoch alle Files, sowohl Marked Item, als auch Field Item und Fixed Item Files, Random Files sein. Bei Field Item Files werden bei GET's und PUT's

jeweils immer der ganze Sektor mit dem FIELD-Statement unterteilt und beeinflusst.

File Item Dies ist eine Einheit von File - Speicherplatz, die entweder null oder mehr Bytes lang ist und entweder einen String oder einen numerischen Wert in seiner internen Form enthaelt.

FP - File Positionierung Positionierungsparameter siehe "FP" auf Seite 131.

FPDE - File Primary Directory Entry Haupteintrag eines Files im Directory - siehe "FPDE" auf Seite 84

FXDE - File eXtented Directory Entry Zusatzeintrag eines Files im Directory - siehe "FXDE" auf Seite 85

GAT - Granule Allocation Table Belegungstabelle im Directory - siehe "GAT-Sektor" auf Seite 82

Granule Die kleinste, reservierbare Einheit auf einer Floppy. Ein Granule besteht aus 5 Sektoren, die Sektornummern beginnen mit 0 oder 5.

Hexadezimal o. hex. Zahlensystem, auf 16 anstatt auf 10 basierend wie beim Dezimalsystem. Die verwendbaren Zahlen sind 0-9 und A-F.

HIT - Hash code Index Table Tabelle des Hashcodes jedes Directory - Eintrags. Siehe "HIT-Sektor" auf Seite 83.

IGEL - Item Group Expression List Liste der Bestandteile eines Records. Siehe "IGEL" auf Seite 133.

IGEL Expression Siehe "IGEL" auf Seite 133.

IGELSN - IGEL - Statement Number Nummer des BASIC - Statements, in dem der IGEL steht bzw. beginnt.

Item Group Eine Gruppe von mehreren oder keinen Bestandteilen eines Records. In BASIC die Gruppe von Bestandteilen, die bei einem GET, PUT, PRINT und INPUT meist das Abbild des momentanen Records darstellt.

LEN oder len Laengenangabe. Siehe "len" auf Seite 138.

Logical Record Eine Gruppe von Bestandteilen eines Files. Die Filedaten sind in physikalische Records unterteilt, der Programmierer kann jedoch seine Daten in eigene Gruppen unterteilen, die dann sog. logische Records sind.

LREL - Logical RECord Length Entweder feste oder maximale Recordlaenge fuer die Records in einem File. Fuer Nicht-BASIC-Files ist die maximale LRECL 255 Bytes (0 = 256 Bytes, steht im 4. Byte des FPDE's und wird nie benutzt). Im BASIC wird LRECL als (len) angegeben und ist maximal 4095 Bytes. Siehe Kapitel 8.4.6.

MF File Ein Marked Item File, der in Records von gleicher Laenge unterteilt ist.

MI File Ein Marked Item File, der nicht in Records unterteilt ist.

MU File Ein Marked Item File, der in Records von unterschiedlicher Laenge unterteilt ist.

Null Das Fehlen eines Parameters oder eines Ausdrucks. Wenn Parameter durch Komma getrennt werden, sind in diesem Falle 2 aufeinanderfolgende Kommas zu schreiben (,,).

Null Character oder Zeichen Ein Character oder ein Byte mit dem Wert 0.

Nullstring Ein String mit der Laenge 0, kann auch ein Ausdruck sein, der einen String mit der Laenge 0 ergibt.

Partial Record I/O Bei dieser Zugriffsart werden nur die Bestandteile eines Records uebertragen, die absolut notwendig sind. Diese Art des Zugriffs ist zwar wesentlich effizienter als die komplette Uebertragung eines Records (Whole Record I/O), dafuer leidet aber etwas die Lesbarkeit und die Verstaendlichkeit eines Programmes darunter. Diese Zugriffsart benutzt alle Varianten der Filepositionierung (FP), die fuer eine Datenuebertragung notwendig sind. Man kann damit auf alle NEWDOS/80 Filetypen zugreifen beim Lesen, auf "FF" und "FI" File kannen damit ein Update erfolgen.

RBA - Relative Byte Address Ein BASIC - Ausdruck, der einen Wert zwischen 0 und 16.777.215 annehmen kann. Er gibt die Adresse eines bestimmten Bytes innerhalb eines Files an. Dieses Byte kann der Anfang eines Bestandteils eines Records sein, der Anfang einer Gruppe von Bestandteilen oder der Anfang eines logischen Records. Der RBA-Wert ist nichts weiter als ein 3 Byte langer Integerwert, dessen ersten 2 Bytes die relative Sektornummer innerhalb des Files bilden (max. 65535), das 3. Byte ist die relative Position des Bytes innerhalb des Sektors (max. 255). In dieser Art werden Fileposition und EOF - Wert im FCB abgespeichert. Die Benutzung des RBA - Wertes fuer EOF ist der grundsaetzliche Unterschied zwischen TRSDOS und NEWDOS/80. Programme, die den EOF-Wert benutzen wie im TRSDOS - Handbuch beschrieben, koennen in dieser Art nicht unter NEWDOS/80 arbeiten.

REMBA - REMembered Byte Address Feld im FCB mit der letzten Byteadresse - siehe "REMRA und REMBA" auf Seite 144

REMRA - REMembered Record Address Feld im FCB mit der letzten Recordadresse - siehe "REMRA und REMBA" auf Seite 144

Sector Der kleinste einlesbare Block auf einer Floppy. Er enthaelt 256 Bytes.

SOR - Start Of Record Der Beginn eines Records bei "MU" Files. Das NEWDOS/80 BASIC benutzt das Zeichen 70H zur Kennzeichnung eines Records.

Track - Spur Die Speichereinheit, die waehrend einer Umdrehung der Floppy beim lesen oder schreiben ueberstrichen wird. Eine Spur (Track) besteht aus mehreren Sektoren. Bei der Minidiskette des TRS80 hat eine Spur 10 Sektoren oder 2 Granules, die bei Sektor 0 einer Spur anfangen.

Whole Record I/O Bei dieser Zugriffsart werden mit jedem GET und PUT immer alle Daten eines logischen Records uebertragen. Es ist zwar am einfachsten zu programmieren, hat jedoch den Nachteil, dass eben immer alle Daten eines Records uebertragen werden muessen, egal ob sie gebraucht werden oder nicht. Diese Technik muss bei updates auf "MU" und "MF" Files benutzt werden, der restliche, freibleibende Platz wird mit Fuellbytes aufgefuellt (hex. 0)

ZAP Das Aendern von Daten oder Programmen. Siehe Kapitel 11.1. Wird auch PATCH genannt.

11.0 FEHLERRUECKMELDUNG UND DURCHFUEHRUNG VON AENDERUNGEN

11.1 ALLGEMEINES

Da das NEWDOS/80 vermutlich wie seine Vorgaenger bisher unbekannte Fehler enthaelt und eventuell teilweise inkompatiebel ist mit seinen Vorgaengern, was erst nach und nach von den Kunden bemerkt wird, muss der Benutzer anhand von Aenderungsmitteilungen des Herstellers APPARAT diese Aenderungen in seinem Betriebssystem vornehmen. Die Fa. APPARAT wird, sobald Fehler bekannt werden, Aenderungen im Betriebssystem vornehemen und diese den Kunden mitteilen.

Die Fa. APPARAT bittet deshalb die Kunden darum, auftretende Fehler mitzuteilen, damit die noetigen Massnahmen ergriffen werden koennen. Dies duerfte zwar fuer den deutschen Markt, und diesen spricht dieses Handbuch an, nicht ganz zutreffen, da das NEWDOS/80 hier erst Wochen oder Monate spaeter verkauft wird. Meist sind dann die Fehler schon bekannt und die entsprechende Aenderung schon veranlasst, die dann aber auch erst wieder Wochen spaeter hier beim Haendler erscheint.

Fa. APPARAT weist darauf hin, dass mehr als die Haelfte der Aenderungen durch Kundenmitteilungen veranlasst wurden. Dafuer waren es aber auch ueber 60 Aenderungen. Im neuen Handbuch sieht es allerdings auch nicht wesentlich besser aus. Bereits in dieser Ausgabe sind schon mehrere Aenderungen enthalten, Fehler sind auch schon genuegend bekannt.

11.2 UNVERTRAEGLICHKEIT MIT ANDEREN BETRIEBSSYSTEMEN

Das NEWDOS/80 ist ein anderes Betriebssystem als andere DOS-Arten wie TRSDOS, VTOS, LDOS, DOSPLUS etc. und unterscheidet sich deshalb zum Teil in wesentlichen Punkten von diesen. Aus diesem Grunde ist es moeglich, dass viele Anwenderprogramme vom NEWDOS in einem anderen Betriebssystem nicht oder erst nach umfangreichen Aenderungen lauffaehig sind. Die beste Moeglichkeit, dies herauszufinden, ist die, dass man ein Programm mit einem anderen Betriebssystem laufen laesst und so die Fehler feststellt. Dass man dafuer nur eine Testfloppy ohne wichtige Daten benutzen sollte duerfte selbstverstaendlich sein.

Die Fa. APPARAT hat in der Vergangenheit versucht, fuer einige viel benutzte Programme die Aenderungen den Kunden mitzuteilen. Aus verschiedenen Gruenden war man aber damit nicht immer erfolgreich:

1. APPARAT wurde von den Herstellern der Programme nicht immer ueber die neuesten Ausgaben und Unvertraeglichkeiten informiert. Aus diesem Grunde stammen einige der Aenderungen von Kunden, diese Aenderungen wurden dann uebernommen und weitergegeben.

2. APPARAT hat nicht die noetige Kapazitaet, jedes kaeufliche Programm auf Vertraeglichkeit zu ueberpruefen und auch Aenderungen zu entwickeln.

3. Der Versand an die vielen registrierten Benutzer wird dadurch verzoegert, dass die Aenderungen nicht einzeln verschickt werden,

Dennoch sind auch noch in dieser Version Aenderungen enthalten, die kaeufliche Programme anderer Hersteller fuer andere Betriebssysteme betreffen. APPARAT versucht aber, die Hersteller soweit zu beeinflussen, dass diese die Aenderungen fuer andere Betriebssysteme mit dem Programm vertreiben. Aus

diesem Grunde erhalten alle Hersteller von kaeuflicher Software kostenlos ein NEWDOS/80, damit gleich fuer NEWDOS/80 passende Programmversionen angeboten werden koennen.

11.3 MITTEILUNGEN AN APPARAT

Bei Mitteilungen an APPARAT ueber notwendige Aenderungen beachten Sie bitte die nachfolgenden Punkte. Sie sparen sich damit unnoetigen Aerger und Zeit.

1. Lesen Sie und versuchen Sie auch die vorliegende Dokumentation zu verstehen.

2. Ueberzeugen sie sich, dass alle Programme, also auch BASIC-Programme, richtig dem Betriebssystem angepasst sind. Es werden grundsaetzlich nur die Programme ueberprueft, die auch von APPARAT geschrieben wurden.

3. Ueberzeugen Sie sich, dass alle Aenderungen im Betriebssystem durchgefuehrt wurden.

4. Versuchen Sie mehrmals und unter verschiedenen Konditionen den Fehler zu erzeugen.

5. Beschreiben Sie praezise und genau den auftretenden Fehler (in Ihrem besten englisch, bitte) und senden Sie diese Beschreibung an

APPARAT Inc.
4401 S. Tamarac Pkwy.
Denver, CO 80237
U. S. A.

6. Geben sie die Seriennummer ihres NEWDOS/80 mit an.

7. Legen Sie eine Kopie der Diskette bei, auf der das oder die Programme sind, bei dem oder bei denen der Fehler auftritt. APPARAT wird, sobald der Fehler analysiert wurde, Ihre Diskette loeschen. Die Diskette wird als Geschenk betrachtet, Sie erhalten sie selbstverstaendlich nicht zurueck !!!

8. Rufen Sie nicht an!!! Erstens meldet sich sowieso am Telefon, und zweitens gehen die Programmierer von APPARAT grundsaetzlich nicht ans Telefon.

9. Geben Sie bitte nicht auch im gleichen Schreiben Bestellungen auf.

11.4 FORMAT DER AENDERUNGEN

In NEWDOS/80 geschieht dies, ebenso wie in aelteren NEWDOS-Versionen, mittels des mitgelieferten Programms 'SUPERZAP'. Der Benutzer sei nochmals daran erinnert, dass der 'SUPERZAP' in NEWDOS/80 ein Maschinenprogramm ist, in den frueheren NEWDOS war dies ein BASIC - Programm. Einige Funktionen sind unterschiedlich, deshalb sollte man zuerst das Kapitel "SUPERZAP" auf Seite 90durchlesen. Fuer die durchzufuehrenden Aenderungen steht aber auch in den folgenden Abschnitten genuegend Information.

Alle NEWDOS/80-Aenderungen sind nummeriert und haben das Datum des Aenderungstages. Eine solche Aenderung ist entweder

zwingend vorgeschrieben oder optional. Sind diese Aenderungen zwingend vorgeschrieben, so haben alle NEWDOS/80, die nach dem Aenderungsdatum ausgeliefert wurden, bereits den neuesten Stand. Sollte es dennoch vorkommen, dass ein NEWDOS/80 mit neuerem Datum noch einen alten Stand hat, so ist es darauf zurueckzufuehren, dass die Diskette zwar zu diesem Zeitpunkt bereits kopiert, aber noch nicht ausgeliefert war.

Jede Aenderung enthaelt eine kurze Beschreibung ihrer Funktion und ihrer Auswirkung. Die eigentliche Aenderung besteht aus drei Teilen in folgendem Format:

1. Der Platz auf der Diskette, wo die Aenderung durchgefuehrt werden soll. Diese Angabe besteht aus drei Parametern mit folgender Bedeutung:

   FNAME,Sektor,Bytenummer

   a. 'FNAME' ist der komletta Name (FNAME/TYP) des zu aendernden Programmes.

   b. 'Sektor' ist der relative Sektor innerhalb dieses Files. Dieser Sektor wird dezimal angegeben.

   c. 'Bytenummer' ist das zu aendernde Byte innerhalb dieses Sektors oder der Anfang dieser Aenderung. Die Nummer des Bytes ist in hexadezimal, allerdings ohne den Zusatz 'H' angegeben.

   Beispiel:

   ```
   DIR/SYS,02,20
   EDTASM/CMD,20,F6
   XFILE,00,88
   ```

2. Die alten Werte im zu aendernden Bereich. Jedes Byte wird mit zwei Zeichen, also ohne fuehrende Null und ohne 'H' am Ende, dargestellt. Zur besseren Unterscheidung werden die Bytes durch mindestens ein Leerzeichen getrennt. Wird anstelle eines Bytes nur ein '-' angegeben so heisst das, dass entweder im Augenblick der Aenderung nicht bekannt ist, was dieses Byte tatsaechlich enthaelt, oder dass dies unwichtig ist.

3. Die neuen Werte, die einzugeben sind, werden genau wie die alten Werte dargestellt.

11.5 AENDERUNGSPROZEDUR

1. Machen sie mindestens 1 Kopie der Diskette, bevor sie die Aenderung durchfuehren. Niemals ohne Backup-Kopie eine Aenderung machen!!!

2. DOS-Kommando 'SUPERZAP' eingeben.

3. Diskette, auf welcher die Aenderung durchzufuehren ist, einlegen.

4. Geben sie die SUPERZAP-Funktion 'DFS' ein.

5. Geben Sie den kompletten Filenamen an, also incl. Typ, der im Parameter FNAME angegeben ist. Wurde der Name dieses Files inzwischen von ihnen mit RENAME geaendert, geben sie den neuen Namen an. Passwort eingeben (falls erforderlich) und Nummer des Laufwerks (kann auch weggelassen werden).

6. Geben Sie den 2. Parameter, den relativen Sektor innerhalb des Files, ein.

7. Der gewuenschte Sektor wird auf dem Bildschirm angezeigt. Kontrollieren Sie zuerst an der angegebenen Stelle die alten Werte. Falls Sie diese nicht finden koennen, schauen Sie nach, ob eventuell diese Aenderung bereits vorhanden ist. Wenn ja, brauchen Sie nichts mehr zu tun. Wenn jedoch grundsaetzlich andere Werte an der angegebenen Stelle stehen, setzen Sie sich bitte mit ihrem Haendler in Verbindung.

8. Haben Sie die zu aendernde Stelle mit den alten Werten gefunden, geben Sie MOD.. ohne ENTER ein. Die zwei Punkte stehen fuer den Wert der Bytenummer innerhalb des Sektors.

9. Der Cursor sollte auf das 1. Digit des Bytes .. springen. Falls nicht, geben Sie nochmals MOD.. ein. Falls der Cursor an die falsche Stelle springt, ueberpruefen Sie nochmals die eingegebenen Werte und die Reihenfolge! Steht der Cursor auf dem richtigen Byte ist das SUPERZAP im Overwritemode. Sind Sie vorsichtig bei der Eingabe! Um mit dem Cursor an eine andere Stelle zu fahren koennen Sie mit den Tasten Up-, Down-, Left- und Right-Arrow und mit der Leertaste arbeiten.

10. Geben Sie mit den Tasten 0-9 und A-F die Aenderungen ein. Der Cursor springt jeweils automatisch auf die naechste Position.

11. Geben Sie alle Werte dieses entsprechenden Sektors ein.

12. Sollten Sie falsche Werte eingegeben haben oder sich nicht ganz sicher sein, ob die eingegeben Werte korrekt waren, koennen Sie durch Druecken der Taste 'Q' wieder in den Anzeigemodus zurueckkehren. Erscheint das Wort 'CHECK' in der 7. Spalte des Bildschirms haben Sie eine falsche Taste gedrueckt. Geben Sie dann '*' ein. In einer anderen Version des SUPERZAP wird der Bildschirm geloescht und 'CHECK' angezeigt.

13. Haben Sie alle Aenderungen eines Sektors eingegeben und wollen diesen auf die Floppy zurueckschreiben, druecken Sie die ENTER-Taste. Der Bildschirm wird dann geloescht, auf die Frage antworten Sie mit 'Y' und ENTER. Der Sektor wird dann zurueckgeschrieben und SUPERZAP geht in den Anzeigemodus.

14. Wenn Sie im Anzeigemodus sind (Cursor wird nicht angezeigt) gibt es folgende Moeglichkeiten:

    a. Eingabe von 'K' - ein anderer Sektor des gleichen Files kann angegeben werden.

    b. Eingabe von 'J' - ein anderer File soll angezeigt werden.

    c. Eingabe von 'X' - das Startmenu wird angezeigt.

    oder Sie koennen mit Schritt 7 fortfahren.

11.6 AENDERUNGSDIENST

Die Fa. APPARAT verlangt von jedem Benutzer, dass er die Registrierungskarte an sie zurueckschickt. Aenderungen werden nur in schriftlicher Form an die angemeldeten Benutzer ausgegeben. Da dies aber, sollte sich das Betriebssystem als fehlerhaft erweisen, eine teure Angelegenheit werden kann, behaelt sich die Fa. APPARAT vor, die Aenderungen ueber die Haendler herauszugeben oder aber diese Aenderungen in Rechnung zu stellen.

Aenderungen werden grundsaetzlich nicht telefonisch, sondern nur in schriftlicher Form herausgegeben !!!

Normalerweise erhaelt der Benutzer, sobald die Registrierungskarte bei APPARAT eingetroffen ist, den ersten Satz der Aenderungen. Dies erfolgt deshalb zwangslaeufig, da das Handbuch bereits vor dem Betriebssystem fertiggestellt sein muss.

Deutsches Handbuch

In dieser Uebersetzung sind bereits alle Aenderungen enthalten, die bis Ende Dezember 1981 erschienen sind. Sollten spaeter noch weitere Aenderungen erscheinen, so wird ein Nachtrag ueber den Haendler, bei dem Sie dieses Handbuch gekauft haben, erhaeltlich sein.

11.7 AENDERUNGEN AN DER NEUEN DISKETTE

Wenn Sie das NEWDOS/80 neu auf Ihrem System installieren, sehen Sie bitte erst in Kapitel 13 nach, ob auf Ihrer Diskette die Aenderungen erforderlich sind oder nicht. Durch die zeitliche Verzoegerung von Fertigstellung von Handbuch und Betriebssystem kann es vorkommen, dass im Handbuch noch in letzter Minute eine Aenderung eingefuegt wurde, die noch nicht im Betriebssystem ist, oder dass Sie Aenderungsmitteilungen mit Aenderungen erhalten, die bereits in Ihrem Betriebssystem beruecksichtigt sind.

Machen Sie in jedem Falle erst eine oder mehrere Kopieen der gekauften Diskette und bewahren Sie diese gut auf.

Sehen Sie mit dem SUPERZAP nach, ob die letzte Aenderung, die in Kapitel 13 aufgefuehrt ist, in Ihrem System enthalten ist. Wenn ja, koennen Sie davon ausgehen, dass auch alle vorhergehenden Aenderungen bereits enthalten sind. Fuegen Sie dann an allen Programmen, die keine Systemprogramme sind, die fuer Sie notwendigen Aenderungen ein.

Ist die letzte Aenderung im Handbuch noch nicht auf Ihrer Diskette, so fuegen Sie diese und von hier ab in umgekehrter Reihenfolge (also von hinten her) die Aenderungen ein, bis Sie zu der Aenderung kommen, die bereits in Ihrem System ist.

Sie sollten sich aber in jedem Falle alle Aenderungen durchlesen, auch die, die Sie nicht betreffen.

11.8 WEITERE AENDERUNGEN AN DER DISKETTE

Die Aussagen vom vorangegangenen Kapitel gelten auch hier. Wenn Sie neue Aenderungen direkt von APPARAT erhalten (in Deutschland vermutlich von Ihrem Haendler), lesen Sie sich alle durch und beginnen Sie mit den Aenderungen von hinten her auf die Diskette zu uebertragen. Auch hier kann es vorkommen, dass Sie bereits eine neuere Diskette besitzen als Handbuch.

11.9 AENDERUNGSDIENST

Bei der NEWDOS/80 Version 1 wurde bei APPARAT ein Aenderungsdienst aufgebaut, der jetzt fuer die Version 2 uebernommen wurde. Wie schon in der vorangegangenen Version kann ein Benutzer eine Floppy mit dem Betriebssystem an APPARAT senden, zusammen mit einer Aenderungsgebuehr von US-$ 10,00, und er erhaelt dafuer die neueste Ausgabe der Version 2. Adressieren Sie die Floppy folgendermassen:

APPARAT Inc.
NEWDOS/80 Diskette Update Service
4401 S. Tamarac Pkwy.
Denver, CO 80237
U. S. A.

Legen Sie bitte keine anderen Schriftstuecke mit bei. Wie schon im vergangenen Kapitel erwaehnt, duerfte dieser Update fuer Europa etwas anderst verlaufen, da die entstehenden Wartezeiten sehr gross sein duerften. Fragen Sie am besten bei Ihrem Haendler nach, ob er eine neue Diskette hat. Das Kopieren duerfte dann kein groesseres Problem mehr sein.

APPARAT wird, wenn Sie eine Floppy einschicken, eine komplette Kopie mit dem neuesten Stand machen, das heisst, dass auf der neuen Floppy alle Aenderungen enthalten sind. Innerhalb der USA wird die Floppy mit UPS versandt, ausserhalb per Post.

Die eingesandte Diskette muss das Orginalschild haben von APPARAT und noch Ihre Registrierungsnummer enthalten. Diese Registrierungsnummer wird mit der von Ihnen eingesandten Karte verglichen. Achten Sie darauf, dass nur Systemprogramme auf der Diskette sind und keine eigenen Programme, da diese geloescht werden. Die neue Diskette wird dann alle Aenderungen haben, die Systemprogramme betreffen. Nicht eingeschlossen sind Aenderungen an kaeuflichen Programmen wie z. B. SCRIPSIT, EDIT etc.

11.10 AENDERUNGSUEBERTRAG AUF ANDERE DISKETTEN

Alle Benutzer haben mehrere Kopien der Systemdiskette, Anwender mit nur einem Laufwerk muessen sogar auf jeder Diskette ein Betriebssystem haben. Wenn Sie einmal die Aenderungen auf eine Diskette gebracht haben und diese Aenderungen sind in Ordnung, haben Sie die Moeglichkeit, entweder diese Aenderungen auf allen Disketten durchzufuehren oder eine Aenderungsprozedur zu schreiben, mit der Sie die geaenderten Files auf eine andere Diskette kopieren. Mit der COPY-Funktion Format 6 und den Parametern CBF und DFO (hier beschrieben und nicht bei COPY, sowie als Aenderung in Kapitel 13) haben Sie dazu die Moeglichkeit. Folgendes Beispiel wird es Ihnen zeigen:

Nehmen wir an, dass die Files SYS0/SYS, SYS2/SYS, SYS17/SYS, SYS14/SYS, BASIC/CMD und DIRCHECK/CMD geaendert wurden und diese Files jetzt auf dem neuesten Stand sind. Ferner muessen diese Programme jetzt selbstverstaendlich in Ordnung sein.

Erstellen Sie jetzt auf dieser Diskette einen ILF-File (der genau wie der Chain-File aufgebaut ist) mit folgenden Zeilen:

```
SYS0/SYS
SYS2/SYS
SYS17/SYS
SYS12/SYS
```

```
BASIC/CMD
DIRCHECK/CMD
```

Nennen Sie diesen File ZAPNAMES/ILF. Erstellen Sie als Naechstes einen Chain-File mit einem der 2 folgenden Kommandos:

```
COPY,0,0,,NFMT,DFO,CBF,ILF=ZAPNAMES/ILF:0
```

fuer ein System mit einem Laufwerk, oder fuer ein System mit 2 Laufwerken das folgende Kommando:

```
COPY,0,0,,NFMT,DFO,CBF,ILF=ZAPNAMES/ILF:0
```

Nennen Sie diesen File ZAPDUP/JCL unspeichern Sie ihn auch auf der geaenderten Diskette ab. Benutzen Sie dafuer entweder das SCRIPSIT oder das BASIC-Programm CHAINBLD.

Die geaenderte Diskette ist gleichzeitig die Systemdiskette und die 'von'-Diskette und muss in Laufwerk 0 sein. Die Diskette, die die Aenderungen erhalten soll, kommt bei 2 Laufwerken in das Laufwerk 1.

Geben Sie jetzt fuer jede NEWDOS/80 Diskette, die geaendert werden soll, folgendes DOS-Kommando ein:

```
DO,ZAPDUP
```

Mit diesem DO-Kommando wird die COPY-Funktion aufrufen. Da in dem COPY-Kommando ILF eingegeben wurde, kopiert das System nur die Files, die in dem ILF-File angegeben sind. Weiter werden, bedingt durch die Option DFO nur solche Files kopiert, die es gleichzeitig auf beiden Disketten gibt. Wenn beispielsweise das Programm DIRCHECK/CMD auf der 'nach'-Diskette nicht vorhanden war, so wird es auch nicht kopiert.

Benutzer mit einem Laufwerk muessen sehr oft die Diskette im Laufwerk tauschen. Markieren Sie sich am Besten die geaenderte Diskette, damit es zu keinen Verwechslungen kommen kann.

Benutzer mit 2 Laufwerken muessen nur 2 Fragen des COPY beantworten pro Kopie.

In COPY Format 6 ist der Parameter DFO (Destination Files Only) nicht mit aufgefuehrt, in Kapitel 13 ist er nur kurz beschrieben, dafuer steht hier ein Beispiel.

12.0 KONVERTIERUNG UND ZUSAETZLICHE HINWEISE

In diesem Kapitel sind Informationen fuer die Umsetzung von Floppies von Version 1 auf Version 2 enthalten sowie andere zusaetzliche Hinweise. Ebenso sind hier Aenderungen aufgefuehrt von anderen Kapiteln, die zum Zeitpunkt der Aenderung bereits im Druck waren und deshalb nicht mehr auf den neuesten Stand gebracht werden konnten.

12.1 RBA

Im Juli hat man bei der Fa. APPARAT erfahren, dass das neue TRSDOS 1.3 fuer den TRS80 Modell III den Wert RBA benutzt, und zwar sowohl fuer das EOF-Feld im FPDE des Directories als auch fuer das NEXT- und EOF-Feld im FCB. Damit waeren endlich, nach 28 Monaten, die groessten Unterschiede (aber noch nicht alle) zwischen diesen 2 Betriebssystemen beseitigt.

Im Kapitel "FPDE" auf Seite 84 wurde das EOF-Feld im 4., 21. und 22. Byte des FPDE erlaeutert. In Kapitel "FCB" auf Seite 86 stehen die Erlaeuterungen fuer das EOF-Feld im FCB (6., 11. und 12. Byte).

In Kapitel "Unvertraeglichkeit NEWDOS/80 Version 2 mit TRSDOS 2.3" auf Seite 166 ist die Unvertraeglichkeit des NEWDOS/80 Version 2 mit dem TRSDOS Version 2.3 beim Modell I aufgefuehrt.

In Kapitel "Unvertraeglichkeit NEWDOS/80 Version 2 mit TRSDOS 1.3" auf Seite 167 ist die Unvertraeglichkeit des NEWDOS/80 Version 2 mit dem TRSDOS Version 1.3 beim Modell III aufgefuehrt.

Es war zwar der richtige Weg, dass das EOF-Feld im FPDE das RBA-Format erhalten hat, ungluecklicherweise entstand dadurch jedoch das Problem, dass Disketten in Version 1.1 und 1.2 im Modell III mit der Version 1.3 nicht direkt lesbar sind. Da man das Gefuehl hatte bei APPARAT, dass die neue Version 1.3 zum Standardsystem fuer das Modell III werden koennte, wurde das Kommando COPY zum Beispiel voll auf das neue Format ausgelegt. So aber koennen im Modell III nur Disketten kopiert werden, die mit dem TRSDOS 1.3 beschrieben wurden.

Als der RBA-Wert fuer das FCB-Feld NEXT und EOF im Maerz 1979 eingefuehrt wurde fuer das NEWDOS80, wollte man an sich auch den EOF-Wert im FPDE auf das neue Format umsetzen. Dadurch waeren aber alle Disketten, die mit einem anderen Betriebssystem beschrieben wurden, fuer das NEWDOS unbrauchbar geworden. Dies haette sicherlich eine zu grosse Einschraenkung fuer die Benutzbarkeit des NEWDOS bedeutet. Da es in jedem Fall momentan sehr wenige Programme gibt, die auf den EOF-Wert im Directory zugreifen, hat man beschlossen, diesen Wert im alten Format zu belassen. Da die Prozedur zur Konvertierung vom alten EOF-Format in das neue RBA-Format sehr einfach ist, kann es auch kaum zu Problemen kommen. Die Regeln sind:

- Konvertierung von NEWDOS/80 und TRSDOS auf das RBA-Format: Wenn das niedrigste Byte des 3 Byte langen Wertes ungleich 0 ist, werden 256 vom 3. Byte abgezogen (bezogen auf den 2 Byte langen Wert: minus 1).

- Konvertierung vom RBA-Format auf das NEWDOS- bzw. TRSDOS-Format: Ist das niedrigste Byte des 3 Byte langen RBA-Wertes ungleich 0, so werden 256 zum 3. Byte des RBA-Wertes hinzugezaehlt. Bezogen auf einen 2 Byte langen Wert: plus 1.

Obwohl es gewisse Anzeichen dafuer gibt, dass ein TRSDOS fuer das Modell I herauskommen soll, das mit dem von Modell III kompatibel ist, und damit das RBA-Format im EOF-Wert des FPDE

benutzt, wurde fuer das NEWDOS/80 Version 2 das alte Format fuer dieses Feld beibehalten.

12.2 KONVERTIERUNG VERSION 1 IN 2 MODELL I

1. Die meisten Programme, die im NEWDOS/80 Version 1 gelaufen sind, laufen auch ohne Aenderung in der Version 2.

2. Das Sperren und Freigeben der BREAK-Taste konnte nicht laenger ueber das Bit 4 der Adresse 4369H gemacht werden. Die Programme koennen dieses Bit zwar noch laenger aendern, das DOS ignoriert dies aber. Siehe dazu auch Kapitel "BREAK" auf Seite 10.

3. Aenderungen im FCB (siehe dazu Kapitel "FCB" auf Seite 86):

   a. Das Bit 2 im ersten Byte des FCB, mit dem Spur- und Sektoroperationen gekennzeichnet wurden, wird nicht mehr benutzt.

   b. Das Bit 3 des 2. Bytes des FBC's und die Bits 7 bis 5 des 3. Bytes haben jetzt eine Funktion.

   c. Die Bytes 17 bis 32 im FCB wurden neu definiert.

4. Aenderungen im Directory:

   Der GAT-Sektor enthaelt nun 'Lumps' anstelle von Spuren. Jedes Byte innerhalb des Bereichs von 00 bis 0BFH im GAT-Sektor deutet jetzt auf ein 'Lump' anstelle einer Spur, und Granule pro 'Lump' anstelle von Granule pro Spur. Das erste Byte eines Erweiterungselementes innerhalb eines FPDE's oder FXDE's ist jetzt die Nummer eines 'Lumps' anstelle einer Sektornummer. Vorausgesetzt, dass im PDRIVE-Kommando der richtige GPL-Wert eingesetzt ist, koennen alle Directories der Version 1 und das 3. Byte des BOOT-Sektors direkt von der Version 1 uebernommen werden.

   a. Die Bits 7, 6 und 5 des 2. Bytes des FPDE's werden jetzt benutzt.

   b. Die Belegungstabelle kann jetzt die ersten 192 Bytes des GAT-Sektors belegen. Wenn die Anzahl der 'Lumps' groesser ist als 96 (60H) reicht diese Tabelle in die Lockout-Tabelle hinein.

5. DEBUG kann nicht laenger ueber den Wert in 4315H aktiviert oder deaktiviert werden. Die Anwenderprogramme koennen zwar diesen Wert setzen, das DOS ignoriert dies aber.

6. Das DEBUG kann nicht laenger ueber die BREAK-Taste aufgerufen werden. Dies geht nur noch ueber die Tasten '123'.

7. Das Kommando PDRIVE wurde sehr stark veraendert. Lesen Sie sich das Kapitel "PDRIVE" auf Seite 34 gut durch. Eine der folgenden PDRIVE-Angaben muss benutzt werden, damit Disketten von der Version 1 gelesen werden koennen.

   a. PDRIVE,dn1,dn2,TI=A,TD=A,TC=35,SPT=10,TSR=3,GPL=2,DDSL=17,DDG

      ist die Angabe fuer die Standard-Diskette mit 5", einseitig beschrieben und einfache Schreibdichte. Fuer Disketten mit 40, 77 oder 80 Spuren veraendern Sie den Wert von TC entsprechend.

   b. PDRIVE,dn1,dn2,TI=A,TD=C,TC=80,SPT=20,TSR=3,GPL=4,DDSL=17,DDG

      ist die Angabe fuer eine Diskette mit 5", doppeltseitig beschrieben und einfache Schreibdichte. Fuer Disketten mit 40, 77 oder 80 Spuren veraendern Sie den Wert von TC entsprechend.

c. PDRIVE,dn1,dn2,TI=B,TD=B,TC=77,SPT=15,TSR=3,GPL=3,DDSL=17,DDG

ist die Angabe fuer eine Diskette mit 8", einseitig beschrieben und einfache Schreibdichte zusammen mit dem OMIKRON-Interface. In der Version 2 kann SPT bis zu 17 gross sein fuer diese Diskettenart. Wenn Sie Ihre Disketten auf dieses Format konvertieren, gewinnen Sie bis zu 12% mehr Platz.

d. PDRIVE,dn1,dn2,TI=CK,TD=E,TC=34,SPT=18,TSR=3,GPL=3,DDSL=17,DDC

ist die Angabe fuer eine Diskette mit 5", einseitig beschrieben und doppelte Schreibdichte zusammen mit dem PERCOM-Doubler. Fuer Disketten mit 39, 76 oder 79 Spuren veraendern Sie den Wert von TC entsprechend. Mit dem LNW-Interface versuchen Sie zuerst TI=EK. Wenn das nicht funktioniert, versuchen Sie es mit TI=CK.

e. Achtung!

5"-Disketten, die doppeltseitig sind und mit doppelter Schreibdichte in der Version 1 beschrieben wurden, koennen Sie in der Version 2 nicht benutzen. Die Files einer solchen Diskette muessen Sie unter der Version 1 auf eine einseitige Diskette mit doppelter Schreibdichte kopieren oder auf eine Diskette mit einfacher Schreibdichte, aber doppeltseitig beschrieben. Sobald Sie dies getan haben, koennen Sie unter der Version 2 diese Files auf eine doppeltseitige Diskette mit doppelter Schreibdichte kopieren.

8. 5"-Disketten mit doppelter Schreibdichte werden in der Version 2 fuer die Modifikation von PERCOM und LNW unterstuetzt.

9. Das Kommando SYSTEM wurde stark erweitert. Lesen Sie sich das Kapitel "SYSTEM" auf Seite 46 gut durch.

a. Die Optionen AH und AK sind entfallen. Die Optionen AT bis BN, ausser BL, sind neu hinzugekommen.

b. Mit der Option BN entscheidet das NEWDOS/80, ob die Directory-Sektoren im Modell I TRSDOS oder MODELL III NEWDOS/80 lesbar sind. Es geht jeweils nur eines von beiden.

c. Mit der Option BJ kann das NEWDOS/80 Schleifen fuer die Zeitverzoegerung entsprechend der CPU-Verschnellerung verlaengern. NEWDOS/80 kann damit mit fast allen CPU-Verschnellerungen arbeiten, die CPU darf aber nicht langsamer werden als 1.772 MHz.

10. Das Kommando COPY wurde ebenfalls wesentlich erweitert. Lesen Sie sich das Kapitel "COPY" auf Seite 12 gut durch.

a. CBF arbeitet, auch wenn die Systemdiskette herausgenommen werden musste oder wenn alle 3 Disketten das gleiche Laufwerk benutzen.

b. Wenn Sie im NEWDOS/80 eine Systemdiskette kopieren, muessen Sie die Option FMT angeben. Andernfalls kann es sein, dass die Information in BOOT/SYS und DIR/SYS falsch ist. Wenn Sie nur einige Systemfiles kopieren wollen, wenn die Diskette also schon vorher ein System enthalten hat, koennen Sie dies ohne den Parameter FMT machen. Diese Information fehlte in der Beschreibung von CBF.

c. Mit COPY koennen Sie Files zwischen NEWDOS/80 Version 2 und Modell III TRSDOS Version 1.3 hin- und herkopieren. Achten Sie auf die richtige PDRIVE-Angabe (Parameter TI, Flag M).

11. Die System-ID im NEWDOS/80 Version 2 wurde im Speicher von 403EH auf 4427H verlegt. In der Version 1 hat die Adresse

403EH entweder den Wert 80 (50H) oder 128 (80H) enthalten. In der Version 2 enthaelt die Adresse 4427H jetzt 130 (82H) als Kennzeichen fuer die Version 2, und die Adresse 442BH enthaelt 01 als Kennzeichen fuer das Modell I und 03 als Kennzeichen fuer das Modell III.

12. Keines der mit NEWDOS/80 Version 1 gelieferten Systemmoduls, BASIC-Moduls oder Programme koennen unter der Version 2 benutzt werden. Aus diesem Grunde muessen alle Anwenderfiles von der Diskette mit Version 1 auf eine Diskette mit Version 2 kopiert werden, ohne dass einer der genannten Systemfiles mitkopiert wird. Fuer Benutzer mit nur einem Laufwerk kann das eine ernorme Arbeit sein, aber auch Benutzer mit 2 oder mehr Laufwerken haben dabei viel zu tun. Benutzen Sie die folgende Prozedur fuer das Kopieren:

    a. Machen Sie eine Kopie einer bereits geaenderten NEWDOS/80 Version 2 Diskette (falls bereits Aenderungen angefallen sind). Machen Sie von dieser Diskette nochmals eine Kopie (zur Sicherheit !) mit dem COPY Format 5 oder 6 und der Option FMT.

    b. Loeschen Sie alle Files, die Sie nicht auf der Systemdiskette haben wollen. Sie koennen aber auch beim vorhergehenden COPY bereits den Parameter ILF benutzen, falls Sie im Format 6 kopieren. Als ILF-File koennen Sie den File NWD82V2/ILF von der gelieferten NEWDOS/80 Version 2 Diskette benutzen, den Sie vorher auf Ihre Beduerfnisse mit CHAINBLD/BAS oder SCRIPSIT anpassen. Schreiben Sie dann aber diesen File unter einem anderen Namen auf die Diskette zurueck. Diesen Namen geben Sie danach beim COPY als ILF-Parameter an.

    c. Diese neue Systemdiskette geben Sie als 'Nach'-Diskette beim COPY im Format 6 an. Die alte Version 1 Diskette, von der die Files kopiert werden sollen, ergibt die 'Von'-Diskette. Rufen Sie das COPY Format 6 mit den Parametern NFMT und XLF=NWD82V2/XLF auf. Mit diesem COPY werden dann alle Benutzerfiles von der alten Version 1 Diskette auf die Diskette mit Version 2 kopiert. Da alle Systemfiles in dem File NWD82V2/XLF aufgefuehrt sind, werden diese nicht kopiert. Der File NWD82V2/XLF befindet sich auch auf der gelieferten Systemdiskette und wurde extra aus diesem Grunde erstellt. Sie koennen ihn ueber CHAINBLD/BAS oder SCRIPSIT ansehen.

    d. Moechten Sie jetzt die neue Diskette mit Version 2 und Ihren Files auf die alte Diskette mit Version 1 kopieren, benutzen Sie das COPY Format 5 oder 6 mit der Option FMT.

12.3 KONVERTIERUNG VERSION 1 MODELL I IN VERSION 2 MODELL III

1. Die meisten Aussagen des vorstehenden Kapitels gelten auch hier. Lesen Sie sich also zuerst das letzte Kapitel gut durch. In diesem Abschnitt hier wird nur die Umsetzung auf Modell III besprochen.

2. Die meisten Anwenderprogramme, die im NEWDOS/80 Version 1 arbeiten, lassen sich mit den folgenden Korrekturen leicht auf Modell III Version 2 umstellen.

    a. Alle direkten Adressangaben, die sich auf den Bereich von 4300H bis 43FFH beziehen, muessen geaendert werden auf einen anderen Bereich. Der angegebene Adressbereich ist jetzt der Systembuffer, der in der Version 1 bei 4200H bis 42FFH lag.

    b. Alle Zugriffe auf die Adresse 4315H (DEBUG) muessen entfernt werden.

c. Das Aktivieren/Deaktivieren der BREAK-Taste wurde von 4312H verlegt nach 4478H. Der Zugriff auf das Bit 4 der Adresse 4369H muss entfernt werde.

d. Der Wert von HIMEM ist jetzt abgespeichert in 4411H und 4412H anstelle von 4049H und 404AH.

e. Der Wert von CLOCK ist jetzt abgespeichert in 4217H bis 4219H anstelle von 4041H bis 4043H.

f. Der Wert von DATE ist jetzt abgespeichert in 421AH bis 421CH anstelle von 4044H bis 4046H.

g. Der Zaehler der 25 msec - Intervalle wurde verlegt von 4040H nach 441FH. Die Interrupts fuer die Anwenderroutinen erfolgen im 25 msec Rythmus, auch wenn die eigentlichen Interrupts alle 1/30 sec oder 1/125 sec erfolgen.

h. Der Vektor auf 4410H, mit dem Interrupt-gesteuerte Routinen in die NEWDOS/80 Warteschlange gestellt wurden, hat sich geaendert auf 447BH.

i. Der Kommando-Buffer des NEWDOS/80 beginnt jetzt auf 4225H gegenueber 4318H vorher.

3. Die Directories der NEWDOS/80 Version 2 Diskette fuer Modell III sind im gleichen Format wie die des Modells I. Sie sind nicht kompatibel mit den Disketten von Modell III TRSDOS.

4. Die FCB's der NEWDOS/80 Version 2 Diskette fuer Modell III sind im gleichen Format wie die des Modells I. Sie sind nicht kompatibel mit den FCB's von Modell III TRSDOS.

5. Folgende PDRIVE-Angaben muessen gemacht werden, um bereits bestehende Disketten der Version 1 auf dem Modell III in der Version 2 lesen zu koennen:

   a. PDRIVE,dn1,dn2,TI=AK;TD=E;TC=39,SPT=18,TRS=3, GPL=2,DDSL=17,DDGA=2

      ist die Angabe fuer eine Diskette mit 5", einseitig, doppelte Schreibdichte und 40 Spuren. Fuer 35, 77 oder 80 Spuren setzen Sie TC auf 34, 76 oder 79.

   b. PDRIVE,dn1,dn2,TI=A;TD=A;TC=80,SPT=10,TRS=3, GPL=2,DDSL=17,DDGA=2

      ist die Angabe fuer eine Diskette mit 5", einseitig, einfache Schreibdichte und 80 Spuren. Fuer 35, 77 oder 80 Spuren setzen Sie TC auf den entsprechenden Wert.

   c. PDRIVE,dn1,dn2,TI=A;TD=C;TC=80,SPT=20,TRS=3, GPL=4,DDSL=17,DDGA=2

      ist die Angabe fuer eine Diskette mit 5", doppeltseitig, einfache Schreibdichte und 80 Spuren. Fuer 35, 77 oder 80 Spuren setzen Sie TC auf den entsprechenden Wert.

   d. Achtung!

      5"-Disketten mit doppelter Schreibdichte und doppeltseitig, die im NEWDOS/80 Version 1 benutzt wurde, koennen nicht direkt beim Modell III benutzt werden.

12.4 UNVERTRAEGLICHKEIT NEWDOS/80 VERSION 2 MIT TRSDOS 2.3

1. Das NEWDOS/80 behandelt das NEXT-Feld im FCB im RBA-Format. Im TRSDOS 2.3 wird das NEXT-Feld als RBA

behandelt, wenn das niedrigste Byte 0 ist oder wenn die momentane Position beim Schreiben innerhalb des Buffers liegt und veraendert wurde, aber der Buffer noch nicht zurueckgeschrieben wurde. In den meisten anderen Faellen behandelt das TRSDOS das NEXT-Feld gleich wie den RBA-Wert plus 256.

2. Das NEWDOS/80 behandelt das EOF-Feld im FCB im RBA-Format. Bei jeder Veraenderung des EOF-Wertes wird auch der EOF-Wert im FCB veraendert. Im TRSDOS 2.3 wird das EOF-Feld nur dann veraendert, wenn der Sektor zurueckgeschrieben wird. Das bedeutet, dass der EOF-Wert im FCB moeglicherweise bei einer Veraenderung von EOF 2 verschiedene Werte haben kann. Dies haengt davon ab, ob der momentane Sektor noch Daten erwartet oder ob er bereits geschrieben wurde. Normalerweise wird der EOF-Wert im FCB wie ein RBA-Wert dargestellt, wenn das niedrigste Byte 0 ist, oder als RBA-Wert plus 256, wenn das niedrigste Byte ungleich 0 ist.

3. Die Aktivierung oder Deaktivierung von DEBUG im TRSDOS wird noch ueber das Byte auf der Adresse 4315H gemacht, was aber beim NEWDOS/80 Modell I ignoriert wird und nicht getan werden darf im NEWDOS/80 Modell III.

4. Die Aktivierung oder Deaktivierung von der Timer-Routinen wird in beiden Systemen unterschiedlich gehandhabt. Beachten Sie das Kapitel 3 in diesem Handbuch.

5. Das TRSDOS und das NEWDOS/80 benutzen die gleiche Art von Directory, ausser dass das TRSDOS limitiert ist auf 35 Spuren und die Directorygroesse auf 2 Granules. Ferner benutzt das NEWDOS/80 einige freie Bytes und Bits.

6. Nachfolgend eine Liste der benutzbaren Systemroutinen aus Kapitel 3, die sowohl im TRSDOS als auch im NEWDOS/80 zur Verfuegung stehen. Alle Routinen haben fast die gleiche Funktion in beiden Systemen. Die nicht aufgefuehrten Routinen aus Kapitel 3 werden entweder nicht benutzt im TRSDOS oder haben eine andere Funktion.

   0013H, 001BH, 402Dh, 4030H, 4400H, 4405H, 4409H, 440DH, 441CH, 4420H, 4424H, 4428H, 442CH, 4430H, 4433H, 4436H, 4439H, 443CH, 443FH, 4442H, 4445H, 4448H, 4467H, 446AH, 446DH, 4470H, 4473

12.5 UNVERTRAEGLICHKEIT NEWDOS/80 VERSION 2 MIT TRSDOS 1.3

1. Disketten, die mit dem TRSDOS Modell III beschrieben wurden, sind nicht kompatibel mit dem NEWDOS/80 Version 2. Disketten mit 5", einfache Schreibdichte, einseitig beschrieben, 35 Spuren und 2 Granules fuer das Directory, das auf Spur 17 beginnen muss, koennen mit dem Konvertierungsprogramm umgesetzt werden. Ebenfalls koennen Files zwischen TRSDOS Version 1.3 Modell III und NEWDOS/80 Version 2 hin- und herkopiert werden, wenn im NEWDOS der richtige Parameter bei PDRIVE angegeben wird (TI Flag M).

   Siehe auch dazu Aenderung $030.

2. Da inzwischen das TRSDOS 1.3 den RBA-Wert im NEXT- und EOF-Feld des FCB's und im EOF-Feld des Directories benutzt, ist einer der Hauptunterschiede zwischen diesen 2 Betriebssystemen beseitigt.

3. Das TRSDOS Modell III Version 1.3 benutzt fuer den FCB einen Buffer von 50 Bytes, NEWDOS/80 dagegen noch die alte Buffergroesse von 32 Bytes. Der Anwender sollte selbst die Unterschiede in der Benutzung des FCB's durchlesen, da hier keine detailierten Angaben gemacht werden.

4. Das Byte, das fuer die Aktivierung und Deaktivierung der BREAK-Taste herangezogen wird, steht bei TRSDOS Modell III auf der Adresse 42EAH, beim NEWDOS/80 Modell III auf 4478H, und beim NEWDOS/80 Modell I auf 4312H. Hat dieses Byte den Wert 0C9H, so ist die Taste aktiviert, bei 0C3H ist sie deaktiviert.

5. Nachfolgend eine Liste der benutzbaren Systemroutinen aus Kapitel 3, die sowohl im TRSDOS als auch im NEWDOS/80 zur Verfuegung stehen. Alle Routinen haben fast die gleiche Funktion in beiden Systemen. Die nicht aufgefuehrten Routinen aus Kapitel 3 werden entweder nicht benutzt im TRSDOS oder haben eine andere Funktion.

   0013H, 001BH, 402DH, 4030H, 4409H, 440DH, 441CH, 4420H, 4424H, 4428H, 442CH, 4430H, 4433H, 4436H, 4439H, 443FH, 4442H, 4445H, 4448H

6. In Kapitel "CMD"x"" auf Seite 117 stehen die verschiedenen Funktionen des Statements CMD im Basic, die im NEWDOS unterstuetzt werden.

7. Die Umleitung von Ein- und Ausgabe ist unterschiedlich in den 2 Systemen. DUAL gibt es nicht im NEWDOS/80, kann jedoch ueber ROUTE erreicht werden.

12.6 ZUSAETZLICHE HINWEISE

1. Einige Benutzer haben sich selbst Systemroutinen geschrieben, die vom System-Loader geladen werden. Diese Benutzer sollten vorsichtig sein, da das NEWDOS/80 Version 2 die FPDE's bis SYS21/SYS benutzt. Das NEWDOS/21 und das TRSDOS konnten nur 14 Systemprogramme laden, der Loader im NEWDOS/80 akzeptiert jedoch bis zu 30 fest zugeordnete Systemprogramme. Der Code, um eine dieser Routinen in einer bestimmten Position des Directories aufzurufen, muss im A-Register stehen, und groesser sein als 1FH, im folgenden uuuubsss- Format:

   sss+2 Relativer Sektor im Directory, in dem der FDE steht.

   bb Dieser Wert mal 32 (20H) ist das Offset des FDE's im Sektor

   uuu Ein Benutzercode groesser 0

2. Alle mit dem NEWDOS zusammen gelieferten Programme benutzen den in HIMEM abgespeicherten Wert als oberes Limit fuer den Speicherbereich.

3. (nur Modell I) Bei Power On, Reset oder einem Sprung auf die Adresse 0 wird im ROM der Status der Diskette auf der Adresse 37ECH geprueft um herauszufinden, ob der Disk - Controller installiert ist oder nicht. Ist dieser Status entweder 0 oder 0FFH nimmt das ROM an, dass keine Floppy angeschlossen ist und startet das normale Level II BASIC. Da 0 jedoch auch ein zulaessiger Status der Floppy ist (Drive Ready), der dann gesetzt ist, wenn das Laufwerk in Betrieb ist (Motor laeuft, rote Lampe ist an), kann es zum unkontrolliertem Start des Level II BASIC kommen. Um dies zu verhindern, sollte man warten, bis das Laufwerk stoppt, d. h. die rote LED ausgeht, bevor man ein Reset oder einen Sprung auf die Adresse 0 ausfuehrt.

4. In aelteren NEWDOS - Versionen wurde fuer einen File immer nur ein Granule reserviert, dadurch konnte man, im Gegensatz zum TRSDOS, auch das letzte Granule noch belegen. Im NEWDOS/80 wurde das wieder geaendert auf 4 granules, um die Geschwindigkeit der Floppy zu erhoehen. Es kann also vor-

kommen, dass in einem Programm der Fehler 'OUT OF SPACE' gemeldet wird, nach einem Close aller Dateien genuegend Platz, der nur reserviert war aber noch nicht benutzt wurde, freigegeben wird.

5. Im NEWDOS/80 wird momentan noch nicht ueberprueft, ob die maximale Spurzahl erreicht ist, wenn der Arm der Floppy bewegt wird. Wenn die Spurzahl zu hoch ist, stoesst der Arm an der Begrenzung an und wird dies so oft wiederholen, wie als Zaehler bei SYSTEM angegeben wurde. Dies kann unter Umstaenden Minuten gehen. Um dies zu umgehen, koennen Sie, wenn dies auftritt, einfach das Laufwerk oeffnen und warten, bis entweder der Motor stehenbleibt oder der Fehler gemeldet wurde.

6. Die Einzelschrittausfuehrung im BASIC (CMD"F=SS") erlaubt keine Zeitkritischen Befehle wie z. B. INKEY$. Ebenso funktioniert die Anzeige nicht, wenn das Display sich im 32-Character Mode befindet.

7. FORMAT-Korrektur. Der Parameter PFST darf nur benutzt werden, wenn nicht Y oder N eingegeben wird.

8. COPY-Korrektur. Wenn im COPY Format 6 (CBF) das NEWDOS/80 - System auf eine neue Floppy kopiert werden soll, muss der Parameter FMT mit angegeben werden. Dies kommt daher, dass die Systemfiles auf einem festen zugeordneten Platz stehen muessen und im BOOT/SYS die richtige Information stehen soll. Dieses COPY muss immer benutzt werden, wenn die PDRIVE-Angaben unterschiedlich sind von der 'Von'-Diskette.

13.0 DIE AENDERUNGEN IM NEWDOS/80

Lesen Sie sich das Kapitel "Fehlerrueckmeldung und Durchfuehrung von Aenderungen" auf Seite 155 der NEWDOS/80-Beschreibung sorgfaeltig durch, darin ist die Anleitung zur Durchfuehrung der hier folgenden Aenderungen beschrieben.

Einige Versionen des NEWDOS/80 haben einen anderen Stand als hier angegeben. Aendern Sie nur die Punkte, die komplett mit der hier beschriebenen 'von'-Zeile uebereinstimmen. Stimmt die Version, die Sie haben, weder mit der 'von'- noch mit der 'in'-Zeile ueberein, wenden Sie sich bitte an Ihren Haendler oder direkt an APPARAT. Falls APPARAT nicht Ihre spezielle Version vorliegen hat werden Sie gebeten, eine Diskette mit Ihrem Betriebssystem einzuschicken. APPARAT wird diese Diskette behalten!

********** Aenderung #001 ********** 04.08.81 **********

Die erste Aenderung betrifft den Editor EDIT/CMD von MICROSOFT. Sie muessen in jedem Falle mehrere Kopien fuer alle unterschiedlichen Betriebssysteme haben, da sie untereinander nicht kompatibel sind. Diese Aenderung betrifft die unterschiedliche Handhabung der Werte NEXT und EOF im FCB.

EDIT/CMD,38,E8 aendern Sie

```
von    7E E6 7F 7E 21 FE FF CA 58 78 EE 80 87 DA 5D 78
in     7E 60 69 D6 01 30 01 2B 29 07 30 01 23 18 04 78
```

EDIT/CMD,32,A2

```
von    78 E6 7F C2 28 72 78 EE
in     78 D6 01 30 08 18 04 EE
```

********** Aenderung #002 ********** 04.08.81 **********

Diese Aenderung betrifft nur Anwender des Programmes PENCIL/CMD. PENCIL veraendert eine Adresse im DOS, was aber weder im NEWDOS/80 noch in anderen NEWDOS-Versionen notwendig ist

PENCIL/CMD,00,AE aendern Sie

```
von    F3 32 9B 46 C3
in     F3 00 00 00 C3
```

Mit folgenden Aenderungen kann das Programm PENCIL/CMD das Directory richtig lesen und das MINI-DOS kann darin gestartet werden.

PENCIL/CMD,05,60 aendern Sie von 58 23 22 in 58 00 22

```
PENCIL/CMD,00,61 aendern Sie von 54 E2 16 40 21
                              in 54 00 00 00 21


SYS0/SYS,12,10 aendern Sie von 00 00 00 00 00 00 00
                              in 0F 0F 0F C6 04 6F C9
```

Mit dieser Aenderung wird erreicht, dass das Programm PENCIL/CMD nicht seine eigene Tastatur-Routine benutzt, sondern die von NEWDOS/80. Aus diesem Grunde kann dann auch 'DFG' benutzt werden.

************* Aenderung #003 *********** 04.08.81 *************

Folgende Aenderungen sind in den Programmen SCRIPSIT/UC und SCRIPSIT/LC noetig, damit sie unter NEWDOS/80 laufen. Da sie mit diesen Aenderungen nicht mehr im TRSDOS und anderen NEWDOS Versionen lauffaehig sind, machen Sie bitte vorher Kopien davon.

Die ersten drei Aenderungen sind noetig, da das NEWDOS/80 die Werte NEXT und EOF im FCB anders behandelt als das TRSDOS. Wenn Sie unter NEWDOS/80 den Fehler feststellen, dass der SCRIPSIT-File einen Sektor zu kurz geladen wird, ueberpruefen Sie, ob diese Aenderungen gemacht wurden. Die vierte Aenderung betrifft den Wert HIMEM in 4049H und 404AH, der so in SCRIPSIT benutzt wird. Die fuenfte Aenderung schaltet die Interrupts wieder ein, es kann dadurch MINI-DOS auch im SCRIPSIT aufgerufen werden.

```
SCRIPSIT/UC,11,75   und   SCRIPSIT/LC,11,75   aendern Sie

von   47 00 CD 6E 7A 4F
in    47 00 3A B9 7C 4F


SCRIPSIT/UC,11,FB   und   SCRIPSIT/LC,11,FB   aendern Sie

von   B7 C4 EF 5D 79
in    B7 32 B6 7C C4


SCRIPSIT/UC,12,00   und   SCRIPSIT/LC,12,00   aendern Sie

von   32 B9 7C 11
in    EF 5D 00 11


SCRIPSIT/UC,00,63   und   SCRIPSIT/LC,00,63   aendern Sie

von   7C 21 FF 00 25 7E 2F 77 AE 20 F9 22
in    7C 2A 49 40 00 00 00 00 00 00 00 22


SCRIPSIT/UC,00,C3   und   SCRIPSIT/LC,00,C3   aendern Sie

von   57 F3 ED
in    57 00 ED
```

********** Aenderung #004 ********** 04.08.81 **********

NEWDOS/80 wartet nur eine halbe Sekunde beim Ansprechen der Floppies, bis der Motor hochgelaufen ist. Sollten Sie das Gefuehl haben, dass es mit einer Sekunde Wartezeit weniger Fehler gibt, geben Sie folgende Aenderung ein:

SYS0/SYS,04,C3 aendern Sie von 06 80 CB in 06 FF CB

********** Aenderung #005 ********** 04.08.81 **********

Folgende Aenderungen sind im Programm APL80/CMD noetig, damit es unter NEWDOS/80 laeuft. Diese Aenderung betrifft eine Routine, die im NEWDOS/80 eine andere Position hat als im TRSDOS oder anderen NEWDOS-Versionen.

APL80/CMD,12,74 aendern Sie von 21 A2 4B in 21 C5 4B

APL80/CMD,14,52 aendern Sie von 21 A2 4B in 21 C5 4B

********** Aenderung #006 ********** 04.08.81 **********

Folgende Aenderungen koennen im Programm EDTASM/CMD gemacht werden, damit es sowohl Klein- als auch Grossbuchstaben akzeptiert. Die meisten Funktionen im EDTASM benoetigen zwar nach wie vor Grossbuchstaben, es koennen aber die Befehle mit Hochkomma (DEFM und DEFB) Kleinbuchstaben enthalten.

EDTASM/CMD,27,F4 aendern Sie von FE 61 D8 in FE 80 D8

********** Aenderung #007 ********** 04.08.81 **********

Dies ist keine Aenderung, sondern nur eine Information.

1. Wo es moeglich war, haben NEWDOS/80 Module Patchbereiche, auf die der Anwender keinen Einfluss haben soll. Diese Bereiche wurden hier in dieser Dokumentation nicht besonders erwaehnt. Wenn nun von APPARAT Aenderungen fuer Systemprogramme bekanntgegeben werden, so liegen die neuen Bereiche oder Erweiterungen in diesem Patchbereich. Dazu ist aber Voraussetzung, dass diese Bereiche nicht von anderer Stelle benutzt werden, sie muessen also auf 0 stehen. Wenn ein Anwender eigene Systemroutinen schreibt und in diese Patchbereiche legt, sollte er sorgfaeltig Buch fuehren darueber.

2. Das Kommando ATTRIB (siehe "ATTRIB" auf Seite 7) wurde um einen weiteren Parameter erweitert, der nicht in diesem

Kapitel aufgefuehrt ist. Mit LRL kann man die logische Recordlaenge eingeben. LRL=xxx ist der optionale Parameter, wobei xxx eine gerade Zahl zwischen 1 und 256 ist. Diese Recordlaenge wird nur im Directory eingetragen, jedoch von keinem Systemprogramm benutzt.

3. MINI-DOS (siehe "MINI-DOS" auf Seite 69) kann nicht im DOS-CALL-Status (siehe "DOS-CALL" auf Seite 74) aufgerufen werden. Werden in dieser Zeit die Tasten 'DFG' gedrueckt, so wird dies ignoriert.

4. Das Kommando ROUTE (siehe "ROUTE" auf Seite 43) wurde so veraendert, dass die gerade aktuellen Parameter nur dann angezeigt werden, wenn mit dem Kommando keine Parameter angegeben wurden.

5. Das Kommando COPY (siehe "COPY" auf Seite 12), Format COPY 6 (siehe "COPY 6" auf Seite 16) hat einen neuen, zusaetzlichen Parameter erhalten. Wird DFO eingegeben (Destination File Only), so werden nur Files kopiert, die auf beiden Disketten vorhanden sind. DFO kann nicht zusammen mit FMT benutzt werden.

6. Das Kommando FORMAT (siehe "FORMAT" auf Seite 26) hat den zusaetzlichen optionalen Parameter RWF erhalten (RaW Format). Wenn dieser Parameter angegeben ist, werden alle auftretenden Fehler ignoriert. Jede Spur wird nur einmal formatiert, egal ob die Daten stimmen oder nicht. RWF wird benutzt, wenn eine defekte Diskette vollkommen geloescht werden soll, und wenn der Anwender dazu keinen Magneten benutzt. RWF kann nicht benutzt werden mit KDN, KDD, DDSL, DDGA und PFST.

********** Aenderung #008 ********** 04.08.81 **********

Mit folgenden Aenderungen beseitigen Sie einen Fehler im PROFILE-Modul INIT. Diese Aenderung bewirkt, dass der Fehlersatus NZ gegenueber C oder NC Prioritaet hat im NEWDOS bei einem RETURN aus einem CALL von 4420H. Nach dieser Aenderung laeuft das Programm nach wie vor unter TRSDOS.

INIT,00,54 aendern Sie

```
von   44 38 09 28 0F FE 1A 28 13 C3 3F 52 CD
in    44 20 19 00 00 00 00 00 00 00 30 08 CD
```

Falls Sie daran interessiert sind, koennen Sie auch die Interrupts wieder aktivieren, was eine Benutzung von '123' und 'DFG' im programm PROFILE unter NEWDOS erlaubt.

PROFILE/CMD,11,45 aendern Sie F3 CD in 00 CD

********** Aenderung #009 ********** 04.08.81 **********

Mit folgenden Aenderungen koennen Sie mit dem BASIC-Modul IBLOAD/CMD von Racet sowohl unter TRSDOS als auch unter NEWDOS arbeiten.

IBLOAD/CMD,08,15 aendern Sie

```
von   15 21 B6 79 11 F4 79 CD 36 44 28
in    15 11 F4 79 CD 13 00 32 B6 79 28
```

********** Aenderung #010 ********** 04.08.81 **********

Die folgende Aenderung muss im DSM-Modul DSMB/CMD von Racet gemacht werden, da der Filetyp maximal 3 Zeichen haben darf und mit einem Endekennzeichen aufhoeren muss (03H). Diese Korrektur ist fuer den Filenamen DSMC/CMDPATCHPATCHPATCH fuer den Fall, dass ein Benutzer einen anderen Namen eingibt mit 03H am Ende.

DSMB/CMD,03,C2 aendern Sie von 44 50 41 in 44 03 41

********** Aenderung #011 ********** 04.08.81 **********

Folgende Aenderung muss gemacht werden, damit das Programm VISICALC auch unter NEWDOS arbeitet. Das geaenderte Programm kann nicht mehr unter TRSDOS laufen, d. h., Sie muessen in jedem Falle 2 Versionen haben. Diese Aenderung betrifft die unterschiedliche Behandlung der 25ms Interrupt Routinen (siehe Kap. 3.7 und 1.6.1).

VC/CMD,03,2B aendern Sie

```
von   09 3E 00 21 20 03 22 51 9F C9 79
in    09 18 BB 00 00 08 08 C3 EF 9B 79
```

VC/CMD,75,15 aendern Sie von 11 28 9C 22 in 11 1E 55 22

VC/CMD,75,24 aendern Sie

```
von   C9 3E 03 C3 13 44 CD 4E 53 F5 CD 16 9C 28 0E 3E
in    C9 11 1E 55 C3 13 44 CD 4E 53 CD 16 9C C8 00 3E
```

Mit folgender Aenderung im VISICALC wird der Unterschied zwischen NEWDOS/80 und TRSDOS im Returncode bei der Suche im Directory beseitigt. Vorher konnt es passieren, dass das Programm bei der Suche im Directory haengen blieb, wenn eines der 4 Laufwerke nicht READY war oder keine Diskette hatte. Mit dieser Aenderung kann VISICALC auch noch mit dem TRSDOS arbeiten.

VC/CMD,73,01 aendern Sie von C9 FE 18 20 in C9 37 C9 20

********** Aenderung #012 ********** 04.08.81 **********

Optionale Aenderung zum Beeinflussen der Empfindlichkeit der Floppy beim Formatieren. Sie koennen unter 3 verschiedenen Bitmustern aussuchen, abhaengig von der Empfindlichkeit der Floppy und des Interfaces. Je empfindlicher das Bitmuster, das auf die Diskette geschrieben wird, umso groesser wird die Moeg-

lichkeit eines Fehlers beim Formatieren. Es kann natuerlich sein, dass eine Diskette beim Formatieren trotz empfindlichem Bitmuster kein Fehler bringt, dafuer aber spaeter beim Lesen oder umgekehrt. Die Wahrscheinlichkeit aber, dass eine Diskette, die mit einem empfindlichen Bitmuster ohne Fehler formatiert wurde, spaeter einen Lese- oder Schreibfehler bekommt, ist gering.

Die moeglichen Bitmuster sind:

E5 E5 Das am wenigsten empfindliche Bitmuster. Es ist das Standard- Bitmuster bei Floppies mit einfacher Schreibdichte.

5B 5B Bitmuster mit mittlerer Empfindlichkeit. Standard-muster fuer das Modell III TRSDOS.

6D B6 Am empfindlichstes Bitmuster fuer das Formatieren. Falls die Floppies, das Interface oder das Laufwerk nicht zu 100% in Ordnung sind, koennen ca. 30% der Floppies nicht formatiert werden.

Entsprechend der Anzahl von Formatierungsfehlern und anderen Fehlern an der Diskette kann der Anwender jetzt entscheiden, welches dieser 3 Bitmuster er einsetzen will. Ueberpruefen Sie zuerst den angegebenen Speicherplatz, ob auch tatsaechlich eines der angegeben Muster eingesetzt ist. Es werden beide Bytes eingegeben. Das nachfolgende Byte muss F5 sein.

SYS6/SYS,31,D9

SYS6/SYS,31,F3

********** Aenderung #013 ********** 04.08.81 **********

Optionale Aenderung, um auch Modell III - Disketten vom TRSDOS 1.2 oder frueher kopieren zu koennen, anstatt Disketten mit dem Directoryformat von TRSDOS 1.3. Falls Sie spaeter nochmals andere Kopien von TRSDOS 1.3 machen wollen, muessen Sie zumindest die Orginalversion sichern.

SYS6/SYS,14,75 aendern Sie 01 00 7B in 01 01 7B

SYS6/SYS,14,C8 aendern Sie

von 01 00 40 4E 01 13 00
in 01 13 40 4E 01 00 00

SYS6/SYS,20,EA aendern Sie 61 C8 5E in 61 C9 5E

********** Aenderung #014 ********** 10.08.81 **********

Mit dieser Aenderung kann der Fehler im COPY Format 5 nicht mehr auftreten, wenn im BOOT-Sektor nicht der richtige Wert fuer den 'Lump' stand, in dem das Directory anfaengt bzw. der Wert nicht mit der PDRIVE-Information uebereinstimmt. Es wurde dann ein Directory - Lesefehler gemeldet.

SYS6/SYS,05,96 aendern Sie 64 01 00 01 18 in 64 CD 80 5C 18

SYS6/SYS,15,46 aendern Sie

```
von  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
in   01 00 01 B7 C0 13 13 1A 32 BC 64 1B 1B C9
```

********** Aenderung #015 ********** 10.08.81 **********

Mit dieser Aenderung beseitigt man den Fehler SYSTEM PROGRAM NOT FOUND, der dann auftrat, wenn als Laufwerksnummer ein Wert von groesser 9 angegeben wurde.

SYS16/SYS,00,6A aendern Sie C3 1A 52 CB in C3 3C 50 CB

********** Aenderung #016 ********** 10.08.81 **********

Mit dieser Aenderung beseitigt man einen Fehler in COPY und FORMAT. Der Parameter DDSL wurde nicht angenommen.

SYS6/SYS,01,FA aendern Sie CB 70 20 in CB 52 20

********** Aenderung #017 ********** 25.08.81 **********

Mit dieser Aenderung beseitigt man einen Fehler im SUPERZAP. Bei der Funktion DM,P wurden zeitweise falsche Daten an den Drucker ausgegeben.

SUPERZAP/CMD,00,04 aendern Sie

```
von  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
in   3A 91 54 CB 5F C0 2A B4 54 22 95 54 C9
```

SUPERZAP/CMD,03,43 aendern Sie 54 3A 91 54 CB
                            in 54 CD 00 52 CB

********** Aenderung #018 ********** 25.08.81 **********

Mit dieser Aenderung beseitigen Sie einen Fehler in CHAINBLD/BAS. Wenn ein File im NEWDOS/80 Format Version 1 auf der Diskette abgespeichert wurde und die letzte Zeile mit /./ geendet hat, wurde der Fehler NEXT WITHOUT FOR gemeldet.

Fuegen Sie an die Zeile 126 von CHAINBLD/BAS folgendes Statement an: :GOTO130

********** Aenderung #019 ********** 25.08.81 **********

Diese Aenderung ist die Fortsetzung der Aenderung Nr. 002.

SCRIPSIT/UC,00,28 und SCRIPSIT/LC,00,28 aendern Sie

von
```
                                  56 49 53 49 4F 4E 20 4F
46 20 54 41 4E 44 59 20 43 4F 52 50 4F 52 41 54
49 4F 4E 31
```

in
```
                                  56 20 4F 46 20 54 41 4E
44 59 20 43 4F 52 50 3A 80 38 CB 4F C8 F6 01 C9
00 00 00 31
```

SCRIPSIT/UC,15,08 und SCRIPSIT/LC,15,08 aendern Sie

von 0B 3A 80 38 FD in 0B CD 33 52 FD

SCRIPSIT/UC,15,34 und SCRIPSIT/LC,15,34 aendern Sie

von 0D 3A 80 38 47 in 0D CD 33 52 47

********** Aenderung #020 ********** 28.08.81 **********

Mit dieser Aenderung im DOS erreichen Sie, dass bei der Umleitung der Ausgabe auf den Drucker im BASIC das PRINT die numerischen Werte nicht doppelt ausgibt.

SYS0/SYS,07,61 aendern Sie von 28 39 CD in 28 5D CD

SYS0/SYS aendern Sie

von
```
                                  C9 23 7E 23 66 5F
7C B5 C8 7E BB 23 20 02 7E BA 23 20 EE E5 D5 C5
7E A0 23 23 23 5E 23 56 D5 DD E1 7A B3 C4 8B 06
C1 D1 E1 CB 40 28 D4 B7 28 D1 C9 CD
```

in
```
                                  C9 7E BB 23 20 02
7E BA 23 20 1A E5 D5 C5 7E A0 23 23 23 5E 23 56
D5 DD E1 CD 8B 06 C1 D1 E1 CB 40 28 02 B7 C0 23
7E 23 66 6F 7C B5 20 D3 C3 9D 4C CD
```

SYS0/SYS,09,7F aendern Sie

```
von  C9 00 00 00 00 00 00 00 00
in   C9 CB 40 C0 79 C9 49 4B 00
```

SYS0/SYS,02,86 aendern Sie von 43 00 00 43 in 43 A1 4C 43

**** Aenderung #021 **** 28.08.81 ****

Mit dieser Aenderung im BASIC erreichen Sie, dass Marked Item und Fixed Item Files korrekt geschrieben werden, wenn die Recordlaenge groesser als 127 ist.

```
BASIC/CMD,10,F1   aendern Sie von  CD BA 61 79 DD
                               in  CD A8 65 00 DD


BASIC/CMD,14,B1   aendern Sie von  00 00 00 00 00 00 00
                               in  C5 CD BA 61 79 C1 C9
```

**** Aenderung #022 **** 28.08.81 ****

Mit dieser Aenderung im EDTASM wird erreicht, dass immer dann der Cursor ausgegeben wird, sobald der '*' angezeigt wird und das Programm auf eine Eingabe wartet. Damit wird auch dann der Cursor angezeigt, wenn vom DEBUG zurueckgesprungen wird in die Kommandoeingabe des EDTASM.

```
EDTASM/CMD,05,12  aendern Sie von  CD 39 59 0E
                               in  CD 00 57 0E


EDTASM/CMD,03,1A  aendern Sie

von  40 7D E6 3F C0 11 C0 FF 19 C9
in   40 CD 39 59 3E 0E C3 39 59 C9
```

**** Aenderung #023 **** 28.08.81 ****

Mit dieser Aenderung im DOS werden einige Probleme beseitigt, die dadurch entstanden, dass im NEWDOS/80 Version 1 eine Sperrtabelle im GAT-Sektor benutzt hat, die Version 2 diesen Teil des GAT-Sektors jedoch als Erweiterung der Belgt-Tabelle benutzt. Beim Beginn der Entwicklung von NEWDOS/80 Version 2 wurde die Regel aufgestellt, dass wenn im PDRIVE der Parameter GPL groesser als 2 ist, die Sperrtabelle nicht benutzt wird. Spaeter wurde es in der Art geaendert, dass wenn die Anzahl der 'Lumps' groesser ist als 96 (60H) oder wenn das relative Byte im GAT-Sektor 60H gleich 0FFH ist, die Sperrtabelle nicht benutzt wird.

Einige Teile des Codes fuer SYS6/SYS wurden nicht der neuen Vereinbarung angepasst, was bewirkt hat, dass entweder die Sperrtabelle nicht angelegt wurde oder nicht richtig erweitert wurde, wenn die 'Nach'-Diskette mehr Granules hatte als die 'Von'-Diskette.

Passen Sie also auf, wenn Sie vorher Disketten mit dem Parameter GPL von groesser als 2 kopiert haben (bei 8"-Disketten oder 5"-Disketten von der Version 1). Im DIRCHECK kann es dabei vorkommen, dass ein oder mehrere Granules als gesperrt gemeldet werden, die aber frei sind und eventuell bereits einem File zugeordnet sind. Eine solche Diskette kann weiterhin benutzt

werden, da der Fehler in diesem Fall beim DIRCHECK liegt. Kopieren Sie aber in jedem Fall frueher oder spaeter eine solche Diskette mit den Parametern CBF und FMT. Damit wird auch der Fehler, den DIRCHECK meldet, behoben sein.

SYS6/SYS,10,0A aendern Sie

von 3A CA 59 FE 02 C0 7D
in   3A C6 59 FE 61 D0 7D

********** Aenderung #024 ********** 30.08.81 **********

Aenderung im DOS fuer:

1. das COPY Format 5 und 6. Dabei wurde bei einem Verify-Error der Fehler END OF FILE ENCOUNTERED oder INPUT PAST END fuer den naechsten Sektor gemeldet.

2. das FORMAT. Der Parameter BDU wird zugelassen und im COPY Format 5 mit den Parametern FMT und BDU kann die 'Nach'-Diskette formatiert werden, auch wenn der Parameter TI von PDRIVE mit M angegeben wurde (gilt fuer Modell III im TRSDOS). Mit der BDU-Option werden keine Kontrollinformationen vom NEWDOS/80 auf die Diskette geschrieben beim FORMAT, dadurch kann eine Modell III - Diskette ueber COPY Format

   Achtung, Die neue Diskette kann nicht mehr im NEWDOS/80 benutzt werden als 'Von'-Diskette.

SYS6/SYS,07,1E aendern Sie

von C6 22 1A 5B 3E
in   C6 CD 8E 5C 3E

SYS6/SYS,15,54 aendern Sie

von 00 00 00 00 00 00 00 00
in   22 1A 5B DD CB 01 F6 C9

SYS6/SYS,27,6A aendern Sie von 6F C2 47 67 in 6F C4 51 67

********** Aenderung #025 ********** 31.08.81 **********

Aenderung im DOS, um unnoetig viele Schreibversuche zu vermeiden beim Fehler LOST DATA. Normalerweise arbeitet der Wiederholungszaehler, aber jedes Mal ging eine Diskettenumdehung (200 ms) verloren.

SYS0/SYS,02,BD aendern Sie

von C5 06 A2 28 03 04 3E 08 32 E1 46 78 32 AC 46 32 B1 46
in   C5 01 02 A2 28 05 01 97 A3 3E 08 32 E1 46 79 CD ab 46

SYS0/SYS,03,62 aendern Sie

```
von         02 3A 87 46 CB 6F 28 07 DB F0 A2 28 FB ED
      A3 DB F0 A2 28 FB ED A2 7B D3 F4 ED A2 20 FA 18
      FE AF

in          02 DB F0 A2 28 FB ED A2 10 FE 05 7B D3 F4
      ED A2 20 FA 18 FE 32 8F 46 78 32 9E 46 32 A6 46
      C9 AF
```

SYS6/SYS,15,5C aendern Sie

```
von   00 00 00 00 00 00 00 00 00
in    32 A6 46 3E 02 32 8F 46 C9
```

SYS6/SYS,30,C9 aendern Sie

```
von   3A F4 45 32 AC 46 32 B1 46 21
in    3A F5 45 32 9E 46 CD 96 5C 21
```

********** Aenderung #026 ********** 01.09.81 **********

Diese Aenderung ist die Erweiterung der Aenderung #011, damit man das VISCALC von Modell I auch fuer Modell III benutzen kann.

VC/CMD,03,69 aendern Sie von E6 01 28 in E6 03 28

VC/CMD,03,83 aendern Sie von E6 01 28 in E6 03 28

Mit der folgenden Aenderung kann eine Suche im Directory durchgefuehrt werden. Beachten Sie, dass das Directory im relativen Sektor 170 beginnen muss und nur 2 Granules lang sein darf. Dies ist der Fall, wenn GPL=2, DDSL=17 und DDGA=2 (Standard).

VC/CMD,68,B8 aendern Sie von 90 80 40 in 90 82 40

********** Aenderung #027 ********** 02.09.81 **********

Diese Aenderung im DISASSEM bewirkt, dass mit der Option RTD verhindert wird, dass zusaetzliche Daten geschrieben werden, die nicht in den File gehoeren.

DISASSEM/CMD,04,11 aendern Sie

```
von    5A D5 DD E1 DD
in     5A CD E0 6D DD
```

DISASSEM/CMD,24,60 aendern Sie

```
von    00 00 00 00 00 00 00
in     D5 DD E1 21 A0 6F C9
```

xxxxxxxxxx Aenderung #028 xxxxxxxxxx 02.09.81 xxxxxxxxxx

Mit dieser Aenderung ein Fehler im FORMAT behoben bei der Eingabe des Parameters PFST. Die Kombination von Y und PFST wird nicht mehr zurueckgewiesen.

Sie werden vielleicht bemerkt haben, dass weder der Parameter KDD noch RWF in der Beschreibung von FORMAT enthalten ist. KDD ist ebenso wie in der Version 1 ein legaler Parameter, RWF wurde bereits in der Aenderung #007 beschrieben.

SYS6/SYS,03,BE aendern Sie von 02 F9 86 in 02 B9 86

SYS6/SYS,04,90 aendern Sie von 00 02 4E in 00 00 4E

xxxxxxxxxx Aenderung #029 xxxxxxxxxx 02.09.81 xxxxxxxxxx

Dies ist keine Aenderung, sondern nur eine Information.

1. In Kapitel 12 wurde wohl nicht deutlich genug gemacht, dass Disketten unter bestimmten Voraussetzungen im NEWDOS/80 Version 2 zwischen Modell I und III getauscht werden koennen. Lesen Sie sich sorgfaeltig die Optionen BK und BN von SYSTEM durch (auch wenn Sie ein Modell III haben), das Kommando WRDIRP und die Funktion W des Programms DIRCHECK.

   a. Benutzer von NEWDOS/80 Modell III koennen Disketten von Modell I benutzen, wenn die SYSTEM-Option BK=Y gesetzt wird, die richtigen PDRIVE-Angaben gemacht wurden fuer einfache Schreibdichte, und entweder das DOS-Kommando WRDIRP oder die Funktion W des Programmes DIRCHECK ausgefuehrt wurde. Danach kann diese Diskette wie jede andere auch im Modell III unter NEWDOS/80 arbeiten. Diese Diskette kann jedoch nicht im Modell I unter TRSDOS Version 1.3 benutzt werden, dafuer jedoch unter NEWDOS/80 Version 1, wenn eine fuer diese Version gueltige Formatangabe bei PDRIVE gemacht wurde. Wenn aber die Diskette fuer die NEWDOS/80 Version 1 benutzt wurde, und im Directory eine Veraenderung vorgenommen wurde, kann sie erst wieder nach erneuter Behandlung mit WRDIRP oder DIRCHECK Option W (NEWDOS/80 Version 2) fuer das TRSDOS Version 1.3 hergenommen werden (NEWDOS/80 Version 1 hat keine dieser 2 Funktionen).

   b. Benutzer, die das NEWDOS/80 Version 2 sowohl fuer Modell I als auch fuer Modell III anwenden, koennen die Disketten mit einfacher Schreibdichte durch Setzen der Parameter BK=Y und BN=Y im Kommando SYSTEM beim Modell I, und des Parameters BK=Y beim Modell III, bei beiden Systemen benutzen. Jede Diskette muss dann noch ueber das Kommando WRDIRP oder die Option W von DIRCHECK schreibgeschuetzt werden. Dies kann sowohl beim Modell I als auch beim Modell III getan werden. Jede so behandelte Diskette kann jetzt sowohl fuer Modell I als auch fuer Modell III benutzt werden (vergessen Sie nicht die richtigen Angaben von PDRIVE), jedoch nicht zusammen mit dem TRSDOS 2.3.

2. In der BASIC-Funktion RENUM in Kapitel 7 sind die Angaben fuer RENUM X und RENUM U,X nicht korrekt und sollten geloescht werden. Falls der Parameter X benoetigt wird, so

muss dies ueber das erste Format angegeben werden. Hierbei ist X jedoch unwirksam, wenn nicht der Wert fuer ppppp und / oder qqqqq mit eingegeben werden.

3. Einige Benutzer haben nach mehr Erklaerungen gefragt fuer das COPY einer ganzen Diskette und dem Unterschied zwischen Format 5 und 6 beim COPY.

   a. Format 5 beim COPY kopiert die ganze Diskette Sektor fuer Sektor. Es duerfte damit etwas schneller sein als das Format 6, das jeden File einzeln kopiert. Bei einer relativ leeren Diskette duerfte allerdings Format 6 schneller sein.

   b. In Format 5 wird das 'Von'-Directory direkt auf die 'Nach'-Diskette uebertragen. Es hat damit die gleiche Groesse und die gleiche Position auf beiden Disketten. In Format 5 wird das Directory getrennt angelegt, so wie in Format 1 bis 4.

   c. Grundsaetzlich muessen in Format 5 die PDRIVE-Angaben fuer beide Disketten in den Parametern GPL, DDSL und DDGA gleich sein. Die anderen Parameter duerfen unterschiedlich sein, jedoch muss die 'Nach'-Diskette mindestens ebensoviele Spuren haben wie die 'Von'-Diskette. In der Version 2 duerfen auch die Anzahl der Sektoren pro Spur unterschiedlich sein.

   d. Im Format 5 mit der Option BDU koennen auch Disketten kopiert werden, die wegen anderer Umstaende nicht kopierbar waeren. In Format 6 kann man Disketten kopieren, die

      - eine unterschiedliche Anzahl von Granules pro 'Lump' haben,
      - einen unterschiedlichen Startsektor des Directories haben,
      - eine unterschiedliche Anzahl von Sektoren fuer das Directory haben, oder
      - die weniger Spuren haben als die 'Von'-Diskette.

   e. Format 6 kopiert alle Files oder auch nur eine bestimmte, festgelegte Anzahl von Files einer Diskette.

   f. Sowohl in Format 5 als auch in Format 6 kann die 'Nach'-Diskette mit FMT formatiert werden oder mit NFMT nicht formatiert. Wird eine Diskette mit einem Magneten geloescht, so wird sie danach komplett formatiert, als auch inclusive der Bytes, die nur vom Controller und den Laufwerken benutzt werden. Manche Anwender loeschen grundsaetzlich eine Diskette vor einem Backup. Selbstverstaendlich werden durch das Formatieren alle Daten geloescht.

   g. Wird im Format 6 eine ganze Systemdiskette kopiert, so muss in der Version 2 der Parameter FMT angegeben werden. Nur dies stellt sicher, dass im BOOT-Sektor und in der Kontrollregion fuer SYSTEM und PDRIVE die richtige Information eingetragen wird. Ebenso werden nur dann die Systemfiles auf der richtigen Position des Directories eingetragen.

   h. In Format 6 mit dem Parameter NFMT werden Systemfiles nur dann kopiert, wenn sie bereits auf der 'Nach'-Diskette stehen. Dies wurde deshalb gemacht, weil Systemfiles einen bestimmten Platz im Directory belegen, und nur FORMAT kann sicherstellen, dass dieser Platz auch zur Verfuegung steht.

   i. Das Format 5 ohne den Parameter BDU und das Format 6 mit dem Parameter FMT bewirken beide, dass die Diskette spaeter als Systemdiskette benutzt werden kann. Aus diesem Grunde wird in beiden Faellen der

BOOT-Sektor richtig initialisert und die angaben von PDRIVE fuer das Laufwerk 0.

********** Aenderung #030 ********** 10.09.81 **********

Mit dieser Aenderung kann das DOS Files von und auf Disketten, die im Modell I unter TRSDOS 2.3B und hoeher laufen, kopieren. Ebenso koennen Disketten vom Modell III mit TRSDOS 1.3 und hoeher kopiert werden. Beachten Sie dabei die Abschnitte "Unvertraeglichkeit NEWDOS/80 Version 2 mit TRSDOS 2.3" auf Seite 166 und "Unvertraeglichkeit NEWDOS/80 Version 2 mit TRSDOS 1.3" auf Seite 167 sowie das Flag M von PDRIVE.

Mit dieser Aenderung wird die Definition des TI-Flags M (siehe "PDRIVE" auf Seite 34) von PDRIVE erweitert. Wenn der Parameter TI=M angegeben ist, und der Parameter TD eine Diskette mit doppelter Schreibdichte spezifiziert (z. B. TD=E), so ist das Flag TI=I mit eingeschlossen und das System nimmt an, dass es sich um eine Floppy mit TRSDOS fuer das Modell III handelt. Ist TI=M und TD fuer Disketten von einfacher Schreibdichte angegeben (z. B. TD=A), so wird angenommen, dass es sich um eine Diskette vom TRSDOS Version 2.3B oder hoeher handelt. TI=I ist dann nicht mit eingeschlossen und darf nicht angegeben werden.

Bei Radio Shack hat man begonnen, im Vertrieb mit neuen Programmen (z. B. COBOL) das Betriebssystem TRSDOS 2.3B mit auszuliefern. Dieses neue Betriebssystem ist etwas unterschiedlich von TRSDOS 2.3, einschliesslich der Behandlung von den RBA-Werten (siehe "RBA" auf Seite 162). Diese Disketten sind nicht kompatiebel mit NEWDOS/80-Disketten, und sie koennen auch nur, wie die Modell III Disketten, ueber COPY angesprochen werden. FREE und DIR arbeiten noch mit diesen Disketten (ausser einigen Flags und des EOF-Wertes).

Files auf Modell III Disketten oder auf Modell I Disketten mit TRSDOS 2.3B koennen nur dann unter NEWDOS/80 benutzt werden, wenn sie auf eine NEWDOS/80 Diskette kopiert werden.

Die Aenderung #013 darf nicht ausgefuehrt sein, wenn Files vom TRSDOS 2.3B kopiert werden sollen. Die Aenderung #013 ist fuer Disketten vom TRSDOS 1.1 oder 1.2.

SYS6/SYS,00,7D aendern Sie CD B3 63 21 in CD 9F 5C 21

SYS6/SYS,15,65 aendern Sie

von
```
                        00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
   01 00 EA 5C 00 00 00
```

in
```
                        CD B3 63 3A BE 59 47 3A B9 59 4F
   3A CC 59 57 3A C7 59 5F 79 A3 CB 6F 28 06 78 AA
   0F DA 47 67 78 CB 69 20 04 CB 6B C8 7A 0F D8 21
   5B 5B 22 F7 55 21 B1 5B 22 5D 5B 21 00 5C 22 61
   5B AF 32 7F 61 67 6F 22 BF 61 22 B5 5B 3E 18 32
   01 00 EA 5C 2D 61 C9
```

SYS6/SYS,22,6B aendern Sie CD B3 63 CD in CD 9F 5C CD

```
SYS16/SYS,01,E9 aendern Sie  05 28 02 3E 03 CD
                       in    05 00 C4 7E 51 CD


SYS16/SYS,02,A0 aendern Sie  CB E3 DD 7E 05 FE 06 C0
                       in    CB 43 3E 06 C4 78 51 C0


SYS16/SYS,04,8B aendern Sie

von    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
in     00 CB E3 DD BE 05 C9 DD CB 0F 56 C8 3E 03 C9
```

********** Aenderung #031 ********** 10.09.81 **********

Diese Aenderung bewirkt, dass

1. das COPY kein Update-Flag setzt im Modell I TRSDOS 2.3B und Modell III TRSDOS, da dieses Bit eventuell anderst belegt ist.
2. im COPY bei CBF auch die Angabe von /typ im TRSDOS 2.3B von Modell I oder TRSDOS von Modell III moeglich ist.

```
SYS6/SYS,10,83 aendern Sie  C3 39 44 DD  in  C3 ED 5C DD


SYS6/SYS,15,B7 aendern Sie

von    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
in     3A 82 42 CB 6F 28 04 DD CB 02 EE C3 39 44


SYS6/SYS,16,F0 aendern Sie  61 20 0D 3A 96 59 CB
                       in   61 3A 96 59 20 0A CB
```

********** Aenderung #032 ********** 10.09.81 **********

Diese Aenderung bewirkt, dass ein Fehler im SUPERZAP bei der Funktion DM und dem Kommando F, beseitigt wird.

```
SUPERZAP/CMD,00,11 aendern Sie

von    00 00 00 00 00 00 00
in     6B 63 4B 22 C3 54 C9


SUPERZAP/CMD,05,0B aendern Sie

von    00 6B 63 4B CB
in     00 CD 0D 52 C3
```

********** Aenderung #033 ********** 11.09.81 **********

Mit dieser Aenderung koennen Sie den COBOL-Compiler der Firma Ryan-McFarland Corp. in der Version 2.3B, der von Radio Shack vertrieben wird, unter NEWDOS/80 Version 2 auf dem Modell I und III betreiben. Diese geaenderte Version laeuft nicht mehr unter TRSDOS, aus diesem Grunde muessen Sie alle COBOL-Files auf eine NEWDOS/80-Diskette kopieren (siehe dazu auch Aenderung #030).

RSCOBOL/CMD,99,7D aendern Sie

```
von    44 24 3E 40 49
in     44 82 27 44 49
```

RSCOBOL/CMD,99,CF aendern Sie

```
von    44 13 1F 44
in     44 82 27 44
```

RUNCOBOL/CMD,115,82 aendern Sie

```
von    44 3E 40 49 40 24 17
in     44 27 44 49 40 82 17
```

RUNCOBOL/CMD,115,99 aendern Sie

```
von    02 1F 44 11 44 13 0A
in     02 27 44 11 44 82 0A
```

########## Aenderung #034 ########## 13.09.81 ##########

Mit dieser Aenderung wird ein Fehler beseitigt, der entweder beim CLOSE oder KILL eines Files auftrat, wenn der zugehoerige File den ersten Eintrag eines Sektors im Directory mit einem FPDE oder FXDE belegte.

Dieser Fehler wurde zusammen mit den Aenderungen fuer das TRSDOS fuer Modell III mit eingebaut und leider erst zu spaet erkannt.

SYS3/SYS,01,A9 aendern Sie

```
von    00 7D D6 17 BB 30 1D
in     00 7B C6 16 BD 38 1D
```

########## Aenderung #035 ########## 13.09.81 ##########

Dies ist nur eine Information, keine Aenderung.

1. Korrektur der Dokumentation von EDTASM

   Das Laden von Source-Text von der Kassette ist nur im Modell I moeglich, ebenso das Schreiben von Source-Text.

2. Das Kommando ROUTE MM=adr1 erfordert einen Wert von groesser 51FFH. Diese Korrektur wurde auch bereits durchgefuehrt.

PAGE IS
MISSING

A.0 BESCHREIBUNG FILEHANDLING

A.1 BEZEICHNUNGEN

Im NEWDOS/80 der Firma APPARAT wurden sehr grosse Erweiterungen gegenueber dem Orginal-BASIC der Firma TANDY gemacht. Genaugenommen wurde der komplette Teil des RAM's neu geschrieben und ueberarbeitet. Viele dieser neuen Moeglichkeiten uebersteigen jedoch das Fachwissen eines Anfaengers und auch von durchschnittlichen Programmierern. Aus diesem Grunde wurde ein Anwender dieses Systems gebeten, aus seiner Sicht heraus eine verstaendliche Beschreibung zu verfassen, in der Hoffnung, dass alle Moeglichkeiten von allen Programmierern ausgeschoepft werden koennen. Als Vorlage fuer diese Beschreibung diente das Handbuch der Fa. APPARAT ueber das NEWDOS/80 (Teil 1). Bei Unklarheiten und eventuellen Differenzen hat in jedem Falle das Handbuch Gueltigkeit, bei dieser deutschen Uebersetzung selbstverstaendlich das Handbuch in englischer Sprache. Zur Einarbeitung wird empfohlen, sich zuerst in die Fachausdruecke einzuarbeiten und dann langsam, Schritt fuer Schritt, durch das Handbuch zu gehen.

Zur besseren Verstaendlichkeit werden hier zuerst die verwendeten Fachausdruecke und Abkuerzungen erlaeutert.

Byte Die kleinste adressierbare Einheit sowohl im Speicher als auch auf der Floppy. Es werden abwechselnd sowohl Byte als auch das Wort Zeichen (Character) benutzt.

Sector Der kleinste einlesbare Block auf einer Floppy. Er enthaelt 256 Bytes.

Granule Die kleinste, reservierbare Einheit auf einer Floppy. Ein Granule besteht aus 5 Sektoren, die Sektornummern beginnen mit 0 oder 5.

Spur Die Speichereinheit, die waehrend einer Umdrehung der Floppy beim Lesen oder Schreiben ueberstrichen wird. Eine Spur (Track) besteht aus mehreren Sektoren. Bei der Minidiskette des TRS80 hat eine Spur 10 Sektoren oder 2 Granules, die bei Sektor 0 einer Spur anfangen.

BOOT/SYS Der File, der bereits von Format auf Spur 0, Sektor 0 erzeugt wird dient zum Laden des DOS-residenten Programmes SYS0/SYS beim Start der Diskette. Wenn dieses Programm nicht auf der Diskette ist, erscheint die Meldung 'NO SYSTEM' auf dem Bildschirm. BOOT/SYS wird vom ROM-Teil des TRS80 geladen beim Reset und beim Einschalten, ebenfalls dann, wenn ein Assemblerprogramm den Befehl HALT ausfuehren will.

DIR/SYS Der 2. File, der bereits von Format auf die Floppy geschrieben wird steht normalerweise auf der Spur 17 (11H). Dieser File enthaelt alle Informationen, die das System ueber Datenformat und -inhalt der Diskette benoetigt. In diesem File stehen auch der Name der Diskette, das Datum der Generierung, die Befehlsfolge fuer Power-up (s. AUTO), Informationen ueber jeden File wie Filename, Erweiterung, Passwort, Adresse auf der Floppy, die Laenge und die Attribute.

FPDE - File Primary Directory Entry Der Eintrag in DIR/SYS, der die Hauptinformationen ueber einen File und seine Adresse auf der Floppy enthaelt. Dieser Eintrag wird zusammen mit einem File erzeugt und auch dann geloescht, wenn dieser File geloescht wird. Tatsaechlich aber bleibt dieser Eintrag weiter im Directory stehen, bis dieser Platz von einem anderen File belegt wird. Der Unterschied zwischen TRSDOS und NEWDOS/80 ist der, dass beim TRSDOS der komplette Eintrag sofort geloescht wird, waehrend beim NEWDOS/80 nur das 1. Wort des Eintrags auf 0 gesetzt wird. Gleiches geschieht mit den FXDE's (s. u.).

Extent Element Ein 2 Byte langer Eintrag im FPDE/FXDE, der entweder die Adresse eines Elementes eines Files oder die Adresse eines FXDE's im Directory angibt.

FXDE - File eXtended Directory Entry Ein Element im Directory, das Erweiterungen des Files enthaelt, wenn diese Information nicht im Haupteintrag (FPDE) untergebracht werden konnte. Der Haupteintrag kann nur 4 Adressen von Erweiterungen enthalten, in dem Moment, wo eine 5. Erweiterung reserviert wird, wird auch der FXDE erzeugt und die Adresse dieser Erweiterung eingetragen. Die Adresse des FXDE's steht im FPDE. Bei einer 9. Erweiterung wird ein 2. FXDE erzeugt und seine Adresse im 1. FXDE eingetragen.

DEC - Directory Entry Code Eine 1 Byte lange Information ueber einen Directory Entry. Dieser DEC wird vom System benutzt, einen Eintrag im Directory oder eine freie Stelle schnellstmoeglich zu finden.

GAT - Granule Allocation Table Ein Teil des Sektor 0 von DIR/SYS. Es kennzeichnet die Granules, die bereits belegt sind oder noch reserviert werden koennen. Es sagt aber nichts ueber den File aus, der ein Granule belegt.

HIT - Hash Index Table Sektor 1 von DIR/SYS. Jeder moegliche FPDE im Directory hat einen festen Platz im HIT-Sektor und enthaelt, wenn dieser Eintrag von

einem FPDE belegt ist, den Hash-Code (codierter Kurzcode) des Programmnamens. Dient zum schnelleren Auffinden eines bestimmten Files im Directory.

FCB - File Control Block Der 32 Byte lange Kontrollblock eines Files im Speicher. Wird auch IOB (Input / Output Block) genannt. Der FCB wird als Verbindung zwischen dem eigentlichen File und dem aufrufenden Programm benutzt. Vor dem OPEN enthaelt dieser FCB den Programmnamen, nach dem OPEN die Kontrollinformationen wie relative Position im File, EOF-Adresse, Adresse des File-Buffers und die Recordlaenge (LRECL). Nach dem CLOSE steht wieder der Filenamen im FCB.

Buffer Ein 256 Bytes langer Bereich im Speicher der als Zwischenbuffer fuer die Daten von und zur Diskette benutzt wird. Ein einzelnes GET / PUT / INPUT / PRINT bedeutet nicht automatisch, dass Zeichen zur oder von der Diskette in diesen Zwischenbuffer uebertragen werden (ausser einigen Ausnahmen, die spaeter erlaeutert werden), in jedem Falle werden aber Daten zwischen dem Variablenbereich des BASIC und dem Zwischenbuffer uebertragen. Das FIELD-Statement bewirkt keine Datenuebertragung, sondern nur einen Austausch von Variablenadressen gegen Variablennamen.

Filearea Ein Kontrollbereich des BASIC, der ungefaehr 300 Bytes lang ist und den FCB sowie den Buffer eines Files enthaelt. Dieser Bereich schafft die Verbindung zwischen BASIC-Programm und dem angesprochenen File und ist gueltig, solange der File geoeffnet ist. Dieser Bereich wird benutzt und angesprochen bei den Statements INPUT, PRINT, GET, PUT, FIELD, LOC, LOF und EOF. Alle Bereiche von Files arbeiten unabhaengig voneinander, auch dann, wenn mehrere gleichzeitig geoeffnet sind.

FAN - FileArea Number Ein BASIC-Ausdruck, der einen Wert von 1 bis incl. 15 hat. Er steht fuer den Filebereich, der einem bestimmten File zugewiesen wird (bei OPEN) oder zugewiesen wurde. Startet man BASIC ohne weiteren Parameter, wird Platz fuer 3 verschiedene Filebereiche reserviert. Anstelle des Defaultwertes 3 kann auch ein groesserer Wert eingegeben werden, was allerdings nur sinnvoll ist, wenn auch tatsaechlich mehr als 3 Files gleichzeitig geoeffnet sein sollen. Die Betonung liegt auf gleichzeitig, da ein Bereich nach dem CLOSE dieses Files einem anderen File zugewiesen werden kann.

RBA - Relative Byte Address Ein BASIC-Ausdruck, der einen Wert zwischen 0 und 16.777.215 annehmen kann. Er gibt die Adresse eines bestimmten Bytes innerhalb eines Files an. Dieses Byte kann der Anfang eines Bestandteils eines Records sein, der Anfang einer Gruppe von Bestandteilen oder der Anfang eines logischen Records. Der RBA-Wert ist nichts weiter als ein 3 Byte langer Integerwert, dessen erste 2 Bytes die relative Sektornummer innerhalb des Files sind (max.

65535), das 3. Byte ist die relative Position des Bytes innerhalb des Sektors (max. 255). In dieser Art werden Fileposition und EOF-Wert im FCB abgespeichert. Die Benutzung des RBA-Wertes fuer EOF ist der grundsaetzliche Unterschied zwischen TRSDOS und NEWDOS/80. Programme, die den EOF-Wert benutzen wie im TRSDOS-Handbuch beschrieben, koennen in dieser Art nicht unter NEWDOS/80 arbeiten.

REMBA - REMembered Byte Address Ist ein Feld in jedem FCB, das die relative Position eines bestimmten Files im RBA-Format enthaelt. Es wird dann benutzt, wenn bei einem GET/PUT als FP = $ angegeben wird und bewirkt, dass der File beim GET/PUT an der gleichen Stelle wie beim Start des letzten GET/PUT positioniert wird. Es wird nicht bei INPUT/PRINT-Files benutzt. Jedes OPEN-Statement setzt den Wert von REMBA auf ungueltig, genau wie die Verwendung von FP = !$RBA und !$% bei GET- oder PUT-Statements.

REMRA - REMembered Record Address Dieses Feld im Filebereich ist genau wie das Feld REMBA im RBA- Format und enthaelt die relative Position des ersten Bytes eines bestimmten Records. Man kann diesen Wert anzeigen mit der Funktion LOC(FAN)# und mit FP = # dazu benutzen, den File am Recordanfang des momentan gerade gueltigen Records zu positionieren, ohne Ruecksicht darauf, wo man gerade innerhalb dieses Records steht. Bei "MI" und "FI" Files sind REMRA und REMBA gleich. Der Wert wird bei verschiedenen FP-Angaben neu gesetzt. Er wird bei jedem PRINT / INPUT auf den RBA-Wert auf die Fileposition am Anfang der Operation gesetzt. Jedes OPEN auf einen File setzt diesen Wert auf ungueltig, ebenso die Verwendung von FP = !$RBA und !$% bei GET- oder PUT-Statements.

EOF - End Of File Der RBA-Wert des letzten Bytes im File plus 1. Er definiert das erste Byte das benutzt werden kann, wenn der File erweitert wird, und das Ende der Daten. Ebenso wird darin die Anzahl der Bytes in einem File angegeben. Diesen Wert kann man anzeigen ueber die Funktion LOC(FAN)%. Es sei nochmals darauf hingewiesen, dass der EOF-Wert im NEWDOS/80 anders benutzt und definiert ist als beim TRSDOS! Programme, die mit diesem EOF-Wert arbeiten, muessen bei einer Betriebssystemumstellung also geaendert werden !

EOR - End Of Record Der RBA-Wert des letzten Bytes in einem Record plus 1. Er gibt also das 1. Byte des naechsten Records an.

EOL - End Of Line Das Ende einer Zeile von BASIC-Text. BASIC benutzt das Zeichen hex. 0 um das Ende der Zeile zu definieren.

EOS - End Of Statement Das Ende eines BASIC-Statements. Das BASIC benutzt das Zeichen ":" um das Ende der Zeile zu definieren.

SOR - Start Of Record Der Beginn eines Records bei "MU" Files. Das NEWDOS/80 BASIC benutzt das Zeichen 70H zur Kennzeichnung eines Records.

FP - File Position Positionierungsangabe in einem GET oder PUT Statement um zu kennzeichnen, an welcher Stelle eines Files die Operation beginnen soll. In vielen Faellen findet jedoch keine Datenuebertragung statt, es werden dann nur verschiedene Werte des FCB's fuer die naechste Operation geaendert. Eine bestimmte FP- Angabe bewirkt, dass nur die Daten vom Filebuffer in diesen File uebertragen werden, ohne dass Daten vom Programm in den Filebuffer uebertragen werden. Fuer eine vollstaendige Beschreibung sehen Sie bitte im entsprechenden Kapitel nach.

File Item Dies ist eine Einheit von File-Speicherplatz, die entweder null oder mehr Bytes lang ist und entweder einen String oder einen numerischen Wert in ihrer internen Form enthaelt. Bei "MU"-, "MF"- und "MI"-Files werden ueber fuehrende Kontrollzeichen Datentyp und -laenge festgelegt.

Item Group Eine Gruppe von null oder mehreren Bestandteilen eines Files, die als eine Einheit bei GET, PUT, INPUT und PRINT Statements behandelt werden. Meist ist diese Gruppe von Variablen ein logischer Record, muss es aber nicht sein, beispielsweise bei teilweiser Ein- und Ausgabe eines Records.

Logical Record Eine logisch und physikalisch zusammengehoerende Gruppe von Daten, die normalerweise als eine Einheit von GET, PUT, INPUT und PRINT Statements verarbeitet werden. Ein oder mehrere Bestandteile eines logischen Records, jedoch nicht der ganze Record, werden ueber die teilweise Ein- und Ausgabe verarbeitet.

IGEL - Item Group Expression List Eine Reihe von einem oder mehreren Ausdruecken, die durch ein Komma getrennt sind und mit einem Semikolon abgeschlossen werden, die mit dem Inhalt eines Files von der momentanen Fileposition an korrespondieren. Die Liste definiert die Variablennamen, welche die Daten enthalten, oder den BASIC-Ausdruck (Formel), dessen Ergebnis bei einem PUT in den File geschrieben werden soll oder die Namen der Variablen, in welche die Daten bei einem GET geschrieben werden sollen. Es spezifiziert die Daten, die bei einem GET oder PUT uebertragen werden. Ein IGEL kann vom Disk-BASIC nur ausgefuehrt werden, wenn er Bestandteil eines GET oder PUT ist. Separierte IGEL's muessen mit einem GOTO uebersprungen werden.

Wenn ein EOS (End Of Statement), EOL (End Of Line), REM oder ein ' (Apostroph) gefunden werden bei der Abarbeitung eines IGEL's, wird der Rest der Zeile uebergangen (bei REM oder '), und mit der naechsten, nummerierten Zeile fortgefahren. Auf diese Art koennen in diese Listen Kommentare eingefuegt werden, die spaeter das Lesen des Programmes einfacher machen.

Die nachfolgenden Beispiele haben die gleiche Funktion:

Beispiel 1:

```
100   GET 1,,110 : GOTO 160
110   REM - BUCHUNGSSATZ-IGEL
120   BN!, : BUCHUNGSNUMMER
130   BA!, : BUCHUNGSART
140   TN%, : TRANSAKTIONSNUMMER
150   TP!; : TRANSAKTIONSPOINTER
```

Beispiel 2:

```
100   GET 1,,,BN!,BA!,TN%,TP!; : ' INFORMATIONS-
SATZ LESEN
```

Wird ein Fehler in einem IGEL entdeckt, so wird in der Fehlermeldung die Zeilennummer des GET / PUT Statements angegeben und nicht die Zeilennummer des IGEL's.

IGEL Expression Der Ausdruck innerhalb eines IGEL's definiert den Namen der Variablen, die den gelesene Wert aus einem File erhaelt bei einem GET Statement, oder er definiert den Namen einer Variablen, die den Wert enthaelt, der auf den File geschrieben werden soll in einem PUT Statement. Der Ausdruck kann folgende Formen haben:

1. Ausdruck

   Normalerweise wird der Name einer BASIC-Variablen gegeben. Bei einem PUT Statement kann es allerdings im Prinzip alles sein, was rechts vom Gleichheitszeichen bei einem LET Statement erlaubt ist, ausgenommen Bestandteile einer Filearea. Bei "FF" und "FI" Files muss jeder Variablenname den BASIC-Zusatz haben, aus dem der Typ der Variablen zu ersehen ist (!, #, $ oder %). Bei Feldvariablen muss nur der eigentliche Feldname diesen Zusatz haben, nicht jedoch der Index. Variable in einem Marked Item File benoetigen diesen Zusatz nicht im IGEL.

2. (Laenge)Ausdruck

   Dieser Ausdruck hat eine Laengenangabe, die entweder selbst wieder ein Ausdruck sein kann, dessen Ergebnis einen Wert zwischen 0 und 255 incl. ergeben kann, oder eine Integerzahl in diesem Bereich. Die Laengenangabe darf nur bei Stringvariablen auftreten und gibt folgendes an:

- Bei Marked Item Files: Die maximale Anzahl von Zeichen, die in den File bei einem PUT geschrieben werden oder bei einem GET gelesen werden. Wenn bei einem PUT die angegebene Laenge groesser ist als die tatsaechlich gesendete Laenge, wird nur die tatsaechliche Anzahl geschrieben. Ebenfalls wird bei einem GET nur die Variable in ihrer tatsaechlichen Laenge gelesen bis zu der als Maximum angegebenen Laengenangabe. Ist die zu schreibende oder zu lesende Variable dagegen laenger als angegeben, so wird sie abgeschnitten. Die Laengenangabe ist optional bei Marked Item Files.

- Bei Fixed Item Files: Die exakte Anzahl von Zeichen, die bei einem GET empfangen oder bei einem PUT gesendet werden. Die Laengenangabe muss bei allen Strings im IGEL von Fixed Item Files angegeben werden.

3. (Laenge)$

   Gibt die Anzahl der Bytes im File an, die bei GET und PUT ignoriert werden sollen. Bei einem PUT auf einen existierenden Record werden die ignorierten Bytes nicht veraendert, bei einem PUT auf einen neuen Record werden diese ignorierten Bytes auf 0 gesetzt. Diese Art von Bestandteil eines IGEL's kann nur bei Fixed Item Files benutzt werden, und da vor allem bei einer teilweisen Ein und Ausgabe eines Records.

4. (Laenge)#

   Gibt die Anzahl der Bytes an, die in einem GET Statement ignoriert werden. Bei PUT Statements werden entsprechend viele Bytes mit 0 in den Record geschrieben, egal, was vorher in dem Record stand. Diese Art kann bei teilweiser Ein- und Ausgabe von Files benutzt werden. Ebenfalls kann es bei "MU" und "MF" Files benutzt werden, um fuer einen spaeteren Update Platz zu reservieren. Fuer einen "MI" File ist es relativ sinnlos, da dieser Filetyp kein Update erlaubt.

IGELSN - IGEL Statement Number Die Statementnummer eines freistehenden IGEL's kann bei GET und PUT Operationen benutzt werden. Wenn der IGEL ueber mehrere Zeilen geht, so ist die erste Zeilennummer anzugeben. Der freistehende IGEL kann ueberall im Sourceprogramm stehen und kann sowohl fuer teilweise als auch komlette Ein- / Ausgabe eines Records benutzt werden. Dieser IGEL kann sowohl fuer die GET als auch fuer die PUT Statements gleichzeitig benutzt werden. Dies kann bei Aenderungen eine grosse Zeitersparnis bringen, wenn nicht mehrere GET's und PUT's geaendert werden muessen, sondern nur ein IGEL.

LRECL - Logical RECord Length Bei record-segmentierten Files wird der aktuelle Wert ("FF" und "MF" Files) oder der Maximalwert (bei "MU" Files) der Recordlaenge in Bytes angegeben. Die maximale Recordlaenge fuer Marked Item Files ist 4095 Bytes. Die LRECL wird im OPEN-Statement angegeben. Bei allen Filetypen darf die Anzahl der uebertragenen Bytes die LRECL nicht ueberschreiten, sonst kommt es zu einer Fehlermeldung. Bytes, die mehrfach in einen Record geschrieben werden, zaehlen natuerlich nur einmal.

Bei "MU" Files wird die Recordlaenge durch die uebertragenen Daten bestimmt, nicht aber durch die LRECL im OPEN, falls angegeben.

Fixed Item Files Bei dieser Art von Filestruktur wird die Art der Abspeicherung von dem benutzten IGEL im entsprechenden GET und PUT bestimmt. Die maximale Recordlaenge bei Fixed Item Files betraegt 4095 Bytes. Ein Fixed Item File kann entweder keine Bestandteile in einem Record besitzen (Record ist leer) oder bis zu mehreren. Zwischen zwei Records oder zwischen zwei Bestandteilen eines Record wird vom System keine Trennung eingefuegt. Ein Record kann ueber mehrere Sektoren gehen. Bei "FF" Files benutzt das System die im OPEN angegebene LRECL, um von einem Record auf den naechsten weiterzugehen. Bei "FI" Files ist der Programmierer selbst fuer die Segmentierung zustaendig, sie kann nicht vom System vorgenommen werden.

Marked Item Files Bei dieser Art von Filestruktur kommt es auf den Typ und die Laenge der Daten an, wie die Segmentierung aussieht. Alle Bestandteile eines Marked Item Files haben mindestens ein Kontrollbyte, dem kein, eines oder mehrere Datenbytes folgen. Da diese Kontrollbytes echt in den File geschrieben werden, muessen sie bei der Festlegung der maximalen Recordlaenge beruecksichtigt werden. Ein Marked Item File kann in Records von gleicher Laenge unterteilt werden ("MF" Files) oder in Records von unterschiedlicher Laenge ("MU" Files). Bei diesen Filetypen achtet das System entweder auf die exakte oder auf die maximale Recordlaenge, die 4095 Bytes nicht ueberschreiten darf (incl. Kontrollbytes). Ein Marked Item File kann auch vom Programmierer selbst in Records segmentiert werden ("MI" Files), das System kann jedoch in diesem Fall nicht die ueblichen Kontrollen durchfuehren.

Field Item Files Ein solcher Filetyp kann vom System mit dem FIELD Statement in einzelne Bestandteile oder in Gruppen von Bestandteilen unterteilt werden. Dies ist die einzige Fileart, auf die wahlfrei im TRSDOS zugegriffen werden kann. Auch NEWDOS/80 unterstuetzt die Anwendung dieses Filetyps.

Record Segmented Ein solcher Filetyp kann vom System in einzelne Records unterteilt werden. Der File kann entweder ein Fixed Item File (FI File) oder ein Marked Item File (MI oder MU File) sein.

User Segmented Ein solcher Filetyp kann vom System nicht in einzelne Records unterteilt werden. Dies bleibt ausschliesslich dem Programmierer ueberlassen. Der File selbst kann entweder ein Fixed Item File (FI File) oder ein Marked Item File (MI File) sein.

MU Files Ein NEWDOS/80 Filetyp, der record-segmentiert ist und markierte Bestandteile enthaelt. Alle Records in einem MU-File koennen unterschiedliche Laenge haben, bis zur maximalen Laenge, die im OPEN-Statement angegeben wird. Der Defaultwert ist 4095, die Laengenangabe ist optional.

FF Files Ein NEWDOS/80 Filetyp, der record-segmentiert ist und feste Bestandteile enthaelt. Alle Records in einem FF-File haben die gleiche Laenge. Die Recordlaenge muss beim OPEN angegeben werden.

MF Files Ein NEWDOS/80 Filetyp, der record-segmentiert ist und markierte Bestandteile enthaelt. Alle Records in einem MF-File haben die gleiche Laenge. Die Recordlaenge muss beim OPEN angegeben werden.

FI Files Ein NEWDOS/80 Filetyp, der benutzersegmentiert ist und feste Bestandteile enthaelt. Er enthaelt in diesem Sinne keine Records, sondern nur einen String von Datenbytes. Der Anwender hat selbst fuer eine Segmentierung zu sorgen.

MI Files Ein NEWDOS/80 Filetyp, der benutzersegmentiert ist und markierte Bestandteile enthaelt. Er enthaelt in diesem Sinne keine Records, sondern nur eine Gruppe von Datenbytes. Der Anwender hat selbst fuer eine Segmentierung zu sorgen.

Whole Record I/O Bei dieser Zugriffsart werden mit jedem GET und PUT immer alle Daten eines logischen Records uebertragen. Es ist zwar am einfachsten zu programmieren, hat jedoch den Nachteil, dass eben immer alle Daten eines Records uebertragen werden muessen, egal ob sie gebraucht werden oder nicht. Diese Technik muss bei updates auf "MU" und "MF" Files benutzt werden, der restliche, freibleibende Platz wird mit Fuellbytes aufgefuellt (hex. 0).

Partial Record I/O Bei dieser Zugriffsart werden nur die Bestandteile eines Records uebertragen, die absolut notwendig sind. Diese Art des Zugriffs ist zwar wesentlich effizienter als die komplette Uebertragung eines Records (Whole Record I/O), dafuer leidet aber etwas die Lesbarkeit und die Verstaendlichkeit eines Programmes darunter. Diese Zugriffsart benutzt alle Varianten der Filepositionierung (FP), die fuer eine Datenuebertragung notwendig sind. Man kann damit auf alle NEWDOS/80 Filetypen zugreifen beim Lesen, auf "FF" und "FI" File kann damit ein Update erfolgen.

A.2 CMD

Das CMD - Verb wurde im NEWDOS/80 auf noch mehr sehr hilfreiche Funktionen erweitert. Die allgemeine Form ist :

CMD String-Ausdruck

wobei String-Ausdruck die gewuenschte Funktion ist. Die erlaubten Werte im NEWDOS/80 BASIC sind :

DOS - Kommando Dies erlaubt es dem Programmierer, im BASIC - Kommando Modus oder von einem laufenden BASIC Programm aus, ein DOS Kommando auszufuehren. Das eingegebene Kommando wird zuerst ueberprueft, ob es im MINI - DOS ausgefuehrt werden kann, wenn ja, wird es sofort abgearbeitet und zu dem naechsten BASIC - Statement zurueckgekehrt bzw. in den BASIC Kommando Modus.

Kann das gewuenschte Kommando nicht im MINI - DOS ausgefuehrt werden, wird der benoetigte Bereich im Speicher freigemacht (das BASIC Programm wird nach oben an die Variablen geschoben) und ueber den BASIC- und Variablenteil die Pruefsumme errechnet. Ist nicht genuegend Platz zum Verschieben vorhanden, erscheint die Fehlermeldung OUT OF MEMORY. Der String-Ausdruck wird in den DOS - Eingabebuffer uebertragen und das DOS wird aufgerufen, um das Kommando auszufuehren. Die Laenge des String-Ausdrucks ist begrenzt auf 79 Bytes, es ist die Laenge des DOS - Eingabebuffers. Nach der Ausfuehrung des DOS - Kommandos springt die Routine wieder in das Basic zurueck und berechnet erneut die Pruefsumme. Wenn diese Pruefsumme nicht gleich ist wie die zuvor errechnete, wird der Fehler MEMORY FAULT gemeldet und das System kehrt in das DOS zurueck und meldet DOS READY, da das BASIC nicht fortgesetzt werden kann.

Diese CMD - Erweiterung ist die groesstmoegliche Hilfe ueberhaupt. In BASIC beispielsweise kann der Benutzer das Directory ansehen, Datum und Uhrzeit vom Programm aus setzen, Files umbenennen und vieles mehr. Es koennen ebenso andere Programme aufgerufen werden. Bei den so aufgerufenen Programmen gibt es allerdings verschiedene Restriktionen, z. B. Programmende und Memoryplatz, die zu beachten sind, damit wieder ins BASIC zurueckgesprungen werden kann. Verschiedene Programme koennen nicht aufgerufen werden.

Bei dieser Art, ein DOS - Kommando aufzurufen, muss streng darauf geachtet werden, dass BASIC nicht auf diese Art aufgerufen werden darf. Will man BASIC neu starten, muss man zuerst voll ins DOS zurueckkehren. Dies hat etwas mit dem Aufruf verschachtelter Programme in NEWDOS/80 zu tun. Mit CMD"BASIC" werden Sie also diverse Probleme bekommen, dagegen kann CMD"S=BASIC" jederzeit angewandt werden.

S=DOS-Kommando Wenn die ersten zwei Zeichen des Strings "S=" sind, wird das BASIC komplett verlassen und das DOS-Kommando ausgefuehrt. Wenn das Kommando ausgefuehrt ist, geht das System in den Eingabemodus DOS READY. Das DOS-Kommando darf in diesem Falle jedes

von DOS aufrufbare Programm sein sowie jeder andere DOS-Befehl wie z. B. DIR.

E Dies bewirkt, dass der letzte angezeigte DOS - Fehler, egal wann und wie dieser aufgetreten ist, nochmals angezeigt wird. Im allgemeinen werden die DOS - Fehler vom BASIC anhand der Fehlernummer nochmals interpretiert und dann als BASIC - Fehler angezeigt und gespeichert. Mit CMD"E" wird dieser Fehler nochmals als DOS - Fehler angezeigt, was manchmal informativer ist als die BASIC - Fehlermeldung.

F=POPS Dieses Kommando bewirkt, dass alle noch vorhandenen Informationen im Stack ueber GOSUB - RETURN und FOR - NEXT geloescht werden. Nach Beendigung dieses Kommandos wird mit dem naechsten Statement fortgefahren. Dieses Kommando findet vor allem da Anwendung, wo beispielsweise aus einer oder mehreren Subroutines direkt zurueckgesprungen wurde oder ein oder mehrere FOR ... NEXT - Loops nicht zu Ende gefuehrt wurden. In diesem Falle bleiben alle Informationen ueber die verwendeten Variablen im Stack und verbrauchen unnoetig Platz.

Mit diesem Kommando wird selbst kein Ruecksprung in den hoechsten Level des Programms ausgefuehrt, es werden nur alle Rueckspruenge geloescht und damit in den logisch hoechsten Level zurueckgekehrt. Der eigentliche Ruecksprung muss der Programmierer selbst ausfuehren.

F=POPR Dieses Kommando bewirkt, dass der momentane GOSUB - RETURN - Level und alle in diesem Level nicht abgeschlossenen FOR - NEXT - Loops vom Stack geloescht werden. Danach wird mit dem naechsten Statement fortgefahren.

Im Gegensatz zu F=POPS wird nur der letzte Ruecksprung und die dieser Subroutine zugehoerigen FOR ... NEXT - Loops geloescht. Wenn dem Programmierer z. B. bekannt ist, dass er in der vierten verschachtelten Subroutine ist und er moechte in den 2. Level zurueckkehren, so kann er schreiben : CMD"F=POPR" : CMD"F=POPR" : GOTO (GOSUB+1).

F=POPN Dieses Kommando bewirkt, dass die Kontrollinformation des letzten FOR ... NEXT - Loops vom Stack geloescht wird. Das Programm faehrt mit dem naechsten Statement fort. Folgt hinter diesem Kommando ein Variablenname, wird angenommen, dass alle FOR ... NEXT - Loops zurueck bis zu diesem Namen geloescht werden sollen.

Beispiel, wo CMD"F=POPN" benutzt werden muss :

```
500  FOR X=1 TO 10
510  FOR Y=1 TO 30
.
.
.
640  GOTO 1020
.
.
```

```
820  NEXT Y
830  NEXT X
.
.
1020 FOR Z=1 TO 200
1030 FOR X=1 TO 3
.
.
1160 NEXT X
1170 NEXT Z
```

Bei dieser Folge erhaelt man einen NEXT WITHOUT FOR - Fehler, obwohl tatsaechlich kein Fehler vorliegt. Man kann ihn umgehen durch aendern der Zeile 640 in :

```
640  CMD"F=POPN" X : GOTO 1020
```

Die zwei Kontrollbloecke fuer X und Y werden vom Stack geloescht, dann erst wird auf das Statement 1020 gesprungen.

A.3 FILE POSITIONIERUNG

Sehr vielfaeltig und auch verwirrend ist die grosse Anzahl der Moeglichkeiten, einen File im NEWDOS/80 zu positionieren. Als benutzerfreundlich kann diese Art nicht gerade angesehen werden. Dafuer koennen aber auch selbst schwierige Faelle programmiert werden.

In diesem Kapitel soll versucht werden, dem Benutzer alle Moeglichkeiten so gut als moeglich zu erlaeutern.

Die File-Positionierung (FP) ist ein Operant in allen NEWDOS/80 GET und PUT Statements. In seiner einfachsten Form ist er ein numerischer Ausdruck. Darueber hinaus gibt es spezielle Zeichen zur Positionierung, die entweder zusammen mit dem numerischen Wert angegeben werden oder auch alleine. Diese Zeichen koennen durch ein Leerzeichen vom numerischen Wert getrennt werden.

FP-Wert und seine Bedeutung

Ohne Angabe (NULL) Wenn es sich um einen "MU", "MF" oder "FF" - File handelt und der Wert 'REMRA' (siehe FCB, - Remembered Record Address) gueltig ist, wird der naechste sequentielle Record des Files bearbeitet. In jedem anderen Fall wird die momentane Position nicht veraendert und die Operation beginnt an dem Punkt, wo sie beim letzten GET oder PUT beendet wurde. Bei einem OPEN auf eine Datei wird 'REMRA' auf ungueltig und die Position auf null gesetzt (ausser bei "E" - Files, Position auf EOF). Der erste Zugriff auf einen File erfolgt immer bei der momentanen File - Position.

M Die momentane File - Position wird nicht veraendert. Mit der Angabe von '*' kann man auf einen bestimmten Record mehrfach zugreifen mit GET oder PUT. Bei "MU", "MF" und "FF" Files kann '*' nicht zur Positionierung auf den naechsten Record benutzt werden, obwohl der zurueckgegebene Wert RBA, den man ueber LOC(FAN)! erhaelt, bereits die Adresse des naechsten Records enthaelt. Auf den naechsten Record kommt man erst, wenn man mit einem GET oder PUT mit einem Positionierungsparameter (oder null) zugreift.

Wenn 'REMRA' gueltig ist, wird auf den gleichen Record nochmals zugegriffen, andernfalls gibt es eine Fehlermeldung. Bei "MU", "MF" und "FF" Files bedeutet dies, dass man auf den gleichen Record wie zuvor nochmals zugreifen kann, und zwar auf den Beginn dieses Records, wenn noetig, auch mit anderen Variablennamen oder anderen Ausdruecken in der IGEL's. Bei "MI" und "FI" Files greift man auf die gleichen Daten zu wie beim vorhergehenden GET oder PUT.

$ Wenn 'REMBA' gueltig ist wird nochmals auf das gleiche Byte innerhalb eines Records zugegriffen,

andernfalls gibt es eine Fehlermeldung. Dies erlaubt es dem Benutzer, nochmals auf das gleiche Byte oder die gleiche Variablengruppe eines Records wie im vorangegangenen GET oder PUT zuzugreifen. Es arbeitet mit allen File-Typen auf die gleiche Art.

% Diese Angabe als Positionierung bewirkt eine "PSEUDO FIELD"- Operation. Es werden keine Daten uebertragen und der FCB wird nicht veraendert. Es muss vorher kein OPEN auf den File erfolgen. Dieser Parameter wird bei "FF" und "FI" Files benutzt um die Datenfelder im BASIC zu initialisieren.

& Die Angabe von '&' erfolgt nur in PUT-Statements und hat keine Auswirkung auf die Positionierung. Es bewirkt nur, dass der momentane Inhalt des File - Puffers auf die Floppy geschrieben wird. Es sollte benutzt werden, wenn sich kritische Daten im Puffer befinden. Es kann benutzt werden um die Filenummer eines PRINT-Files diesem File zuzuordnen.

!RBA Bei dieser Art der FP - Angabe wird der File an einer bestimmten Stelle positioniert. Bei "FF" und "MF" Files wird ueberprueft, ob tatsaechlich ein Record an dieser Stelle beginnt, bei "MU" Files muss der Wert von !RBA auf ein 'SOR' - Zeichen deuten. Groesste Vorsicht seitens des Programmierers ist erforderlich, da eine falsche Angabe von !RBA oder ein falscher Wert in !RBA einen File zerstoeren koennen (speziell bei "FF" und "FI" Files). Dies ist die einzige Art, wie man wahlfrei auf einen Record in "MI" und "FI" Files zugreifen kann.

!% Bei dieser FP - Angabe wird in der gleichen Art wie im vorangangenen Teil verfahren, mit dem Unterschied, dass der File auf das momentane Ende (EOF - Wert) positioniert wird. Diese Angabe wird benutzt, um einen File zu erweitern. Der File muss zuvor mit "R" eroeffnet werden. Eine Fehlermeldung wird erzeugt, wenn der File mit "D" eroeffnet wurde.

!$RBA Diese RBA - Angabe erlaubt es, den File auf einem bestimmten Record, als Vorbereitung zum naechsten Zugriff, zu positionieren, es findet kein Datentransfer statt. In diesem GET oder PUT Statement braucht kein 'IGEL' angegeben zu werden. Die Positionierung selbst findet jedoch erst beim naechsten PRINT/INPUT oder GET/ PUT statt, und auch nur dann, wenn in diesem GET/PUT oder PRINT/ INPUT keine Positionsangabe vorhanden ist. Diese Positionierung kann angewandt werden z. B. bei einem GET oder PUT in einer Subroutine, der File wird zuvor positioniert, und in der Subroutine wird im GET oder PUT keine Positionierung (NULL) angegeben.

!$% Prinzipiell gleiche Funktion wie !$RBA, nur dass automatisch als naechste Position der

End-Of-File-Wert genommen wird. Wird ebenfalls erst beim naechsten GET/PUT oder INPUT/PRINT wirksam.

!@RBA Diese Angabe wird nur mit einem PUT benutzt und bewirkt, dass der EOF-Wert dieses Files auf den Wert von 'RBA' gesetzt wird. Um den echten EOF-Wert im 'FPDE' des Files zu aendern muss dieser File geschlossen (CLOSE) werden. Der Wert 'RBA' muss dem Filetyp entsprechen. Bei "MF" und "FF" Files muss der 'RBA'-Wert ein Vielfaches der Recordlaenge (LRECL) sein.

RN (Record Nummer) Dieser Wert hat die gleiche Funktion wie in TRSDOS. Er muss eine ganze Zahl zwischen 1 und 32767 sein (einschl.). Der einzige Unterschied zwischen TRSDOS und NEWDOS/80 ist der, dass in NEWDOS/80 anstelle der physikalischen Record - Nummer (der Sektor - Nummer) eine logische Recordnummer angegeben wird. Dieser Wert entspricht dem tatsaechlichen Record und wird in einen RBA - Wert umgerechnet und wie !RBA benutzt (gilt fuer "MF" und "FF" Files).

Wie man aus dieser Aufstellung erkennen kann, aendern die Positionsangaben die Werte REMBA, REMRA oder EOF. Die nachfolgende Aufstellung kann zur Entscheidungshilfe herangezogen werden.

FP	REMBA	REMRA	EOF
(NULL)	1	1	6
*	1	2	6
@	3	4	6
@	4	4	6
%	4	4	4
&	4	4	4
!RBA	4	1	6
!%	1	1	7
!@RBA	1	5	4
!@%	5	5	4
!@RBA	4	4	1
RN	1	1	6

Bedeutung der Zahlen :

1. das Feld wird auf den RBA - Wert gesetzt, der sich aus dem FP - Wert ergibt.
2. wenn REMRA zum Zeitpunkt der Ausfuehrung ungueltig ist oder wenn es sich um einen "MI" oder "FI" File handelt, wird dieses Feld auf den Wert von RBA gesetzt. In anderen Worten : der File wird auf den Recordanfang gesetzt oder, falls bereits auf Recordanfang, bleibt unveraendert.
3. das Feld wird auf den Wert von REMRA gesetzt.
4. das Feld wird nicht veraendert.
5. das Feld wird auf einen ungueltigen Wert gesetzt.
6. das Feld wird geaendert, wenn der File durch ein PUT-Statement erweitert wird (bei OUTPUT- und UPDATE-Files).
7. das Feld wird bei OUTPUT- und UPDATE-Files bei einem PUT veraendert.

Zusammenfassend kann man sagen, dass es 4 Werte im FCB fuer die Positionierung gibt :

Momentane Fileposition Das Feld momentane File - Position nimmt bei einem GET/PUT Statement drei verschiedene Werte an. Dies sind

GPP1 die File - Position beim Start der Ausfuehrung des GET oder PUT. Solange der File nicht geschlossen und wieder eroeffnet wird ist es der Wert GPP3 des letzten GET oder PUT.

GPP2 dieser Wert ist der RBA - Wert nach der Positionierung und vor der ersten Daten - Tranfer.

GPP3 dieser Wert ist der RBA - Wert nach dem letzten Datentransfer, auch dann, wenn nur Daten uebersprungen wurden.

REMRA bei "MU", "MF" und "FF" Files enthaelt dieser Wert den RBA - Wert des Recordanfangs des momentanen Files. Bei "MI", "FI" und INPUT/PRINT - Files ist es der momentane REMBA - Wert (Byte - Adresse).

REMBA ist der Wert, bei dem das vorangegangene GET oder PUT gestartet wurde. Bei Files, die record-segmentiert sind, wird REMRA auf der Wert von REMBA gesetzt, wenn REMBA auf einen Recordanfang deutet.

EOF EOF ist der RBA-Wert des letzten Bytes eines Files plus 1. Bei "MU", "MF" und "FF" Files deutet dieser Wert auf den naechsten Record, der geschrieben werden soll. Bei "MI", "FI" und INPUT/PRINT Files deutet dieser Wert auf das naechste sequentielle Byte, das geschrieben werden soll.

Der grundsaetzliche Ablauf und die Verarbeitung des FP - Wertes im FCB sieht folgendermassen aus :

- Die Positionierung erfolgt, falls notwendig, von der momentanen Position (GPP1) auf die gewuenschte Position. Dies kann das Schreiben eines veraenderten Puffers auf die alte Position, das Errechnen des neuen Sektors und das Lesen des neuen Sektors in den Puffer beinhalten.
- Der RBA-Wert der gewuenschten Position wird als momentane Fileposition abgespeichert (GPP2).
- REMBA wird auf die momentane Fileposition (GPP2) gesetzt.
- Wenn es sich bei dem File um einen PRINT/INPUT File oder einen segmentierten File handelt und die momentane Position (GPP2) auf einen Record - Anfang deutet, wird REMRA auf die momentane Position (GPP2) gesetzt.

- Die Datenuebertragung wird ausgefuehrt, die momentane Fileposition (GPP3) enthaelt den RBA - Wert des letzten uebertragenen Bytes plus 1.
- Wenn der File erweitert wurde oder FP = I@RBA, wird der EOF-Wert auf die neue Position gesetzt.

A.4 OPEN

Genau wie beim TRSDOS muss ein File zuerst eroeffnet werden, bevor man auf ihn zugreift. Das OPEN-Statement bewirkt die Verbindung zwischen dem aufrufenden Programm und dem File bzw. seinem Puffer im Speicher. Diese Verbindung zwischen BASIC und einem File wird im FCB eingetragen und kontrolliert. Ist ein File einmal eroeffnet, koennen Daten von diesem File ueber INPUT oder GET gelesen werden, oder Daten koennen in diesen File ueber PUT und PRINT geschrieben werden. Ist die Datenuebertragung beendet, sollte dieser File wieder geschlossen werden, d. h. die Verbindung zwischen Basicprogramm oder DOS und diesem File wird unterbrochen.

In NEWDOS/80 werden die gleichen Funktionen in OPEN unterstuetzt wie in TRSDOS: "I" fuer sequentielle INPUT-Files (INPUT- Statement), "O" fuer sequentielle OUTPUT-Files (PRINT-Statement), "R" fuer Files mit wahlfreiem Zugriff (GET- und PUT- Statements), "E" fuer sequentielle Files, auf die ab dem EOF- Zeichen gestartet wird (E = Extent) und "D" fuer Files mit wahlfreiem Zugriff, bei denen der Benutzer nicht hinter EOF zugreifen will.

Ferner unterstuetzt das NEWDOS/80 das GET und PUT bei Files, die mit "I", "O" und "E" eroeffnet wurden, allerdings muss vorher beim OPEN ein NEWDOS/80-File angegeben werden.

Es wird nochmals darauf hingewiesen, dass dieser Teil der NEWDOS/80-Beschreibung zur besseren Verstaendlichkeit von einem Benutzer geschrieben wurde nach der Vorlage der NEWDOS/80- Spezifikation. In Zweifelsfaellen hat diese Spezifikation Gueltigkeit.

Allgemeine Form des NEWDOS/80 OPEN :

```
OPEN M,FAN,FILENAME
OPEN M,FAN,FILENAME,FT
OPEN M,FAN,FILENAME,FT,LRECL
```

wobei

M der String oder die Stringvariable ist, die entweder "I", "O", "R", "E" oder "D" angibt. Daraus ergibt sich also die Zugriffsart auf den eroeffneten File und die Positionierung.

FAN ist die Nummer des Files der eroeffnet wird.

FILENAME ist der String oder die Stringvariable, die den Filenamen und Typ etc. des Files enthaelt, der eroeffnet werden soll.

FT ist der Ausdruck (entweder String oder Stringvariable), der den NEWDOS/80 Filetyp angibt. Er enthaelt entweder "FI", "FF", "MI", "MF" oder "MU". Sobald 'FT' angegeben wird in einem OPEN-Statement, muss ueber GET und PUT auf diesen File zugegriffen werden. INPUT und PRINT duerfen nicht benutzt werden, ebenso darf das BASIC-FIELD-Statement nicht benutzt werden. Alle GET/PUT muessen entweder einen 'IGEL' oder eine 'IGELSN' enthalten. Das Programm darf nicht direkt auf den File-Puffer zugreifen oder die Daten dort direkt veraendern. Zwei 'FT'-Angaben benoetigen die Angabe von 'LRECL' im OPEN-Statement, bei einer weiteren 'FT'-Angabe kann 'LRECL' angegeben werden.

LRECL ist ein Ausdruck oder eine Zahl zwischen 1 und 4095, die bei allen record-segmentierten Files angegeben werden muss. Bei "FF" und "MF" Files gibt sie die genaue Recordlaenge an, bei "MU" Files ist sie die maximale Recordlaenge.

Die Grundform des BASIC-Statements 'OPEN' hat sich nicht geaendert, braucht also bei einer Umstellung auf NEWDOS/80 nicht veraendert zu werden. Die Erweiterung zu der Grundform zeigt an, dass es sich bei einem File um einen NEWDOS/80-File handelt und dieser beim Zugriff entsprechend behandelt werden muss.

Von allen Filetypen in NEWDOS/80 ist der "MU"-File der, welcher am leichtesten zu verstehen und zu benutzen ist. "MU" definiert einen File, der variable Recordlaenge hat und jeder Record ein Steuerzeichen enthaelt. Die Laenge des Records ergibt sich aus der Adresse (RBA) des momentanen SOR-Zeichens und der Adresse des naechsten SOR-Zeichens oder von EOF (beim letzten Record). Die Recordlaenge braucht im OPEN-Statement nicht angegeben zu werden; falls eine Recordlaenge angegeben ist, so ist diese die maximale Laenge eines Records und darf dann nicht ueberschritten werden.

Ein Record eines "MU"-Files kann veraendert oder ausgetauscht werden mit einem anderen, gleichlangen oder kuerzeren Record, jedoch nicht mit einem laengeren Record. Wird ein Record durch einen kuerzeren Record ersetzt, so steht dieser neue Record linksbuendig am Platz des alten, er wird aufgefuellt mit dem Fuellzeichen 0 (hex. NULL) bis zum naechsten SOR-Zeichen. Dieser kuerzere Record kann zwar spaeter wieder gegen einen laengeren Record ausgetauscht werden, jedoch darf dann dieser neue Record wiederum nicht laenger sein als der urspruengliche Record.

Der "MU"-File wurde entwickelt, um den alten sequentiellen PRINT/INPUT-File zu ersetzen. Sein Vorteil ist, dass zwischen fortlaufenden Strings kein Trennungszeichen vorhanden sein muss (im alten PRINT/INPUT-File musste man ein Komma zwischen zwei Bloecken schreiben, um sie fuer das INPUT-Statement zu trennen). Ein weiterer Vorteil ist auch, dass BASIC-Variable in ihrer internen Form in den File geschrieben werden und nicht als String. Eine Zahl mit doppelter Genauigkeit wird also als 8 Byte lange Folge geschrieben und nicht als Character-String mit bis zu 14 Bytes. Beim Input wird die Zahl wieder direkt in die Variable kopiert. Es sei nochmals darauf hingewiesen, dass diese Zahl dann aber mit dem SOR-Zeichen insgesamt 9 Bytes lang ist. Wenn im OPEN die Recordlaenge angegeben wurde, muss diese auch die SOR-Zeichen beruecksichtigen (in diesem Fall darf also die Recordlaenge nicht kleiner als 9 sein).

Die naechste einfache Zugriffsart auf Files ist der "MF"- und "FF"-File. Beide Filetypen sind record-segmentiert und haben Records von fester Laenge. Bei beiden Arten wird angenommen, dass sie intern die gleiche Datenstruktur haben. Das OPEN-Statement muss die exakte Recordlaenge enthalten. Bei "MF"-Files muss ebenfalls das SOR-Byte hinzugezaehlt werden, "FF"-Files haben kein Trennbyte. Jedes GET- und PUT-Statement vergleicht anhand des IGEL's die Laenge des Records mit der Recordlaenge, die im OPEN-Statement angegeben wurde und meldet einen Fehler, wenn diese voneinander abweichen.

Die schwierigste zu benutzende Zugriffsart sind die "MI" und "FI" Files. "MI" und "FI" gibt an, dass der File record-segmentiert ist unter der Kontrolle des Benutzers. Die 'LRECL' darf beim OPEN nicht angegeben werden. In dieser Art kann ein File eine komplexe Datenstruktur erhalten, ohne dass diese von BASIC unterstuetzt wird. BASIC kann nicht von einem Record auf den naechsten gehen.

PAGE IS
MISSING

A.6 GET

Im TRSDOS-BASIC wird das GET-Statement dazu benutzt, einen bestimmten Record (Sektor) in den Puffer eines bestimmten File-Bereiches zu lesen. Mit dem FIELD-Statement werden dann die Pointer auf die gewuenschte Adresse innerhalb des Puffers gesetzt. Aufgrund der Zugriffsmethode wurde diese Art von File FIELD-ITEM-File genannt.

In NEWDOS/80 wird zur zusaetzlichen Unterstuetzung dieses das GET-Statement dazu benutzt, Daten eines Files in eine Variable zu schreiben oder einen File zu positionieren fuer den spaeteren Zugriff. Der eigentliche Transfer von Daten von der Diskette in den Puffer wird nur dann vom System ausgefuehrt, wenn diese benoetigt werden, d. h. wenn der Inhalt des Puffers ein anderer ist als der gewuenschte.

Der Verfasser weisst nochmals darauf hin, dass die Informationen in diesem Kapitel aus der Spezifikation von NEWDOS/80 entnommen wurden, und, wenn es zu einer Diskrepanz kommt zwischen Handbuch und Spezifikation (gleiches gilt fuer die deutsche Uebersetzung), immer die Angaben in der Spezifikation Gueltigkeit haben (englisches Orginal).

Die grundsaetzliche Form von GET:

```
GET FAN                          (FP = NULL)
GET FAN,FP
GET FAN,FP,,IGELSN
GET FAN,FP,,IGEL
```

Die Angaben 1 und 2 werden fuer die sog. FIELD-ITEM-Files benutzt und sind kompatibel mit dem TRSDOS-BASIC, solange nicht eine spezielle Form von FP benutzt wird, die es nur im NEWDOS/80 gibt. Diese Arten koennen selbstverstaendlich auch im NEWDOS/80 benutzt werden.

Format 3 und 4 gibt es nur im NEWDOS/80-BASIC. Diese Art muss benutzt werden, wenn im OPEN der Filetyp FT angegeben wurde. Format 3 wurde erweitert durch die verschiedenen zusaetzlichen FT-Angaben im NEWDOS/80.

In Format 3 wird die Nummer des Statements angegeben, in dem der 'IGEL' steht, der die verschiedenen Variablennamen enthaelt. In Format 4 werden direkt die Namen der Variablen angegeben als Teil des GET-Statements.

Wurde das GET-Statement erfolgreich beendet (ohne Fehler), so enthaelt der File-Control-Block FCB die Angaben, die fuer das naechste GET oder PUT ohne Positionierungsangabe FP folgende Resultate bringt:

1. das naechste Byte im File bei Fixed Item Files
2. das naechste Byte im File bei Marked Item Files
3. den naechsten Record mit 256 Bytes bei Field Item Files

Wenn ein Fehler waehrend des Lesens auftritt, bleiben alle Werte des FCB's erhalten wie vor dem GET-Statement. Nachdem der Fehler behoben ist, kann deshalb das gleiche GET nochmals ausgefuehrt werden. Die Werte der Variablen, die im IGEL stehen, sind nach einem Fehler ungueltig und sollten nicht benutzt werden.

Wenn das GET-Statement entweder einen IGEL oder die Zeilennummer des IGEL's (IGELSN) enthaelt, so werden die gelesenen Werte des Files in die Variablen uebertragen, deren Namen im IGEL stehen.

Bei Fixed Item Files :

- Strings werden in der Laenge uebertragen, die als Zusatz vor der Stringvariablen stehen. Die Laenge der Variablen (len(.#)) wird auf diesen Wert gesetzt.
- Numerische Angaben im IGEL werden in der Laenge uebertragen, die der internen Laenge der Variablen entsprechen. (Integerzahlen sind 2, Gleitkommazahlen mit einfacher Genauigkeit sind 4 und Gleitkommazahlen mit doppelter Genauigkeit sind 8 Bytes lang).
- Vor dem ersten GET, mit dem Daten von der Floppy gelesen werden, sollte ein GET stehen, dessen FP * % ist. Der File, dessen Nummer mit FAN angegeben wird, muss dafuer nicht unbedingt eroeffnet sein, da dieses GET nur eine PSEUDO-FIELD-Operation fuer Fixed-Item-Files ausfuehrt. Wenn dabei die einzelnen Elemente (Variablen) des IGEL's durchlaufen werden, bleiben die numerischen Variablen gleich, die Stringvariablen werden auf die im IGEL vor dem Variablennamen angegebene Laenge gesetzt. Dabei werden sie, wenn sie schon existieren und unterschiedlich lang sind, entweder gekuerzt oder verlaengert. Wenn der momentane Inhalt dieser Variablen nicht im Stringbereich des BASIC liegt, werden diese Variablen mit der richtigen Laenge in den Stringbereich gebracht. Dies wurde getan um sicherzustellen, dass im Stringbereich des BASIC genuegend Platz fuer die Ausfuehrung des eigentlichen GET-Statements zur Verfuegung steht. Wurde einmal diese PSEUDO-FIELD-Operation ausgefuehrt, sollten die betroffenen Variablen nur noch mit LSET oder RSET veraendert werden.
- Wenn der File record-segmentiert ist und von der momentanen Position bis zum Ende des Records weniger Zeichen zur Verfuegung stehen als im IGEL angegeben, wird die Fehlermeldung "RECORD OVERFLOW" angezeigt.

Bei Marked Item Files :

- Wird ein IGEL ohne Variable angegeben, so wird dieser Record uebersprungen.
- Die Laengenangabe vor einer Stringvariablen wird benutzt, um die zu uebertragende Anzahl von Bytes auf diese Laenge zu begrenzen. Ist die Stringvariable im File kuerzer als die angegebene Laenge, so wird die Laenge der Variablen auf den tatsaechlichen Wert gesetzt. Ist der String im File dagegen laenger als angegeben, so wird die Variable entsprechend gekuerzt (wie bei LSET).
- SOR und Fuellzeichen werden beim Lesen ignoriert. Ist der File record-segmentiert und beim GET ist im Buffer weniger Platz vorhanden, als Zeichen angefordert wurden, so kommt es zur Fehlermeldung "RECORD OVERFLOW".
- Versucht man, eine Variable mit einem unkorrekten IGEL einzulesen, kommt es zu der Fehlermeldung "TYPE MISMATCH". Dies ist dann der Fall, wenn versucht wird, mit einer Stringvariablen einen Teil einzulesen, der beispielsweise eine Integerzahl ist.

Zwei spezielle Arten von Filepositionierung FP koennen benutzt werden, um den File fuer weitere GET's zu positionieren. Dies sind FP = !@RBA und !$%. Bei diesen Angaben werden REMRA und REMBA auf einen ungueltigen Wert gesetzt. Diese Angaben duerfen jedoch nicht in Format 3 oder 4 auftreten, solange keine Daten transferiert werden.

Es darf mehr als ein GET benutzt werden, um Daten von einem einzelnen Record zu lesen. Diese Zugriffsart wird PARTIAL RECORD I/O genannt (teilweise Ein- / Ausgabe auf einen Record). Diese spezielle Zugriffsart wird beisielsweise dann benutzt, wenn das erste Byte die Art des zu lesenden Records angibt und man nach dem Lesen dieses ersten Bytes gleich auf die entsprechende Verarbeitungsroutine springt, die dann den Rest einliest.

Die Zugriffsart mit teilweiser Ein- / Ausgabe eines Records kann sowohl bei Fixed-Item-Files als auch bei Field-Item-Files benutzt werden. Bei Field Item Files wird fuer jeden 256-Byte Block das Feld erneut aufgebaut fuer die notwendigen Felddefinitionen. Bei Marked Item Files werden die zu uebergehenden, also die nicht benoetigten Daten, einfach im IGEL freigelassen, d. h. nicht eingelesen. Bei Fixed Item Files muss die genaue Laenge der zu ueberlesenden Daten errechnet werden und im IGEL als (len)$ angegeben werden. Der enorme Vorteil von Fixed Item Files gegenueber den Marked Item Files ist der, dass in Fixed Item Files jedes Bestandteil eines Records unabhaengig von den anderen geaendert werden kann. Bei Marked Item Files muessen alle Bestandteile, die hinter dem zu aendernden liegen, auch zurueckgeschrieben werden, d. h., sie muessen in jedem Falle zuerst gelesen werden. Der Vorteil jedoch von Marked Item Files gegenueber den Fixed Item Files ist der, dass im Marked Item File durch die Kennzeichnung bedingt Records in unterschiedlichen Formaten geschrieben werden koennen. Dies koennen sowohl unterschiedliche Variablentypen als auch unterschiedliche Variablenanzahl sein.

A.7 PUT

Im TRSDOS BASIC bewirkt das PUT - Statement, dass der Buffer des angegebenen Files auf den vom Benutzer angegebenen Record (Sektor) geschrieben wird. Mit dem FIELD - Statement werden die Variablenpointer auf die entsprechende Adresse innerhalb des Buffers gesetzt. Laut Vereinbarung werden diese Files Field Item Files genannt.

Im NEWDOS/80 werden im PUT - Statement weiterhin die Field Item Files unterstuetzt, zusaetzlich jedoch noch folgende neue Filetypen : Fixed Item Files und Marked Item Files. Im NEWDOS/80 - BASIC bewirkt das PUT - Statement, dass bei Fixed und Marked Item Files die Daten, die im IGEL vereinbart sind, von der entsprechenden Variablen in den Filebuffer uebertragen werden. Die Uebertragung vom Filebuffer in den entsprechenden File wird direkt vom System kontrolliert. Ferner gibt es ein bestimmtes PUT- Statement, mit dem man direkt den Buffer auf die Diskette schreiben kann, ohne dass die Positionierung innerhalb des Buffers veraendert wird.

Wir weisen erneut darauf hin, dass die hier stehende Information der Beschreibung, Teil 1, entnommen wurde. Sollte es grundsaetzlich zu Unterschieden zwischen Teil 1 und 2 kommen, ist immer die englische Orginalbeschreibung des NEWDOS/80 von APPARAT massgebend.

Die Grundformen des PUT - Statements sind :

```
PUT fan                          (fp = Null)
PUT fan,fp
PUT fan,fp,IGELSN
PUT fan,fp,,IGEL
```

Die Grundformen 1 und 2 werden bei Field Item Files benutzt und sind kompatibel mit dem TRSDOS - BASIC, solange keine spezielle Art der Positionierung aus dem NEWDOS/80 benutzt wird. Diese Formen koennen selbstverstaendlich weiterhin in NEWDOS/80 benutzt werden.

Die Grundformen 3 und 4 sind neu im NEWDOS/80. Sie muessen immer dann bei einem File benutzt werden, wenn dieser File mit einem OPEN mit FT (Filetyp) eroeffnet wurde. In Format 3 wird die Zeilennummer des zu benutzenden IGEL's angegeben, in dem die Variablen oder Ausdruecke stehen. In Format 4 sind die Bestandteile des Files direkt im PUT angegeben. Format 2 kann im NEWDOS/80 ebenfalls benutzt werden, um den File auf eine bestimmte Position fuer die Datenuebertragung mit Format 3 und 4 zu stellen.

Nach Beendigung des PUT - Statements ist der Filebereich so positioniert, dass beim naechsten GET oder PUT ohne speziellen Positionierungsparameter die Verarbeitung bei folgenden Punkten beginnt :

1. beim naechsten Byte im File bei Fixed Item Files,
2. beim naechsten Bestandteil des Files bei Marked Item Files oder
3. beim naechsten Record mit 256 Bytes beim Field Item File.

Wird bei der Bearbeitung des PUT - Statements ein Fehler festgestellt, so wird der gleiche Status und die alte Positionierung, wie vor dem PUT, wieder hergestellt. Die Daten sind in

PAGE IS
MISSING

- Nahezu alles, was auf der rechten Seite des Gleichheitszeichens im LET - Statement erlaubt ist, darf auch als Ausdruck im IGEL vorkommen. Verboten sind nur Ausdruecke, die sich auf einen Filebereich beziehen. Ebenfalls ausgenommen sind die Funktionen LOC, LOF, EOF und andere Ausdruecke, die eine Filenummer (FAN) enthalten.

- Wenn eine Stringvariable einen Vorsatz mit der Laengenangabe hat bedeutet dies, dass es sich um die maximal zu uebertragende Laenge handelt. Wenn ein String kuerzer ist als diese Laengenangabe wird er uebertragen wie er ist, ist der String laenger, so wird er rechts abgeschnitten wie bei einem LSET- Statement.

- Strings benoetigen entsprechend ihrer Laenge ein oder zwei Markierungsbytes. Liegt die Stringlaenge zwischen 0 und 127 Bytes wird nur ein Markierungsbyte benoetigt, bei 128 Bytes und laenger werden zwei Markierungsbytes eingesetzt. Diese Markierungsbytes muessen bereits beim Definieren der Recordlaenge im OPEN beruecksichtigt werden.

- Numerische Bestandteile eines Records werden in ihrer internen BASIC - Form im Buffer abgespeichert, d. h. Integerzahlen mit 2 Bytes Laenge, Gleitkommazahlen mit 4 Bytes, Gleitkommazahlen mit doppelter Genauigkeit mit 8 Bytes Laenge. Da jede Zahl ein Markierungsbyte hat werden im Buffer 3, 5 und 9 Bytes belegt.

- Numerische Werte und Ergebnisse werden vor dem Schreiben in den File zuerst in die kompakteste Form gebracht, die moeglich ist. So wird beispielsweise die Zahl 3.14159 als Gleitkommazahl in den Buffer geschrieben (5 Bytes incl. Markierungsbyte), das Ergebnis von LEN(A$)-LEN(B$) als Integerzahl (3 Bytes incl. Markierung).

- Es duerfen 2 oder mehr PUT's benutzt werden, um alle Bestandteile eines Records in den Buffer zu schreiben. Die Anzahl der Bytes eines logischen Records darf aber nicht die max. Recordlaenge LRECL aus dem OPEN - Statement ueberschreiten, da es sonst zu dem Fehler RECORD OVERFLOW kommt.

- Wenn "MU" und "MF" Files mit wahlfreiem Zugriff (random access) zum Update eroeffnet wurden, wird bei einem PUT auf einen Record von der Position, bei der das PUT anfaengt, bis ans Recordende (bei LRECL, einem SOR oder EOF Byte) alles ueberschrieben. Sind die geschriebenen Daten kuerzer als der Rest vom Record, wird dieser mit Fuellzeichen aufgefuellt. Wenn also in dem Record hinter den zu ueberschreibenden Daten noch etwas steht, das spaeter noch benoetigt wird, muss dies zuvor gelesen werden und zusammen mit den zu veraendernden Daten erneut geschrieben werden. Es kommt auch dann zu einem Fehler, wenn urspruenglich mehr Bestandteile in einem Record standen, die aber durch das PUT geloescht wurden, und man versucht, die urspruengliche Anzahl von Bestandteilen zu lesen.

- Bestandteile eines "MI" Files koennen nicht veraendert werden, da das System nicht weiss, wo der Record aufhoert und keine Fuellzeichen einfuegen kann wie bei "MU" und "MF" Files.

Gilt fuer Fixed und Marked Item Files :

- Der Filebuffer wird auf die Diskette geschrieben, wenn der Buffer voll ist und mehr Daten hineingeschrieben werden sollen.

- Wenn ein PUT mit der Positionierung & ausgefuehrt wird, wird der Buffer in seinem momentanen Zustand auf die Diskette geschrieben.

- Der Buffer wird bei einem CLOSE (entweder explizit mit der Filenummer oder alle Files ohne jede Filenummer) ebenfalls auf die Diskette geschrieben.

Wenn in einem Filebuffer kritische Daten stehen, sollte der Programmierer auf jeden Fall ein PUT mit & ausfuehren, um diese Daten zu sichern. Der Buffer wird auf die Diskette geschrieben, ohne dass die Positionierung des Files veraendert wird. Befinden sich im Buffer keine veraenderten Daten, wird dieses Statement ignoriert. Benutzt ein anderer Filebereich diese Daten speziell von der Diskette, so muss das PUT FAN,& zuerst ausgefuehrt werden. Alle Daten, die in den Buffer geschrieben werden, muessen mit einem PUT im Format 3 oder 4 in den Buffer uebertragen werden. Die Benutzung von PUT FAN,& ist nicht beschraenkt auf NEWDOS/80- Filetypen, es kann auch bei PRINT - Files angegeben werden.

Zwei spezielle Arten von Positionierungsparametern koennen fuer die Positionierung des naechsten GET, PUT, INPUT oder PRINT entsprechend dem Filetyp benutzt werden. Dies sind FP=!$RBA und FP=!%. Wird im naechsten GET, PUT, INPUT oder PRINT keine Positionierung angegeben, so erfolgt dieser Zugriff entweder auf den vorher angegebenen RBA - Wert oder auf EOF. Die Werte REMRA und REMBA werden dabei auf ungueltig gesetzt. Daten duerfen bei diesem PUT nicht uebertragen werden, da es sich nur um das Format 2 handelt.

Wie auch beim GET - Statement kann PUT auch beim sequentiellen Zugriff benutzt werden. Ein File kann sequentiell beschrieben werden, nachdem man ihn mit dem Mode "O" eroeffnet hat, ebenso wie er gelesen werden kann, nachdem er im Mode "I" eroeffnet wurde. Ebenso kann dieser File wahlfrei veraendert werden mit PUT's. Einzelne Datenfelder koennen, wenn es sich um einen "FF" oder "FI" File handelt, mit der teilweisen Record-I/O veraendert werden.

Wenn ein File vor unerlaubtem oder unbeabsichtigtem Veraendern geschuetzt werden soll, kann man ihn anstatt mit "R" mit "I" eroeffnen. Sobald auf so einen File ein PUT gemacht wird, zeigt das System den Fehler BAD FILE MODE an.

A.8 LOF

Mit der Funktion LOF erhaelt man die Nummer des letzten Records. Die allgemeine Form ist:

```
X = LOF(FAN)
```

FAN gibt an, fuer welchen File die Nummer des letzten Record verlangt wird. Wenn dieser File nichts enthaelt, ist in diesem Falle X=0. LOF darf nur bei FIELD-ITEM-Files und bei NEWDOS/80-Files vom Typ "MF" und "FF" benutzt werden.

A.9 LOC

Mit der Funktion LOC erhaelt man (im TRSDOS-BASIC) die Nummer des letzten Records (Sectors) eines bestimmten Files, der zuletzt mit einem GET oder PUT angesprochen wurde. In NEWDOS/80 wurde diese Funktion erweitert. Folgende Arten sind jetzt zulaessig :

```
LOC(FAN)
LOC(FAN)$
LOC(FAN)%
LOC(FAN)!
LOC(FAN)#
```

wobei mit FAN die Nummer des Files fuer den gewuenschten File angegeben wird. Der Aufruf erfolgt wie bei LOF mit X = LOC(FAN).

Format 1 (ohne Zusatz) entspricht der LOC-Art vom TRSDOS-BASIC. Bei FIELD-ITEM-Files erhaelt man die Nummer des mit GET zuletzt gelesenen oder mit PUT geschriebenen Sektors. Bei "MF" und "FF" Files erhaelt man die Nummer des mit GET/PUT zuletzt gelesenen/geschriebenen logischen Records. Wurde auf diesen File zuvor noch nicht zugegriffen, so erhaelt man den Wert 0, ausser der File wurde mit "E" (Extent) eroeffnet. In diesem Fall erhaelt man den Wert des letzten Records im File. Wann der angesprochene File keine Records von fester Laenge enthaelt, wird der Fehler "BAD FILE MODE" zurueckgemeldet.

Aus dem Resultat von Format 2 (("$") kann die relative Position innerhalb des Files zu EOF ersehen werden. Der Wert kann entweder 0 (logisch 'TRUE') oder 1 (logisch 'FALSE') annehmen und gilt fuer folgende Faelle :

Bei record-segmentierten Files (MU, MF und FF) :

- Wenn REMRA gueltig ist und der RBA-Wert des naechsten Records (nicht des momentanen !) gleich oder groesser als EOF ist, wird LOC(.)$ = 'TRUE', andernfalls 'FALSE'.
- Wenn REMRA ungueltig ist und der RBA-Wert der momentanen File-Position gleich oder groesser als EOF ist, wird LOC(.)$ = 'TRUE', andernfalls 'FALSE'.

Bei benutzersegmentierten Files (MI und FI) und bei PRINT/INPUT-Files :

- Wenn der RBA-Wert der momentanen File-Position gleich oder groesser als der EOF-Wert ist, nimmt LOC(.)$ den Wert 'TRUE' an, andernfalls 'FALSE'.

Das Ergebnis von Format 3 ("%" als Zusatz) ergibt die momentane EOF-Adresse des Files im RBA-Format. Dieser Wert kann dazu benutzt werden, eine Index-Tabelle aufzubauen, wenn ein neuer Wert ans Ende eines Files angefuegt wird. Diese Art der Benutzung von LOC erlaubt es, Indices fuer einen sequentiellen File bei der Erzeugung aufzubauen.

Als Ergebnis von Format 4 (mit "!" als Zusatz) erhaelt man, wenn REMRA einen gueltigen Wert hat, bei "MU", "MF" und "FF"

Files den RBA-Wert der File-Position des naechsten Records. In allen anderen Faellen (auch bei PRINT und INPUT Files) erhaelt man den Wert der momentanen Position. Bei record-segmentierten Files kann dieser Wert ebenfalls dazu benutzt werden, eine Indextabelle vor dem Schreiben aufzubauen. Bei benutzersegmentierten Files und bei PRINT/INPUT Files kann damit auch eine Indextabelle einer Datengruppe vor dem Schreiben aufgebaut werden.

Damit dieser Index-Wert auch tatsaechlich seinen richtigen Wert hat, muss ein PUT mit dem FP-Wert 0 (oder ein PRINT) erfolgen um die Daten in den File zu schreiben. Nahezu alle anderen Arten von FP-Wert erzeugen einen anderen RBA-Wert und damit einen Fehler. Ebenso wie in Format 3 kann mit LOC(.)! eine Indextabelle fuer sequentielle Files aufgebaut werden.

Mit Format 5 (Zusatz "@") erhaelt man den momentanen REMRA-Wert im RBA-Format. Wenn REMRA ungueltig ist erhaelt man die Fehlermeldung "BAD FILE MODE", beispielsweise dann, wenn zuvor ein GET oder PUT mit dem FP-Wert !@% erfolgt ist. Dieser ueber LOC(.)@ erhaltene Wert kann ebenfalls zur Erzeugung einer Indexliste benutzt werden, jedoch erst nach dem Schreiben der Daten oder des Records.

Durch die Benutzung der Werte, die man ueber LOC(FAN)%, LOC(FAN)! oder LOC(FAN)@ erhaelt, ist der Benutzer in der Lage, eine Indextabelle fuer entweder Records (bei record-segmentierten Files) oder einer Gruppe von Daten (bei benutzersegmentierten Files und PRINT / INPUT Files) aufzubauen. Diese Werte koennen beispielsweise in einen File (den gleichen oder einen anderen) geschrieben werden und spaeter beim GET oder PUT als 'FP' als !RBA oder !@RBA benutzt werden.

A.10 "MU" FILES

Der Filetyp "MU" ist der am einfachsten anzuwendende Filetyp im NEWDOS/80. Er wurde urspruenglich als Ersatz fuer den sequentiellen File im TRSDOS entwickelt. Im TRSDOS koennen die sequentiellen Files nachtraeglich nicht mehr veraendert werden (ausser dem "MI" File).

Der "MU" File ist in Records unterschiedlicher Laenge segmentiert und kann durch den automatischen Zusatz der SOR-Bytes direkt ohne Laengenangabe vom System gefunden werden. Aus diesem Grunde braucht sich der Programmierer nicht um die Groesse jedes Records zu kuemmern. Er kann aber auch einen kleineren Wert als den vom System erlaubten Maximalwert von 4095 Bytes je Record als seinen eigenen Maximalwert festlegen durch Angabe von LRECL im OPEN-Statement dieses Files. Jedes versuchte Ueberschreiten dieser Recordgrenze verursacht dann den Fehler "RECORD OVERFLOW".

Da der File record-segmentiert ist und jeder Record eine variable Laenge hat, wird jedes Bestandteil dieses "MU" Files vom System markiert. Diese Markierung erfordert natuerlich Platz in jedem Record und muss deshalb beim Festlegen der Recordgroesse beruecksichtigt werden. Dieses SOR-Zeichen markiert jeweils den Anfang jedes Records und kennzeichnet gleichzeitig die Art der Daten, die in dem Record enthalten sind. Werden Strings in einen Record geschrieben, so enthaelt dieses SOR-Byte die Laenge des Strings. Die Strings koennen genau wie bei BASIC- Strings 255 Bytes lang sein. Bei einer Stringlaenge von 128 bis 255 Bytes besteht das SOR-Zeichen aus 2 Bytes, sonst immer aus einem Byte. Numerische Werte werden in der BASIC-internen Form in den File geschrieben: IntegerZahlen also mit 2 Bytes, Gleikommazahlen mit 4 Bytes und Gleitkommazahlen mit doppelter Genauigkeit mit 8 Bytes Laenge.

Da die numerischen Werte in ihrer Basic-internen Form in den Record geschrieben werden, ist die Benutzung der Befehle CVx und MKx nicht mehr zulaessig, da dabei die Zahlen in einen Pseudostring umgewandelt werden.

Ein "MU" File kann erzeugt werden durch die Angabe von "O" im OPEN-Statement; der File wird dann generiert und mit den Daten aus den verschiedenen PUT's beschrieben, ohne Ruecksicht darauf, ob dieser File beim OPEN schon existiert hat oder nicht. Der "MU" File kann ebenfalls durch die Angabe von "R" im OPEN-Statement erzeugt werden, er darf dann aber vorher nicht existieren. Die dritte Methode ist das "E" im OPEN-Statement, es kann fuer bereits bestehende oder fuer neue Files benutzt werden.

Ein "MU" File kann sequentiell durch Angabe der Zugriffsart "E" (Extent) im OPEN-Statement erweitert werden. Wie schon oben erwaehnt muss der File dazu noch nicht existieren, in diesem Fall wird er erzeugt. Eine andere Art der Erweiterung eines Files ist die Betriebsart "R" im OPEN-Statement. Wird in dieser Art ein Positionierungsparameter angegeben, der hinter EOF liegt, wird der dazwischenliegende Platz mit Fuellzeichen (NULL) belegt. Ist der FP-Wert kleiner als EOF erfolgt keine Erweiterung des Files.

Ein "MU" File kann sequentiell gelesen werden durch Angabe von "I" im OPEN-Statement. Auf diese Art ist der File vor einem ungewollten Update geschuetzt. Auch in dieser Zugriffsart kann wahlfrei gelesen werden. Wenn der File im Moment des OPEN's noch nicht existiert erfolgt eine Fehlermeldung. Ebenfalls sequentiell kann auf einen File in der Betriebsart "R" und "D" zugegriffen werden. Zu beachten ist jedoch, dass jedes GET, ohne ein vorhergehendes PUT zur Positionierung, zu einem Fehler fuehrt.

Auf einen "MU" File kann auch ein Update erfolgen, wenn im OPEN- Statement die Zugriffsart "R" oder "D" angegeben wurde. Die Zugriffsart "D" schliesst eine Erweiterung des Files aus. In diesen Betriebsarten koennen sowohl ganze Records als auch

einzelne Bestandteile eines Records, entsprechend des FP-Wertes und des angegebenen IGEL's, geschrieben werden.

Zur besseren Verstaendlichkeit des "MU" Files sollte folgendes Beispiel eingegeben werden. In diesem Beispiel wird zuerst ein "MU" File erzeugt und dann beschrieben.

```
10  CLEAR 250
20  OPEN"O",1,"MU/DAT","MU"
30  PUT 1,,,"ABCDEF","2nd STRING";
40  PUT 1,,,STRING$(120,"*")+"0123456789";
50  I% = 2 : I! = 4 : I# = 8
60  PUT 1,,,I$,I%,I!,I#;
70  CLOSE
```

Retten Sie das Programm moeglichst auf Floppy fuer den Fall, dass Sie es spaeter brauchen.

In diesem Programm wurde die einfachste Form eines IGEL in den Statements 30 und 40 benutzt. Die Werte, die auf diesen File geschrieben werden sollen sind bereits in dem IGEL enthalten. Das PUT-Statement 60 enthaelt die 4 verschiedenen Variablentypen, die in BASIC moeglich sind: String, Integer, Gleitkomma und Gleitkomma mit doppelter Genauigkeit. Im OPEN-Statement wurde keine LRECL angegeben. Dadurch hat man die Moeglichkeit, Records mit einer Laenge von bis zu 4095 Bytes zu erzeugen.

Starten Sie das Programm durch Eingabe von RUN und untersuchen Sie den erzeugten Datenfile "MU/DAT" mit dem SUPERZAP, Betriebsart DFS (Display File Sector).

Das erste Byte im Sector ist 70H, es ist das SOR-Byte. Alle Records in diesem File starten mit diesem SOR-Byte. Seien Sie aber vorsichtig: wenn z. B. ein "p" geschrieben wird ist dies auch 70H, jedoch kein SOR. Dieser Fall kann ebenfalls bei numerischen Werten auftreten.

Das 2. Byte im Record ist das Markierungsbyte und besagt, dass die naechsten 6 Bytes ein String sind. Wenn Sie die 6 Bytes zum momentanen Stand hinzuzaehlen, kommen Sie auf das Markierungsbyte des naechsten Strings (8AH). Dieses Byte kennzeichnet eine Stringlaenge von 10 Bytes. Wenn Sie diesen String abzaehlen, kommen Sie auf das SOR-Byte des naechsten Records (Displacement 13H). Das nachfolgende Markierungsbyte (71H) besagt, dass ein String mit einer Laenge von > 127 folgt. Das darauffolgende Byte gehoert in diesem Falle nicht zum String, sondern gibt die Laenge an. Wird dieses Byte zum momentanen Displaystand hinzugezaehlt, so erhaelt man das Offset des naechsten SOR-Bytes bei 98H. Das darauffolgende Markierungsbyte kennzeichnet einen sog. Nullstring. Das naechste Markierungsbyte (72H) zeigt an, dass eine Integer-Zahl mit 2 Bytes Laenge folgt. Nach dieser Zahl kommt das Markierungsbyte fuer eine 4 Bytes lange Gleitkommazahl (73H), nach dieser Gleitkommazahl steht das Markierungsbyte 74H fuer eine 8 Bytes lange Gleitkommazahl mit doppelter Genauigkeit. Ab dieser Stelle (Offset 0ABH) wurde nichts mehr in diesem File geschrieben, alle nun folgenden Daten sind nicht mehr gueltig.

Es wurde nun erlaeutert, wie die Daten in dem File abgespeichert sind. Dies erfolgt in allen "MU" Files in der gleichen Art. Jetzt soll erklaert werden, wie die Daten von diesem File gelesen werden mit GET-Statements. Beenden Sie dazu den SUPERZAP und starten Sie das BASIC. Geben Sie folgende

Statements direkt ein: CLEAR 50 und NEW, geben Sie dann folgendes Programm ein:

```
10  PRINT LOC(1)$; "$ EOF TEST    "; LOC(1)%; "% EOF RBA"
20  PRINT LOC(1)!; "! NEXT RCD RBA    ";
30  IF LOC(1)! = 0 THEN PRINT ELSE PRINT LOC(1)#; "# REMRA"
```

Der Zweck dieses Programms ist es, die Positionierungswerte des erzeugten Files anzuzeigen. Das erste Zeichen des Strings entspricht jeweils dem Zusatz der LOC-Funktion, deren Wert ausgegeben wird. Der Text entspricht der Bedeutung dieser LOC-Funktion. Retten Sie dieses Programm ebenfalls auf der Floppy, da es bei allen anderen Filearten ebenfalls benoetigt wird. Sie sparen sich zusaetzliche Arbeit damit.

Eroeffnen Sie jetzt den File, indem Sie das direkte Statement

```
OPEN"I",1,"MU/DAT","MU"
```

eingeben. Starten Sie das Programm durch Eingabe von GOTO 10 (nicht RUN, da RUN alle geoeffneten Files schliesst !). Es wird dann folgender Text ausgegeben:

```
0 $ EOF TEST   171 % EOF RBA
0 ! NEXT RCD RBA
```

Beachten Sie, dass der REMRA-Wert nicht ausgedruckt wird. Er wurde vom System noch nicht gesetzt und ist deshalb ungueltig. Das eingegebene Programm ist nicht gerade sehr komfortabel, es prueft nur, ob der Wert fuer den naechsten Record 0 ist, wenn ja, ist der REMRA-Wert ungueltig.

Lesen Sie jetzt den naechsten Record mit dem folgenden direkten Statement:

```
GET 1,,,A$,B$; : PRINT A$,B$ : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
ABCDEF    2nd STRING
0 $ EOF TEST   171 % EOF RBA
19 ! NEXT RCD RBA   0 # REMRA
```

Zu beachten ist, dass die 2 Werte fuer EOF sich nicht geaendert haben, dafuer jedoch der Wert fuer den naechsten Record. Der Wert REMRA ist jetzt gueltig und hat den Wert 0. Es sei daran erinnert, dass in REMRA der RBA-Wert des letzten Records steht, ausgenommen dann, wenn ein File eroeffnet oder mit I#RBA zugegriffen wurde.

Lesen Sie jetzt den gleichen Record nochmals mit einem FP-Wert, der den File erneut auf den zuletzt gelesenen Record mit dem REMRA-Wert positioniert. Indem die Variablennamen in umgekehrter Reihenfolge angegeben werden, koennen Sie nachweisen, dass dieser Record auch wirklich ein 2. Mal gelesen wurde. Geben Sie folgendes Statement ein:

```
GET 1,#,,B$,A$; : PRINT A$,B$ : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
2nd STRING   ABCDEF
0 $ EOF TEST   171 % EOF RBA
19 ! NEXT RCD RBA   0 # REMRA
```

Wie schon zuvor haben sich die zwei Werte fuer EOF nicht geaendert. Sie duerfen sich auch nicht bei allen Zugriffen auf diesen File aendern. Folgende Transaktion wurde jetzt vom

System ausgefuehrt: der REMRA-Wert wurde in den Wert fuer NEXT RECORD ADDRESS geschrieben, die Daten wurden gelesen und in die angegebenen Variablen geschrieben und der File auf das SOR-Byte des naechsten Records positioniert.

Ignorieren Sie jetzt den 2. Record, er enthaelt nur 120 '*' und '0123456789', uebergehen Sie ihn durch Eingabe von

```
GET 1,,,; : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
0 # EOF TEST   171 % EOF RBA
152 ! NEXT RCD RBA   19 # REMRA
```

In diesem Fall wurde nichts besonderes gemacht. Durch das GET-Statement ohne FP-Parameter wurde der File auf den naechsten Record positioniert, der Wert RBA des naechsten Records wurde in REMRA geschrieben. Da keine Variablen angegeben wurden im GET, hat auch kein Datentransfer stattgefunden, der Record wurde nur durchsucht nach dem naechsten SOR-Byte.

Probieren Sie jetzt einmal den teilweisen Record I/O (Partial Record I/O) aus. Fangen Sie an, indem Sie nur den String des 3. Records lesen, und den File beim 2. Wert des 3. Records (die Integer-Zahl) positioniert lassen. Geben Sie folgendes ein:

```
GET 1,,,A$; : PRINT A$ : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
(Leerzeile)
-1 # EOF TEST   171 % EOF RBA
171 ! NEXT RCD RBA   152 # REMRA
```

Auf mehrere Dinge sollte an dieser Stelle hingewiesen werden. Es wurde jetzt der letzte Record eines Files bearbeitet. Dies wird zwar vom System nicht mitgeteilt, kann aber ueber LOC(FAN)# abgefragt werden. Bei normaler, sequentieller Aufbereitung von Daten wird das EOF als Funktion der GET-Routine abgefragt und in die End-of-Data-Routine verzweigt. Gibt es diese Routinen nicht, muss vor jedem GET ueber LOC(FAN)# der EOF-Status abgefragt werden und das Entsprechende veranlasst werden.

Obwohl nur ein Teil des Records eingelesen wurde, ergibt die Funktion LOC(FAN)! den RBA-Wert von EOF. Dies ist deshalb, weil bei record-segmentierten Files immer der Wert des naechsten Records zurueckgegeben wird, bei benutzersegmentierten Files dagegen der momentane RBA-Wert. Es gibt bei record-segmentierten Files im NEWDOS/80 keine Moeglichkeit, den momentanen Stand innerhalb eines Records abzufragen. Ebenfalls kann dies auch nicht vom REMBA-Wert erfragt werden. Man kann hoechstens versuchen, ueber einige PEEK's an diesen Wert heranzukommen, einen offiziellen Weg in Basic gibt es aber nicht.

Um zu zeigen, dass bis zu der Integerzahl gelesen wurde, gibt man folgendes ein:

```
GET 1,,,I; : PRINT I : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
2
-1 # EOF TEST   171 % EOF RBA
171 ! NEXT RCD RBA   152 # REMRA
```

Erinnern Sie sich daran, dass 'I' beim Schreiben in den File eine Integerzahl war und jetzt als Gleitkommazahl eingelesen wurde. Das Vertauschen von Variablen ist solange gestattet, wie es auch im LEVEL II BASIC erlaubt war.

Lesen Sie jetzt nochmals die Integer- und die Gleitkommazahl mit der Position REMRA vom letzten GET ein mit:

```
GET 1,0,,K,J; : PRINT J;K : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
 4  2
-1 0 EOF TEST   171 % EOF RBA
171 ! NEXT RCD RBA   152 # REMRA
```

Beim letzten GET wurde der Wert REMBA auf den RBA-Wert gesetzt. Unabhaengig von den ueberlesenen oder uebertragenen Bestandteilen des Records werden diese Werte richtig gesetzt. Auch beim letzten Beispiel haben sich die Werte nicht veraendert.

Um zu zeigen, dass sich diese Werte auch nicht nach mehrmaligem Lesen veraendern, lesen Sie jetzt die Integerzahl und die Gleitkommazahl mit doppelter Genauigkeit ein:

```
GET 1,0,,J,,I; : PRINT I;J : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
 8  2
-1 0 EOF TEST   171 % EOF RBA
171 ! NEXT RCD RBA   152 # REMRA
```

Beachten Sie, dass in diesem Falle ein Teil (die Gleitkommazahl) ueberlesen wurde durch einfaches Auslassen an dieser Stelle. Wieder wurden die gelesenen Zahlen im Typ geaendert.

Versuchen Sie nun einige andere Positionierungen mit dem RBA-Wert. Geben Sie ein:

```
GET 1,!0,,A0; : PRINT A0 : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
ABCDEF
0 0 EOF TEST   171 % EOF RBA
19 ! NEXT RCD RBA   0 # REMRA
```

Die Benutzung eines speziellen RBA-Wertes durch den Programmierer, sei es durch eine Zahl wie in dem Beispiel oder durch eine Variable oder einen Ausdruck, bewirkt, dass dieser Wert in den Wert REMBA gestellt und durch das ! als naechste Byteadresse benutzt wird. Das Gleiche geschieht in der FP mit der Recordadresse, wenn # eingesetzt wird. Der Ablauf fuer beide FP-Arten ist von hier ab der Gleiche.

Versuchen Sie eine andere Positionierungsart. Geben Sie folgendes ein:

```
I = 152 : GET 1,!0I : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
0 0 EOF TEST   171 % EOF RBA
152 ! NEXT RCD RBA
BAD FILE-MODE
```

Was ist passiert ? Der REMRA-Wert wurde aufgrund des FP-Typs vom System als falsch erkannt. Das System macht nicht mehr als die Pointer fuer den naechsten Record zu setzen. Es findet keine Datenuebertragung statt. Aus diesem Grund kann der Wert REMRA nicht gesetzt werden.

Versuchen Sie es nochmals, aber mit einer der Uebertragung vorgesetzten Positionierung. Geben Sie ein:

```
GET 1,!019 : GET 1,,,A0; : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
OUT OF STRING SPACE
```

Warum nochmals ein Fehler ? Ganz einfach: Am Anfang haben Sie CLEAR 50 eingegeben, der zu uebertragende String ist aber 130 Bytes lang. Da das NEWDOS/80 nicht einfach nur den Pointer fuer den String aendert von dem Variablenbereich in den Sektorpuffer, sondern den String in den Stringbereich uebertraegt (wie beim Statement LET), wird dieser Fehler ausgegeben.

Versuchen Sie nochmals:

```
GET 1,,,(10)A$ : PRINT A$ : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
**********
0 % EOF TEST   171 % EOF RBA
152 ! NEXT RCD RBA   19 # REMRA
```

Die 10 Sterne werden uebertragen, weil sich die Position im File durch den Fehler nicht geaendert hat. Der String wurde rechts abgeschnitten.

Als naechstes soll versucht werden, den Inhalt des Files zu veraendern, sowohl die ganzen Records als auch teilweise. Dazu muss aber der File zuerst geschlossen werden, da er nur als Inputfile im OPEN deklariert wurde. Geben Sie ein:

```
CLOSE : OPEN"R",1,"MU/DAT","MU" : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
0 % EOF TEST   171 % EOF RBA
0 ! NEXT RCD RBA
```

Diese Ausgabe ist die gleiche wie beim letzten OPEN, nur dass der File jetzt veraendert werden kann. Soll der File nur vor EOF veraendert werden, kann man in OPEN auch "D" angeben.

Zuerst soll der 1. Record mit einem einzelnen String ueberschrieben werden. Geben Sie ein:

```
PUT 1,,,"RECORD REPLACED" : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
0 % EOF TEST   171 % EOF RBA
19 ! NEXT RCD RBA   0 # REMRA
```

Beachten Sie, dass die 2 Werte von EOF sich nicht geaendert haben, es wurde nur der Wert fuer den naechsten Record neu eingesetzt. 19 ist die Byteadresse des naechsten SOR-Zeichens (Start Of Record). REMRA wurde jetzt auf gueltig gesetzt und enthaelt 0. Zu Ihrer Erinnerung: REMRA ist die Byteadresse des zuletzt bearbeiteten Records und wird beim OPEN oder beim Zugriff mit !#RBA auf ungueltig gesetzt.

Veraendern wir jetzt den 3. Record mit der doppelt genauen Gleitkommazahl und ersetzen wir diese 3 Mal durch ihr Komplement. Geben Sie ein:

```
I=152 : GET 1,!I,,,,,
GET 1,#,,D#: PUT 1,#,,3#-D#: GOTO 10
```

Das System gibt jetzt folgendes aus:

```
-1 # EOF TEST   171 % EOF RBA
171 ! NEXT RCD RBA   152 # REMRA
```

Auch in diesem Fall haben sich der EOF-Wert nicht veraendert, der File ist auf den naechsten Record positioniert, dieser steht auf EOF. Man kann aus diesem Grunde zwar nicht mehr lesen, aber es kann geschrieben werden.

Um dies zu demonstrieren geben Sie ein:

```
PUT 1,,,"THIS IS THE FOURTH RECORD"; : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
 0 $ EOF TEST   198 % EOF RBA
198 ! NEXT RCD RBA   171 # REMRA
```

Man kann sehr leicht sehen, dass der File erweitert wurde. Beachten Sie aber auch, dass die neuen Werte noch nicht in das Directory eingesetzt wurden. Sollt jetzt ein Stromausfall kommen, waere danach der File so, wie er mit 'R' eroeffnet wurde. Dies kommt daher, dass das System noch keinen Grund hatte, den Buffer zurueckzuschreiben. Die Daten koennten jetzt dadurch gesichert werden, indem man ein PUT mit dem FP-Wert '&' eingibt. Dadurch wuerde zwar der Buffer zurueckgeschrieben, nicht jedoch der neue EOF-Wert. Dieser wird erst beim CLOSE upgedatet.

Gehen wir jetzt zurueck zum zweiten Record und ersetzen da das einzelne grosse Bestandteil durch mehrere kleine. Dies machen wir durch mehrere PUT's. Geben Sie ein:

```
PUT 1,!19,,"ITEM 1",3.14159#2;
PUT 1,#,,"ITEM 3",4,10D2;
PUT 1,#,,"LAST ITEM RECORD 2"; : GOTO 10
```

Das System gibt jetzt folgendes aus:

```
 0 $ EOF TEST   198 % EOF RBA
152 ! NEXT RCD RBA   19 # REMRA
```

Wie schon zuvor hat der Recordpointer die RBA des naechsten Records, und REMRA hat die RBA des zuletzt bearbeiteten Records.

Um zu zeigen, dass der Record veraendert wurde, geben Sie ein:

```
GET 1,#,,A$,I,B#,J,K,C#; : PRINT A$,B#,C#,I,J,K : GOTO 10
```

Das System gibt folgendes aus:

```
ITEM 1    ITEM 3   LAST ITEM IN RECORD 2
6.28318  4  1000
 0 $ EOF TEST   198 % EOF RBA
152 ! NEXT RCD RBA   19 # REMRA
```

Falls wir jetzt ein GET versuchen wuerden mit dem aktuellen Pointerstand, wuerden wir einen 'RECORD OVERFLOW' gemeldet bekommen. Wir koennten solange Daten in diesen Record schreiben, solange die Orginallaenge von 131 Bytes nicht ueberschritten wird.

Das Einzige, was noch zu tun bleibt, ist den EOF-Wert auf den aktuellen Stand zu bringen. Geben Sie einfach ein:

```
CLOSE
```

Starten wir wieder den SUPERZAP (CMD"S" oder CMD"SUPERZAP"). Die SOR-Bytes stehen auf den Displacements 0, 13, 98 und AB. Schauen wir uns den 1. Record einmal genauer an. Das Markierungsbyte des Strings (8FH) zeigt an, dass es sich um einen String von 15 Bytes Laenge handelt. Wenn wir jetzt 0FH zum Displacement addieren, erhalten wir 11H, an dieser Stelle stehen 2 Bytes mit 0. Dies sind die Fuellzeichen, die beim GET ueberlesen wurden. Haetten wir versucht, 2 Strings aus dem ersten Record zu lesen, haetten wir den Fehler RECORD OVERFLOW erhalten.

Im 2. Record, der beim Displacement 13H beginnt, finden Sie das SOR-Byte, das gefolgt ist von einem Markierungsbyte, das einen

String von 6 Bytes Laenge kennzeichnet. Wenn Sie weiterzaehlen auf das 7. Byte nach dem Markierungsbyte, finden Sie eine andere Markierung fuer einen einzelnen Gleitkommawert. 5 Bytes weiter steht die Markierung fuer einen weiteren, 6 Bytes langen String. Weitere 7 Bytes weiter steht das Markierungsbyte fuer eine Integerzahl, 3 Bytes danach fuer eine Zahl mit doppelter Genauigkeit. 9 Bytes weiter steht das Markierungsbyte des letzten Bestandteils des Records, die 8AH kennzeichnet einen 10 Bytes langen String. Der Rest des Records bis A8H ist aufgefuellt mit Fuellzeichen. Sollt es notwendig sein, den 2. Record durch foellig neue Daten zu ersetzen, kann der neue Inhalt incl. SOR-Byte 131 Bytes lang sein. Mit dem richtigen FP-Wert wird alles hier ueberschrieben.

Der letzte Teil des Records braucht wohl nicht mehr in allen Einzelheiten erlaeutert zu werden. Der einzige Unterschied in seinem Inhalt vorher und jetzt ist der, dass auf dem Displacement 0A2h jetzt eine Gleitkommazahl doppelter Genauigkeit steht. Ferner beginnt jetzt ein 4. Record mit dem Displacement 0ABH und endet bei 0C5H.

Leider koennen hier nicht alle Moeglichkeiten des "MU"-Files gezeigt werden. Mit zunehmendem Gebrauch werden Sie aber diesen Filetyp besser kennenlernen und auch sehen, welche Moeglichkeiten er noch bietet.

A.11 "MF" FILES

Nachdem Sie jetzt (hoffentlich) erfolgreich mit dem "MU" Filetyp gearbeitet haben, ist fuer Sie wohl ersichtlich, welche Moeglichkeiten Sie im NEWDOS/80 mit den neuen Filetypen haben. Versuchen Sie es jetzt mit dem Typ "MF". Wie bereits in der technischen Beschreibung des NEWDOS/80 steht, ist ein MF-File aus markierten Records von gleicher Laenge aufgebaut. Die Recordlaenge wird als LRECL im OPEN-Statement angegeben.

Wie der MU-File kann auch der MF-File veraendert werden, und zwar entweder als Bestandteil eines Records oder Record fuer Record. Die neuen Daten muessen nicht unbedingt den gleichen Typ haben wie die alten Daten, ebensowenig wie sie die gleiche Laenge haben muessen wie vorher. Der Update kann sowohl von der Mitte eines Records aus erfolgen als auch vom Beginn aus. Beachten Sie aber, dass alles vom Beginn des Updates bis zum Recordende neu geschrieben wird.

Fuer den Test des MF-Files benutzen wir die gleiche Technik wie fuer den MU-File. Schreiben Sie erst das nachfolgende Programm und speichern Sie es fuer spaeter auf der Floppy.

```
10  OPEN"O",1,"MF/DAT","MF",20
20  PUT 1,,,"STRING1","STR 2","STR3";
30  PUT 1,,,"MAXIMUM STRING (19)";
40  I! = 4 : I# = 8 : I% = 2;
50  PUT 1,,,I#,I!,I%;
60  PUT 1,,,I#*10,I!*100,I%*1000;
70  CLOSE
```

Starten Sie jetzt das Programm, damit ein MF-File erzeugt wird. Starten Sie jetzt den SUPERZAP (CMD"SUPERZAP") und schauen Sie sich ueber DFS den Sektor 0 dieses Files an. Wie Sie gleich feststellen werden, ist am Beginn des Record kein SOR-Zeichen. Diese SOR-Zeichen gibt es nur im MU-File, da alle anderen Filetypen eine feste Recordlaenge haben und das System damit berechnen kann, wo der naechste Record anfaengt. Im ersten Byte des Records ist das Markierungsbyte fuer einen 7 Byte langen String. Auf dem Displacement 8 ist ein Markierungsbyte fuer einen 5 Byte langen String, und auf Displacement 0E ist das Markierungsbyte fuer einen String mit 4 Byte Laenge. Auf dem naechsten errechneteten Displacement 13 steht jedoch kein Markierungsbyte mehr, sondern 00. Es wurden insgesamt 19 Bytes geschrieben, als Recordlaenge wurden jedoch 20 Bytes vereinbart. Das System fuellt automatisch mit Nullen auf. Die Markierungsbytes muessen mitgezaehlt werden !

Der 2. Record beginnt auf Displacement 14. Hier steht ein Markierungsbyte fuer einen 19 Byte langen String (13H). Der 3. Record beginnt bei Displacement 28, Die Markierungsbytes stehen auf 28, 31 und 36 fuer eine Gleitkommazahl mit doppelter Genauigkeit, eine normale Gleitkommazahl und eine Integerzahl. Der Record hat eine beschriebene Laenge von 17 Bytes, die 3 Fuellzeichen stehen ab dem Displacement 39 (bis 3B). Der 4. und letzte Record hat die gleiche Struktur wie der 3. Record, seine Markierungen stehen auf 3C, 45 und 4A. Die Fuellzeichen sind auf 4D bis 4F. Die Daten ab 50 sind ungueltig. Es sind die gleichen Daten, die bereits vor unserem File in diesem Sektor standen.

Verlassen Sie wieder den SUPERZAP und laden Sie das vorher hoffentlich gesicherte Programm, das fuer alle Files benoetigt wird:

```
10  PRINT LOC(1)%; "% EOF TEST   "; LOC(1)%; "% EOF RBA"
20  PRINT LOC(1)!; "! NEXT RCD RBA   "!
30  IF LOC(1)! = 0 THEN PRINT ELSE PRINT LOC(1)%; "% REMRA"
```

Dieses Programm wird in der gleichen Art benutzt wie bei dem MU- File. Es werden in etwa die gleichen Versuche gemacht, aber nicht alle, es wird mehr auf die Unterschiede zwischen den 2 Filearten eingegangen.

Beginnen Sie wieder mit dem Eroeffnen des Datenfiles und zeigen Sie die LOC-Werte an. Geben Sie folgendes direkte Statement ein:

```
OPEN"I",1,"MF/DAT","MF",20 : GOTO 10
```

Das System antwortet mit:

```
0 % EOF TEST   80 % EOF RBA
0 ! NEXT RCD RBA
```

Ausser dem Wert EOF RBA sind die Ergebnisse die gleichen wie beim MU-File. Der File ist damit initialisiert, man kann beim 1. logischen Record, bei Displacement 0, anfangen. Geben Sie ein:

```
GET 1,,,,A%,B%; : PRINT A%,B% : GOTO 10
```

Das System antwortet mit:

```
STR 2      STR3
0 % EOF TEST   80 % EOF RBA
20 ! NEXT RCD RBA   0 % REMRA
```

Beachten Sie hierbei, dass die letzten 2 Bestandteile des 1. Records uebertragen wurden. Dies deshalb, weil der erste String nicht eingelesen wurde (anstelle des Variablennamens keine Eingabe nach dem 3. Komma).

Von der momentanen Position aus koennen wir nochmals den I. Record mit dem 1. String einlesen:

```
GET 1,#,,A%,B%; : PRINT A%,B% : GOTO 10
```

Das System antwortet mit:

```
STRING1      STR 2
0 % EOF TEST   80 % EOF RBA
20 ! NEXT RCD RBA   0 % REMRA
```

Die dritte Variable kann auch noch eingelesen werden mit:

```
GET 1,#,,C%; : PRINT C% : GOTO 10
```

Das System antwortet mit:

```
STR3
0 % EOF TEST   80 % EOF RBA
20 ! NEXT RCD RBA   0 % REMRA
```

Der '#' als Positionsangabe teilt dem System mit, die Ein- oder Ausgabe von der Stelle aus zu starten, wo es beim letzten GET oder PUT aufgehoert hat. Oder in anderen Worten: von der momentanen Fileposition, als auch innerhalb eines Records.

Versuchen Sie jetzt, den 4. Record zu lesen, ohne dass der 2. und 3. Record bearbeitet wird. Geben Sie ein:

```
GET 1,!(4-1)*20,,J,K,L: : PRINT J; K; L : GOTO 10
```

Das System antwortet mit:

```
 80  400  2000
-1 $ EOF TEST   100 % EOF RBA
100 ! NEXT RCD RBA   80 # REMRA
```

Beachten Sie den Ausdruck fuer die Fileposition. Das Ergebnis ist 60, es ist errechnet aus der Recordnummer, die verlangt wird minus 1, multipliziert mit der Recordlaenge. Die Positionierung mit dem Wert !RBA ist, wie Sie sehen koennen, sehr einfach bei den MF und FF-Files. Versuchen Sie jetzt selbst die gleiche Positionierung mit Angabe der Recordnummer (RN) anstelle von !RBA.

Versuchen wir jetzt ein paar wahlfreie Zugriffe und Updates von Records und ueberpruefen die Resultate. Eroeffenen Sie den File neu:

```
CLOSE : OPEN"R",1,"MF/DAT","MF",20 : GOTO 10
```

Das System antwortet mit:

```
 0 $ EOF TEST   80 % EOF RBA
 0 ! NEXT RCD RBA
```

Bisher also das gleiche Ergebnis wie beim OPEN auf den File als Inputfile. Ersetzen Sie jetzt den 1. Record mit dem neuen Inhalt:

```
PUT 1,,,I$: : GOTO 10
```

Das System antwortet mit:

```
 0 $ EOF TEST   80 % EOF RBA
20 ! NEXT RCD RBA   0 # REMRA
```

Diese Antwort zeigt, dass der 1. Record bearbeitet wurde. Sie muessen sich jetzt darueber im Klaren sein, dass die momtane Adresse innerhalb des Files 1 ist, obwohl bei NEXT RCD RBA 20 angezeigt wird. Sie koennen das einfach sehen, indem Sie mit der Positionierungsangabe '*' nochmals in den gleichen Record schreiben:

```
PUT 1,*,,"2",2; : GOTO 10
```

Das System antwortet mit:

```
 0 $ EOF TEST   80 % EOF RBA
20 ! NEXT RCD RBA   0 # REMRA
```

Das Ergebnis des Updates schauen wir uns spaeter an. Fuegen Sie jetzt zuerst einmal 2 weitere Records an den bestehenden File an. Geben Sie ein:

```
PUT 1,!%,,"RCD 5"; : PUT 1,,,"RECORD 6"; : GOTO 10
```

Das System antwortet mit:

```
-1 $ EOF TEST   120 % EOF RBA
120 ! NEXT RCD RBA   100 # REMRA
```

Diese Zahlen geben an, dass der File jetzt 6 Records lang ist.

Schliessen Sie den File jetzt und starten Sie wieder den SUPERZAP (CLOSE: CMD"SUPERZAP"). Schauen Sie mit der Funktion DFS den Sektor 0 des Files an. Die Records dieses Files beginnen bei den Displacements 0, 14, 28, 3C, 50 und 64 (hex.). Das Markierungsbyte auf Displacement 0 zeigt an, dass hier ein sog. Nullstring steht. Auf Displacement 1 steht ein Markierungsbyte fuer einen String mit 1 Zeichen Laenge, auf 3 wird eine Integerzahl gekennzeichnet. Der Rest des Files wurde mit 0 aufgefuellt. Der Inhalt der Records 5 und 6 bedarf an sich keiner

weiterer Erlaeuterungen. Die Daten nach dem 6. Record wurden durch das Update nicht veraendert. Das System behandelt dann zwar diesen Teil noch, als wuerden Daten darin stehen, durch die internen Pointer kann es aber nicht darauf zugreifen (spez. EOF RBA).

Obwohl dieser Teil im Gegensatz zum MU-File relativ kurz war sollten Sie nicht gleich glauben, dass der MF-File wesentlich einfacher zu handhaben ist als der MU-File. Sie koennen eigene Records aufbauen und entweder ueber die Recordnummer (RN) oder ueber die Byteadresse (!RBA) wie in unserem Beispiel ansprechen. Sind Sie vorsichtig bei der Positionierung ueber !RBA ! Wenn der angegebene Wert aus irgend einem Grunde falsch ist, beispielsweise durch unterschiedliche Recordlaengen, und Sie sind nicht genau an der Stelle, wo Sie sein sollten (z. B. Recordanfang), zerstoeren Sie sich unter Umstaenden beim naechsten PUT den File oder Sie erhalten beim naechsten GET ziemlich unmoegliche Werte.

A.12 "MI" FILES

Jetzt kommen wir zum letzten der Marked Item Files - dem MI-File. Die wichtigsten Unterschiede zu den MU- und MF-Files sind:

1. Ein MI-File kann nicht veraendert werden (kein Update).
2. Ein MI-File hat keine vom System beruecksichtbare Recordlaenge.

Auf Grund dieser Beschraenkungen kann dieser Filetyp nur fuer kompakte Referenzfiles benutzt werden, da er nur geschrieben und ggf. erweitert und spaeter gelesen werden kann. Um eine bestimmte Datengruppe lesen zu koennen, muss die Positionierung ueber IRBA erfolgen. Aus diesem Grunde sind auch die bisher benutzten Beispiele nur eingeschraenkt zu verwenden.

Beginnen Sie wieder mit dem Testprogramm zum Erzeugen des Datenfiles. Aendern Sie dazu das Beispiel vom MF-File entsprechend.

```
10 OPEN"O",1,"MI/DAT","MI"
20 PUT 1,,,"STRING1","STR 2","STR3";
30 PUT 1,,,"MAXIMUM STRING (19)";
40 I! = 4 : I# = 8 : I% = 2;
50 PUT 1,,,I#,I!,I%;
60 PUT 1,,,I#*10,I!*100,I%*1000;
70 CLOSE
```

Sichern Sie das Programm wieder auf der Floppy und starten Sie es dann. Es wird ein benutzersegmentierter Datenfile mit 73 Bytes Laenge erzeugt. Verlassen Sie das BASIC und starten Sie den SUPERZAP oder gehen Sie direkt ueber CMD"SUPERZAP" in den SUPERZAP. Schauen Sie sich ueber die Funktion DFS den Sektor 0 des eben erzeugten Datenfile an.

Wie Sie sehen, enthaelt dieser File weder SOR-Zeichen noch Fuellbytes. Es gibt keine sichtbaren Records sondern nur einen zusammenhaengenden String von Daten. Fuer die Daten im File, die Struktur und die Organisation ist ausschliesslich der Programmierer verantwortlich. Es kommt also voll auf das gute Design des Programmierers an, wie gut und effizient die Daten strukturiert und damit verfuegbar sind.

Da in dem File nur Datenbytes stehen muss man, um auf diese zuzugreifen, genau die Struktur wissen. In unserem File sind es vier Strings und sechs numerische Bestandteile. Um diese wahlfrei lesen zu koennen, muss man die genaue RBA-Adresse der Markierungsbytes kennen. Andernfalls erhaelt man, im besten Falle, die Fehlermeldung BAD FILE DATA, oder, was auch moeglich ist, man erhaelt falsche Daten.

Gehen wir schnell durch die vom SUPERZAP angezeigten Daten. Die Markierungsbytes der Strings stehen auf den Displacements 00, 08, 0E und 13. Der erste Satz der numerischen Werte hat seine Markierungsbytes auf 27, 30 und 35, der zweite Satz auf 38, 41 und 46. Wir benutzen diese Displacements, um auf diese Daten zuzugreifen. Im Moment ist die Byteadresse von EOF 49.

Gehen Sie wieder in das BASIC zurueck und laden Sie das Programm, das die verschiedenen LOC-Werte anzeigt. Damit sehen Sie, wie der MI-File vom System her unterstuetzt wird.

Zuerst muss der File wie ueblich eroeffnet werden. Geben Sie ein:

```
OPEN"I",1,"MI/DAT","MI" : GOTO 10
```

Das System antwortet mit:

```
 0 $ EOF TEST   73 % EOF RBA
 0 ! NEXT RCD RBA
```

Wie bei den anderen Filetypen wird beim Eroeffnen eines Files am Anfang positioniert, so dass das System, wenn als FP beim ersten GET nichts angegeben ist, der erste Record gelesen wird.

Um den Unterschied zum OPEN im Extent-Mode zu zeigen, geben Sie bitte ein:

```
CLOSE : OPEN"E",1,"MI/DAT","MI" : GOTO 10
```

Das System antwortet mit:

```
-1 $ EOF TEST   73 % EOF RBA
73 ! NEXT RCD RBA
BAD FILE MODE IN 30
```

Die letzte Message ist entstanden, weil das System versucht hat, den REMRA-Wert auszudrucken, obwohl er durch das OPEN auf ungueltig gesetzt wurde (das Programm versucht deshalb den REMRA- Wert auszudrucken, weil der Wert der RBA-Adresse des naechsten Records ungleich 0 ist).

Der File ist jetzt im Output-Modus. Erweitern Sie den File um drei Integer-Werte mit der Eingabe:

```
PUT 1,,,-1,-2,-3; : GOTO 10
```

Das System antwortet mit:

```
-1 $ EOF TEST   82 % EOF RBA
82 ! NEXT RCD RBA  73 $ REMRA
```

Beachten Sie hierbei, dass der EOF-Wert um 9 Bytes hoeher ist als der Orginal-Wert. Beim MI-File wird der REMRA-Wert auf den gleichen Wert gesetzt wie REMBA. Sie stehen beide an gleicher Stelle, wenn ein GET oder PUT gestartet wird.

Gehen wir jetzt zurueck um einige Werte aus dem File anzuzeigen. Geben Siew ein:

```
CLOSE : OPEN"R",1,"MI/DAT","MI"
GET 1,!19,,A$; : PRINT A$ : GOTO 10
```

Das System antwortet mit:

```
MAXIMUM STRING (19)
 0 $ EOF TEST   82 % EOF RBA
39 ! NEXT RCD RBA  19 $ REMRA
```

Die Zahl vor REMRA zeigt jetzt den Beginn der Datenuebertragung beim GET, NEXT RCD RBA zeigt die Byteadresse des ersten numerischen Wertes. Wird im naechsten GET kein FP-Wert angegeben, so beginnt dann die Datenuebertragung an dieser Stelle. Geben Sie dafuer ein:

```
GET 1,,,,J%,K$,I!; : PRINT J%;K$;I! : GOTO 10
```

Das System antwortet mit:

```
 4  2  80
 0 $ EOF TEST   82 % EOF RBA
65 ! NEXT RCD RBA  39 $ REMRA
```

Beachten Sie in diesem Fall, dass alle numerischen Werte, die hier uebertragen wurden, einen anderen Typ im IGEL haben, als sie in den File geschrieben wurden. Einer der groessten Vorteile der Marked Item Files ist die Konvertierung der numerischen Variablen.

Um zu zeigen, dass beim MI-File die Werte REMRA und REMBA gleich sind, geben Sie folgendes zweimal ein, mit unterschiedlichem FP-Parameter:

```
GET 1,#,,I,J,K; : PRINT I;J;K : GOTO 10
```

und dann:

```
GET 1,$,,I,J,K; : PRINT I;J;K : GOTO 10
```

In beiden Faellen gibt das System folgendes aus:

```
 8  4  2
 0 $ EOF TEST   82 % EOF RBA
 56 ! NEXT RCD RBA  39 # REMRA
```

Beachten Sie hierbei besonders, dass der REMRA- und REMBA-Wert bei MI- und FI-Files immer gueltig ist, bei MU-, MF- und FF-Files dagegen nur, wenn die Datenuebertragung bei GET und PUT am Anfang eines logischen Records beginnt.

Um zu zeigen, wie ein MI-File nach dem Eroeffnen im Modus "R" auch erweitert werden kann, geben Sie ein:

```
PUT 1,!%,,15,-15; : GOTO 10
```

Das System gibt folgendes aus:

```
-1 $ EOF TEST   88 % EOF RBA
 88 ! NEXT RCD RBA  82 # REMRA
```

Der EOF-Wert wurde um 6 Bytes verschoben gegenueber vorher. Der File ist jetzt so positioniert, dass er bei einer FP-Angabe von Null oder 'W' erweitert wird.

Dies waere alles, was so an Versuchen mit dem MI-File gemacht werden kann. Sie koennen selbst versuchen, den Inhalt des Files zu veraendern: Sie werden den Fehler BAD FILE MODE erhalten (lt. Spezifikation ist kein UPDATE moeglich !). Wenn Sie noch genauer wissen wollen was passiert, wenn Sie einen anderen FP-Wert im GET oder PUT benutzen und wie sich dieser bei z. B. falscher Positionierung auswirkt, geben Sie ein kurzes Testprogramm dafuer ein. Es koennen hier leider nicht alle Fehlermoeglichkeiten aufgezeigt werden.

A.13 "FF" FILES

Der Fixed Item File ist in mehreren Punkten zum Marked Item File unterschiedlich. Dies faengt bei den Markierungsbytes an. Die Beschreibung eines Records wird ausschliesslich dem IGEL entnommen und nicht dem File selbst. Wenn Sie beispielsweise im IGEL festlegen, dass ein String von 20 Bytes gelesen werden soll, so liest das System exakt diese 20 Bytes in eine Stringvariable, auch dann, wenn zuvor numerische Werte in den Record geschrieben wurden. Ebenso muessen z. B. Variable mit dem gleichen Typ gelesen werden, wie sie in den File geschrieben wurden, da sonst falsche Werte uebernommen werden, da unterschiedliche Variablentypen unterschiedliche Laengen haben.

Ein weiterer Unterschied ist der, dass Fixed Item Files ueber einen sog. echten teilweisen Filezugriff veraendert werden koennen, d. h., dass wenn ein Bestandteil eines Records veraendert wird, sind alle anderen Bestandteile dieses Records davon nicht betroffen. Beim Marked Item File muss im Gegensatz dazu jedes folgende Bestandteil eines Records nochmals geschrieben werden.

Der dritte Unterschied zu Marked Item Files ist der, dass im IGEL eines Fixed Item Files nur Variablennamen auftreten duerfen, Stringvariable mit der Laenge in () vor dem Namen. Ganze Ausdruecke bzw. Formeln duerfen nicht benutzt werden.

Fixed Item Files gibt es in zwei Arten: den FF-File, in dem alle Records die gleiche Laenge haben (LRECL), und die FI- Files, die fuer das System keine erkennbaren Records haben. Im Moment befassen wir uns nur mit den FF-Files. Genau wie bei den Marked Item Files schreiben wir erst ein kurzes Programm, in dem ein FF-File mit den noetigen Testdaten erzeugt wird. Sichern Sie auch dieses Programm auf der Floppy, falls Sie es spaeter nochmals benutzen wollen, haben Sie es leichter. Starten Sie dann das Programm.

```
10  CLEAR 100
20  OPEN,"O",1,"FF/DAT","FF",20
30  PUT 1,%,40 : GOTO 50
40  (20)I$;
50  LSET I$ = "ABCDEFGHIJK"
60  PUT 1,,,(20)I$;
70  LSET I$ = "12345678901234567890"
80  PUT 1,,,(20)I$;
90  I% = 2 : I! = 4 : I# = 8
100 PUT 1,,,(4)I$,I%,I!,I#;
110 I% = I%*10 : I! = I!*100 : I# = I#*1000
120 PUT 1,,,(4)I$,I%,I!,I#;
130 CLOSE
```

Wie Sie sicher bemerkt haben, ist dieses Programm etwas anders als die bisherigen Programme fuer Marked Item Files. Zum ersten werden die Strings nur von Variablen im IGEL geschrieben und stehen nie direkt im IGEL. Weiter werden keine Ausdruecke (Formeln) im IGEL benutzt, sondern die Werte immer vorher berechnet. Bei der Wertzuweisung zu I$ wurde LSET benutzt anstelle von LET. Dies wurde getan, um die urspruengliche Laenge von I$ zu erhalten, die in der PSEUDO-FIELD-Operation in Statement 30 und 40 gesetzt wurde. Diese PSEUDO-Operation wird allerdings nicht unbedingt benoetigt. Sie kennzeichnet mehr oder weniger nur diese Variable als einen String mit 20 Bytes Laenge. Ferner belegt diese Operation den Platz im Stringbereich von Basic und wuerde damit noch vor den GET's und PUT's melden, wenn nicht genuegend Platz zur Verfuegung steht.

Es ist besser, wenn gleich gemeldet wird, dass nicht genuegend Platz zur Verfuegung steht, als wenn dies kurz vor Ende eines Programmes geschieht. Beachten Sie auch, dass alle Stringvariablen die zu uebertragende Laenge in Klammern vorgesetzt haben. Dieser Wert besagt, wieviele Bytes uebertragen werden, nicht die PSEUDO-Operation. Es sei nochmals daran erinnert, dass die Strings entweder rechts aufgefuellt oder, wenn sie zu lang sind, rechts abgeschnitten werden.

Starten Sie jetzt den SUPERZAP und schauen Sie mit der Funktion DFS den ersten Sektor des eben geschriebenen Files an. Sie sehen, dass der erste Record (14H Bytes lang) aus 9 Datenbytes besteht und nach rechts mit Leerzeichen aufgefuellt wurde. Das Auffuellen mit Leerzeichen wurde von LSET gemacht, nicht von PUT. Der 2. Record enthaelt 20 Datenbytes ohne Leerzeichen. Der geschriebene String war 20 Bytes lang - waere er laenger gewesen, so haette ihn das LSET abgeschnitten. Der 3. und der 4. Record haben das gleich Format: zuerst 4 Stringzeichen, dann eine 2 Byte lange Integerzahl, eine 4 Byte lange Gleitkommazahl, eine 8 Byte lange Gleitkommazahl mit doppelter Genauigkeit und 2 Fuellzeichen. Zu erwaehnen waere hier, dass der Record keine Markierungsbytes fuer den Typ von Variablen enthaelt. Der EOF- Wert ist 80 (hex. 50). Alle folgenden Bytes in diesem Sektor wurden nicht vom Programm veraendert.

Laden Sie jetzt wieder das BASIC-Programm, das die verschiedenen Positionen ueber die LOC-Funktion anzeigt. Es kann sowohl bei den Marked als auch bei den Fixed Item Files benutzt werden.

Geben Sie das folgende direkte Statement ein um zu sehen, wie die Positionierung nach dem OPEN ist:

```
OPEN"I",1,"FF/DAT","FF",20 : GOTO 10
```

und Sie erhalten:

```
 0 @ EOF TEST  80 % EOF RBA
 0 ! NEXT RCD RBA
```

Wie erwartet steht die Positionierung auf dem naechsten (ersten) Record des Files.

Um den ersten Record zu lesen geben Sie ein:

```
GET 1,,,(20)A@; : PRINT LEN(A@); A@ : GOTO 10
```

und Sie erhalten:

```
 20 ABCDEFGHIJK
 0 @ EOF TEST  80 % EOF RBA
 20 ! NEXT RCD RBA  0 @ REMRA
```

Wie Sie sehen, wurden 20 Zeichen in die Variable uebertragen, die im IGEL stand. Genauso einfach haette man auch nur einen Teil dieses Records uebertragen koennen.

Nur um zu zeigen wie das funktioniert, nehmen wir an, dass dieser Record aus 3 je 6 Bytes langen Bestandteilen aufgebaut ist, die mit 3 verschiedenen GET's gelesen werden. Dafuer muessen natuerlich verschiedene FP-Werte benutzt werden. Geben Sie ein:

```
GET 1,@,,(6)A@; : GET 1,*,,(6)B@; : GET 1,*,,(6)C@;
PRINT LEN(A@);A@ : PRINT LEN(B@);B@ : PRINT LEN(C@);C@
```

Das System gibt aus:

```
6  ABCDEF
6  IJK
6
```

geben sie dann noch ein:

```
GOTO 10

0 ! EOF TEST  80 % EOF RBA
20 ! NEXT RCD RBA  0 ! REMRA
```

Wir haben hier 3 Felder aus dem gleichen Record gelesen mit der noetigen Anzahl von GET-Statements. Dies zeigt, welche Freiheiten man bei der teilweisen Record- Ein- und Ausgabe hat.

Eine andere Freiheit haben Sie beim Ueberlesen bzw. Ignorieren von Zeichen im Record um dann den Teil zu lesen, den Sie benoetigen. Geben Sie ein:

```
GET 1,,,(12)!,(4)A$; : PRINT LEN(A$); A$ : GOTO 10
```

Das System gibt aus:

```
4  3456
-1 ! EOF TEST  80 % EOF RBA
40 ! NEXT RCD RBA  20 ! REMRA
```

Die 12 ignorierten Bytes haetten ebensogut wie 12 Zeichen eines ASCII-Strings auch beispielsweise 6 Integer-Zahlen sein koennen. Die Ursache ist, dass das System weder den Inhalt der ignorierten Bytes kennt noch ihn beachtet, es weiss nur, dass (len) Bytes zu ueberlesen sind.

Machen wir nun absichtlich einen kleinen Fehler beim Lesen des 4. Records: vergessen wir einmal, dass wir einen 4 Bytes langen String an den Anfang gestellt haben. Geben Sie ein:

```
GET 1,4,,I%,I!,I$; : PRINT I%; I!; I$ : GOTO 10
```

Das System gibt aus:

```
12849  0  0
-1 ! EOF TEST  80 % EOF RBA
80 ! NEXT RCD RBA  60 ! REMRA
```

Das Ergebnis ist bestimmt nicht das, was wir in den File geschrieben haben. Im Gegensatz zum Marked Item File, wo ein Fehler entdeckt und gemeldet wird, wird im Fixed Item File einfach alles uebertragen, was gerade auf der momentanen Position steht. Eine Ueberpruefung kann nicht stattfinden. Der Grund fuer die Werte 0 war der, dass sowohl der Exponent der Gleitkommazahl als auch der Gleitkommazahl in doppelter Genauigkeit 0 war.

Der File ist jetzt auf dem EOF-Wert positioniert (bei einem Zugriff mit FP=Null). Genaugenommen steht der File auf dem 15. Byte des 4. Records. Die Funktion LOC(FAN)! ergibt den Wert des naechsten sequentiellen Records.

Um zu zeigen, dass wir tatsaechlich auf dem 15. Byte stehen, geben Sie ein:

```
FOR I=1 TO 5 : GET 1,#,,(1)A$; : PRINT ASC(A$); : NEXT
```

Das System antwortet mit:

```
0  0  122  141  0  0
```

Wenn Sie die erhaltenen Dezimalwerte in Hexadezimal umwandeln, erhalten Sie die Werte fuer die Mantisse und das Byte des Exponenten der Gleitkommazahl mit doppelter Genauigkeit, die urspruenglich in den 4. Record geschrieben wurde.

Gehen wir noch einmal zurueck und lesen den 4. Record korrekt ein. Geben Sie ein:

```
GET 1,4,,(4)A$,I%,I!;I#; : PRINT I#;I%;I!;I# : GOTO 10
```

Das System gibt aus:

```
 1234  20  400  8000
-1 # EOF TEST  80 % EOF RBA
80 ! NEXT RCD RBA  60 # REMRA
```

Es hat jetzt den Record so gelesen, wie er geschrieben wurde. Der 4. Record wurde jetzt durch Angabe der Recordnummer (RN) ein zweites Mal gelesen. Beim Erzeugen von Indices fuer Fixed Record Files (FF, MF und Field Item Files) koennen Sie erheblich Platz sparen durch Verwendung der Recordnummern, die nur in Integer angegeben werden, anstelle der RBA-Adresse der LOC-Funktion, die im Gleitkommaformat zurueckgegeben wird. Es macht keinen Unterschied, ob jetzt mit Recordnummer oder RBA-Adresse auf einen Record zugegriffen wird.

Schliessen wir jetzt den File und eroeffnen wir ihn neu fuer Updates. Geben Sie ein:

```
CLOSE : OPEN"R",1,"FF/DAT","FF",20 : GOTO 10
```

Das System gibt aus:

```
0 # EOF TEST  80 % EOF RBA
0 ! NEXT RCD RBA
```

Konzentrieren wir uns in diesem Abschnitt auf die teilweise Ein-/ Ausgabe eines Records. Hier hat der Fixed Item File gegenueber dem Marked Item File enorme Vorteile. Nehmen wir jetzt an, dass der erste und der zweite Record gleiches Format haben: 4 Bestandteile zu je 5 Bytes Laenge. Machen wir jetzt ein Update auf das 2. Bestandteil des ersten Records und das letzte Bestandteil des zweiten Records. Geben Sie ein:

```
A# = "2ND" : PUT 1,1,,((2-1)*5)#,(5)A#; : GOTO 10
```

Das System antwortet mit:

```
0 # EOF TEST  80 % EOF RBA
20 ! NEXT RCD RBA  0 # REMRA
```

Veraendern wir jetzt den 2. Record. Geben Sie ein:

```
A# = "LAST" : PUT 1,2,,((4-1)*5)#,(5)A#; : GOTO 10
```

Das System antwortet mit:

```
0 # EOF TEST  80 % EOF RBA
40 ! NEXT RCD RBA  20 # REMRA
```

In beiden Faellen waere eine Angabe der Recordnummer nicht noetig gewesen. Es wurde in den Beispielen nur fuer die bessere Verstandlichkeit benutzt. Wie Sie hier an diesem Beispiel sehen koennen, ist ein Update sehr leicht durchzufuehren, auch wenn ein File aus mehreren hundert Records besteht. Wie Sie gesehen haben, wurde die genaue Position des zu aendernden Bestandteils innerhalb des Records durch Errechnen der zu ueberspringenden Bytes gefunden. Beim zweiten Record wurden diese zu ueberspringenden Bestandteile nicht nur uebersprungen, sondern auch auf Null gesetzt. Wir werden das spaeter sehen.

Veraendern wir jetzt die Integer-Werte des 3. und des 4. Records. Geben Sie dazu ein:

```
I% = -50 : PUT 1,3,,(4)0,I%; : PUT 1,4,,(4)0,I%;
GET 1,3,,(4)J0,J%,J1,J0; : PRINT J0;J%;J1;J0 : GOTO 10
```

Das System antwortet mit:

```
 1234  -50  4  8
 0 0 EOF TEST  80 % EOF RBA
 60 ! NEXT RCD RBA  40 0 REMRA
```

Die erste Zeile der Antwort zeigt, dass der Update nur das erste Feld geaendert hat - alle anderen Bestandteile waren nicht betroffen. Bei einem MI oder MU-File mit gleichen Records (mit Markierungsbytes) haette man die Integer-, Gleitkommazahl und Gleitkommazahl mit doppelter Genauigkeit im IGEL mit angeben muessen, um nur die Integerzahl zu aendern. Ferner haette man in diesem Fall zuerst den Record lesen muessen, um die momentanen Werte dieser Zahlen zu erhalten. Hier im FF-File jedoch musste man nur die nicht gewuenschten Positionen ueberlesen, die nicht geaendert werden sollen, und das zu aendernde Bestandteil zurueckschreiben.

Im Zugriffsmodus "R" kann der File auch erweitert werden. Um dies zu zeigen, geben Sie ein:

```
PUT 1,6,,J%,J%,J%; : GOTO 10
```

Das System antwortet mit:

```
-1 0 EOF TEST  100 % EOF RBA
100 ! NEXT RCD RBA  80 0 REMRA
```

In diesem Beispiel wurde der 5. Record einfach uebergangen. Das System erzeugt bei dieser Fileart einen 5. Record und fuellt ihn mit Nullen. Erst dann wird der 6. Record geschrieben.

Schliessen Sie diesen File jetzt und starten Sie den SUPERZAP mit der DFS-Funktion. Lassen Sie sich den ersten Sektor anzeigen. Sie werden feststellen, dass die zwei String-Records 1 und 2 veraendert wurden, dass die 2 Integerwerte im 3. und 4. Record auf CEFFH veraendert wurden, dass der 5. Record (Displacement 50H bis 63H) nur Nullen enthaelt, und dass der 6. Record 3 Mal CEFFH, gefolgt von 14 Nullen enthaelt.

A.14 "FI" FILES

Kommen wir jetzt zum letzten der NEWDOS/80 Files, dem FI-Typ. Wie schon der MI-File ist der FI-File benutzersegmentiert. Er ist jedoch, wie der FF-File, ein Fixed Item File, hat also keine Markierungsbytes. Aus diesem Grunde kann auch, im Gegensatz zum MI-File, ein Update erfolgen. Dadurch kann man ihn leichter anwenden und auch mehr mit ihm anfangen.

Wie bisher mit allen Filetypen, erzeugen wir auch hier ein kleines Testprogramm, um diesen File zu erzeugen. Geben Sie folgende Statements ein und starten Sie dann das Programm (ev. vorher sichern):

```
10  OPEN"O",1,"FI/DAT","FI"
20  A$ = "1ST STRING" : B$ = "STR 2"
30  I% = 2 : I! = 4 : I# = 8
40  PUT 1,,,(15)A$,(6)B$,I%,I!,I#
50  I% = I%*-1000 : I! = I!*-100 : I# = I#*-10
60  PUT 1,,,(15)B$,(6)A$,I%,I!,I#
70  CLOSE
```

Das erste, was Ihnen vielleicht aufgefallen ist, ist die Tatsache, dass in diesem Testprogramm keine PSEUDO-FIELD-Operation (FP=%) durchgefuehrt wird. Bei diesem Filetyp ist dies absolut unnoetig. BASIC legt die Strings an, wie sie gerade gebraucht werden, und das PUT/GET erweitert oder schneidet die Strings ab, so wie sie benoetigt werden und wie es in der Laengenangabe der Stringvariablen im IGEL steht.

Das zweite ist, dass in den PUT-Statements beide Datengruppen das gleiche Format haben: einen 15 Byte langen String, einen 6-Byte-String, einen Integer-Wert, eine Gleitkommazahl und eine Gleitkommazahl mit doppelter Genauigkeit. Bei groesseren Datenmengen ist es auf jeden Fall sinnvoll, allen Records die gleiche Laenge und Struktur zu geben.

Laden Sie wieder, falls noch vorhanden, das Programm mit den verschiedenen LOC-Funktionen. Es sei hier noch einmal aufgefuehrt:

```
10  PRINT LOC(1)$; "$ EOF TEST   "; LOC(1)%; "% EOF RBA"
20  PRINT LOC(1)!; "! NEXT RCD RBA   ";
30  IF LOC(1)! = 0 THEN PRINT ELSE PRINT LOC(1)#; "# REMRA"
```

Geben Sie ein:

```
OPEN"R",1,"FI/DAT","FI" : GOTO 10
```

Das System antwortet mit:

```
0 $ EOF TEST   70 % EOF RBA
0 ! NEXT RCD RBA
```

Wie erwartet ist der File positioniert, dass mit dem ersten GET oder PUT der erste Record bearbeitet wird, falls nicht ein besonderer FP-Wert benutzt wird. Wie Sie hier sehen, gilt diese Aussage fuer alle Filetypen, ausser dann, wenn ein File mit "E" eroeffnet wird, wobei der File auf den EOF RBA positioniert wird.

Da wir die Struktur der zwei geschriebenen Records genau kennen ist es einfach, direkt auf den 2. Record zuzugreifen ueber die RBA Positionierung. Geben Sie ein:

```
GET 1,!35,,(15)A1$,(6)A2$,I%,I!,I#;
PRINT A1$,A2$,I%;I!;I# : GOTO 10
```

Die Antwort des Systems:

```
STR       1ST ST      -2000  -400  -80
-1 # EOF TEST   70 % EOF RBA
70 ! NEXT RCD RBA  35 # REMRA
```

Da alle Daten im 2. String abgehandelt wurden ist der File jetzt auf EOF positioniert, wie man an der Funktion LOC(FAN)# erkennen kann.

FI-Files koennen auf die gleiche Art erweitert werden wie FF-Files. Machen wir das einmal und lassen dabei 10 Bytes frei zwischen dem momentanen EOF und den neuen Daten. Geben Sie ein:

```
L = LOC(1)! : J = EXP(1) : PUT 1,,,(10)#,J! : GOTO 10
```

Die Antwort vom System ist:

```
-1 # EOF TEST   84 % EOF RBA
78 ! NEXT RCD RBA  70 # REMRA
```

Die Variable L enthaelt die Byteadresse, von der ab geschrieben wurde (Fuellzeichen). Wir werden diese Variable gleich noch benutzen. Beachten Sie, dass bei der Wertzuweisung der Variablen J diese keinen Zusatz fuer ihren Typ hatte (!, % oder #). Dies war deshalb, weil wir den Defaulttyp nicht geaendert haben (Gleitkomma). Dieser Zusatz wurde jedoch im IGEL angegeben (!), da hier fuer Fixed Item Files der Typ stehen muss, um dem System mitzuteilen, wie lange diese Variable ist. Haette an dieser Stelle nur ein J gestanden, haette das System nicht gewusst, welche von den Variablen (J!, J% oder J#) es haette schreiben sollen. In diesem Punkt ist eine Abweichung vom BASIC vorhanden.

Gehen wir aber zurueck und schreiben etwas in die 10 Fuellzeichen. Wir benutzen dazu die RBA-Positionierung L vom letzten PUT. Geben Sie ein:

```
A# = "ABCDEFG" : PUT 1,!L,,(4)A#,L!; : GOTO 10
```

Die Antwort vom System ist:

```
0 # EOF TEST   84 % EOF RBA
78 ! NEXT RCD RBA  70 # REMRA
```

Beachten Sie besonders die Leerzeichen im letzten GET-Statement. Es zeigt Ihnen, welche Freiheiten Sie diesbezueglich haben. Richtig angewandt, koennen einige Leerzeichen ein Programm wesentlich uebersichtlicher machen.

Lesen wir jetzt einige der Daten, die in den File geschrieben wurden. Geben Sie ein:

```
GET 1,!58,,I!,I#,(4)B$,K!;
PRINT I!; I#; B$; K! : GOTO 10
```

Das System antwortet mit:

```
-400  -80  ABCD  70
0 $ EOF TEST   84 % EOF RBA
78 ! NEXT RCD RBA  58 # REMRA
```

Wir lesen die Daten mit dem richtigen Datentyp, mit dem sie auch in den File geschrieben wurden. Aus diesem Grunde sind die Werte auch die gleichen wie beim Schreiben. Waere die Position auch nur um ein Byte verschoben gewesen, haetten wir ganz andere Ergebnisse erhalten.

Um dies zu zeigen, verschieben wir die Positionierung einmal um ein Byte nach unten und lesen damit ein. Geben Sie ein:

```
GET 1,!57,,I!,I$,(4)B$,K!;
PRINT I!; I$; B$; K! : GOTO 10
```

Das System antwortet mit:

```
 2.36125E+21  2147483648.000008 xABC 6.01858E-36
 0 $ EOF TEST   88 % EOF RBA
 77 ! NEXT RCD RBA  57 # REMRA
```

Hoffnungslos, oder ? Aber das System hat nichts anderes getan als das, was wir ihm gesagt haben. Eine Fehlermeldung kann bei einem FI-File nicht erfolgen, da jede Kontrollmoeglichkeit fehlt (wie z. B. Markierungsbytes).

Wenn Sie es bis zu diesem Punkt ausgehalten haben, haben Sie einen groben Ueberblick bekommen, was man mit den 5 verschiedenen Filetypen im NEWDOS/80 alles machen kann. Als Unterstuetzung des TRSDOS wurden die Field Item Files und die PRINT/INPUT-Files nicht geaendert im NEWDOS/80, aus diesem Grunde gibt es hier auch keine Beschreibung. Nehmen Sie jetzt am besten das NEWDOS/80- Handbuch, das Sie zusammen mit den Betriebssystemdisketten erhalten haben (in der deutschen Uebersetzung: Teil 1) und lesen Sie da nach, wie dieser Filetyp erweitert wurde. Mit den Erkenntnissen aus diesem Teil des Handbuches und den Testfiles muesste es leicht sein fuer Sie, selbst einige Testprogramme fuer die Erweiterungen zu schreiben und selbst auszutesten. Viel Erfolg !

B.0 BASIC - BEISPIELE

Der Sinn und Zweck dieses Anhangs ist es, Ihnen einige Beispiele in die Hand zu geben ueber die Ein- und Ausgabe bei Marked Item und Fixed Item Files, sowie um Ihnen eine etwas unterschiedliche Beschreibung zu geben gegenueber dem Anhang A. In Kapitel 8 steht bereits die Spezifikation fuer die Ein- und Ausgabe. Dieser Anhang soll nur etwas die Zusammenhaenge veranschaulichen, ist aber keine Spezifikation!

In diesem Kapitel beziehen wir uns, wie im gesamten Handbuch, nur auf die Kurzbezeichnungen der einzelnen Filetypen. Die genauere Erlaeuterung steht in Kapitel 10 fuer die Definition der MU-, MI-, MF-, FF- und FI-Files. In diesem Kapitel werden ebenfalls andere Abkuerzungen benutzt wie z. B. IGEL, RBA, REMRA, REMBA etc., die auch in Kapitel 8 erlaeutert sind.

Der Author hat versucht, die dargestellten Beispielsprogramme so einfach wie moeglich zu gestalten, damit der Unterschied leichter zu erkennen ist.

Da man nur daran interessiert war, die Anwendung der verschiedenen Filetypen darzustellen, und nicht die eigentliche Verarbeitung (ausgenommen mit CMD"O"), ist anstelle der Verarbeitung nur ein GOSUB eingefuegt. Wenn der Anwender diese Beispielsprogramme selbst in der Praxis einsetzen moechte, muss er dazu selbstverstaendlich die Anwendung schreiben.

In allen Beispielen wurden in der Ein- und Ausgabe nur Variable mit angefuegter Typkennung ($, %, !, #) benutzt - siehe dazu Zeile 120 des Beispiels 3. Dies wurde getan, damit der Leser sofort weiss, um welche Variablentypen es sich handelt. Der Verfasser empfiehlt auch mit allem Nachdruck, dass Sie in Ihren eigenen Programmen das Gleiche machen. Es kann sehr leicht geschehen, dass man sich im Programm nicht mehr an den Variablentyp erinnern kann und einen falschen Typ benutzt, was leicht zu einer Zerstoerung des ganzen Files fuehren kann. In der Version 1 des NEWDOS/80 war dies sogar von der Software noch fuer Fixed Item Files zwingend vorgeschrieben, in der Version 2 ist dies nicht mehr. Um ein Beispiel fuer einen IGEL zu erhalten, der Variable ohne explizite Typvereinbarung enthaelt, koennen Sie die Zeilen 10 und 120 in Beispiel 7 aendern in:

```
 10 CLEAR 2000:DEFSTR N,S:DEFINT A,I:DEFSNG F:DEFDBL D
120 (20)NM,AN,AM!,DT,(15)ST,FP,DP;
```

Beachten Sie aber, dass die Angabe des Typkennzeichens strikt verlangt wird.

GET und PUT arbeiten in 2 Phasen:

1. Die Positionierungsphase im File. In dieser Phase wird der File entsprechend dem 2. Parameter des GET oder PUT, dem Positionierungsparameter, positioniert. Am Ende dieser Phase werden fuer die verschiedenen weiteren Positionierungen die Werte REMRA und REMBA gespeichert fuer den Fall, dass der naechste Aufruf mit # oder # erfolgt.

2. Die Uebertragungsphase der Daten. In dieser Phase werden die Daten zwischen dem File und den Variablen, die im IGEL angegeben sind, uebertragen.

1. Beispiel - Sequentielles Schreiben in einen MU-File.

Die MU-Files waren gedacht als Alternative fuer die PRINT/INPUT-Files. Ein MU-File benutzt weniger Platz auf der Floppy als ein PRINT/INPUT-File, er kann mit gewissen Einschraenkungen veraendert werden, kann ueber den Parameter !RBA positioniert werden und benoetigt keine Sequenz mit ;",": um Strings zu separieren beim Schreiben.

```
10  CLEAR 2000
20  OPEN"O",1,"XXX/DAT:1","MU"
30  GOSUB 10000              DATEN VORBEREITEN
40  IF RN% = 0 THEN CLOSE : END
50  PUT 1,,,NM$,AN%,AM!,DT$,ST$,IG%,FP!,DP$;
60  GOTO 30
```

Der File wird eroeffnet fuer sequentielle Ausgabe von Records, deren Laenge von der Groesse der 2 Stringvariablen in jedem Record abhaengen.

Der Positionierungsparameter ist 0, was bedeutet, dass bei jeder Ausfuehrung des PUT-Statementes der naechste Satz geschrieben wird.

Die Daten werden in der Routine 10000 erzeugt, die der Anwender selbst einzufuegen hat. Wenn keine neuen Records mehr erzeugt werden, setzt man RN% auf 0. Andernfalls setzen Sie RN% auf ungleich 0 und belegen Sie die Filevariablen NM$, AN%, DT$, ST$, IG%, FP! und DP$ mit dem Wert, der in den File geschrieben werden soll.

Der IGEL in diesem Beispiel ist im PUT-Statement enthalten und besteht aus 8 Ausdruecken (in diesem Falle alles Variable). Die Variablen sind voneinander durch Kommas getrennt, der IGEL schliesst mit einem Semikolon ab.

Der volle Inhalt der Stringvariablen NM$ und ST$ wird in den File geschrieben, jeweils mit einem oder zwei Markierungsbytes. Das zweite Markierungsbyte ist fuer Strings mit einer Laenge von 128 bis 255 Character.

Jede der 2 Integerzahlen AN% und IG% wird durch 3 Bytes im File dargestellt, einem Markierungsbyte mit dem Wert 72H folgen 2 Bytes Daten im Format, wie es BASIC darstellt.

Jede der 2 Gleitkommazahlen AM! und FP! wird durch 5 Bytes im File dargestellt, einem Markierungsbyte mit dem Wert 73H folgen 4 Bytes Daten im Format, wie es BASIC darstellt.

Jede der 2 Gleitkommazahlen mit doppelter Genauigkeit DT$ und DP$ wird durch 9 Bytes im File dargestellt, einem Markierungsbyte mit dem Wert 73H folgen 8 Bytes Daten im Format, wie es BASIC darstellt.

Ueber den IGEL im PUT-Statement werden die minimale und die Maximale Recordlaenge ausgerechnet. Die minimale Laenge betraegt 37 Bytes (beide Strings haben eine Laenge von 0 und einschl. das SOR-Byte), die maximale Laenge betraegt 549 Bytes (beide Strings haben eine Laenge von 255 Bytes).

2. Beispiel - Sequentielles Lesen von einem MU-File.

```
10  CLEAR 2000
20  OPEN"I",1,"XXX/DAT:1","MU"
30  IF EOF(1) THEN END
40  GET 1,,,NM$,AN%,AM!,DT$,ST$,IG%,FP!,DP$;
50  GOSUB 10000              ' DATEN VERARBEITEN
60  GOTO 30
```

Dieses Beispiel ist umgekehrt wie Beispiel 1, es werden aber die Variablen und der gleiche IGEL benutzt. Die einzelnen Datenrecords werden nacheinander gelesen und verarbeitet. Der Programmierer muss selbst seine Verarbeitungsroutine 10000 schreiben.

Der Positionierungsparameter im GET-Statement ist null. Das bedeutet, dass bei jedem GET der darauffolgende Satz gelesen wird.

Die EOF(1) Funtion kehrt mit dem Wert TRUE zurueck, wenn der File genau auf EOF steht und laeuft dann auf Ende.

3. Beispiel - Sequentielles Lesen und Veraendern eines MU-Files.

```
 10  CLEAR 2000
 20  OPEN"R",1,"XXX/DAT:1","MU"
 30  IF EOF(1) THEN END
 40  GET 1,,120
 50  GOSUB 10000       ' DATEN VERARB. UND VERAENDERN
 60  PUT 1,#,120
 70  GOTO 30
120  NM$,AN%,AM!,DT#,ST$,IG%,FP!,DP#;
```

In diesem Beispiel wird der gleiche File erzeugt wie in Beispiel 1. Der File wurde eroeffnet fuer Lesen und Schreiben. Die Records werden sequentiell in die BASIC-Variable gelesen, eine oder mehrere Variable werden veraendert in der Benutzerroutine 10000 und dann zurueckgeschrieben.

Der Positionierungsparameter im PUT-Statement ist #. Damit wird der File wieder auf die gleiche Startadresse des gleichen Records wie beim GET-Statement positioniert, also der gleiche Record zurueckgeschrieben.

Sowohl das GET als auch das PUT benutzen den gleichen IGEL in der Zeile 120. Dieser IGEL ist identisch mit den IGEL's in Beispiel 1 und 2, nur dass der IGEL nicht Bestandteil ist des GET bzw. PUT-Statements, sondern seperat in einer Zeile steht und das GET und PUT die Statementnummer enthaelt.

Ein Fehler wird gemeldet, wenn im PUT-Statement eine andere Recordlaenge gefunden wird, die laenger ist als die urspruengliche Recordlaenge in Beispiel 1. Dies wird aber nur dann geschehen, wenn die String-Variablen im File laenger werden als vorher und auch der zusaetzlich geschriebene Platz (mit Angabe der Funktion (len)#) verbraucht wird. Die numerischen Variablen belegen in jedem Falle den gleichen Platz wie schon zuvor. Man sollte also in jedem Falle vorsichtig sein, wenn man die String-Variable in einem MU-File veraendert, damit sie nicht laenger wird als vorher.

4. Beispiel - Einlesen, sortieren und zurueckschreiben eines MU-Files.

```
10 CLEAR 10000 : DEFINT I
15 DIM NM$(200),AN%(200),AM!(200),DT#(200)
17 DIM ST$(200),IG%(200),FP!(200),DP#(200),IX%(200)
20 IX=0 : OPEN"I",1,"XXX/DAT:1","MU"
30 IF EOF(1) THEN 80
```

```
 40 IX=IX+1
 45 IF IX>200 THEN PRINT"TOO MANY RECORDS" : END
 50 GET 1,,60 : GOTO 30
 60 NM$(IX),AN%(IX),AM!(IX),DT$(IX),    ' 1. ZEILE IGEL
 70 ST$(IX),IG%(IX),FP!(IX),DP$(IX);    ' L. ZEILE IGEL
 80 IF IX=0 THEN PRINT"EMPTY FILE" : END
 90 CMD"O",IX,*IX%(1),AM!(1),NM$(1)
100 CLOSE : OPEN"O",1,"XXX/DAT:1","MU"
110 IY = IX : FOR IZ = 1 TO IY
120 IX = IX%(IZ) : PUT 1,,60
130 NEXT IZ : CLOSE : END
```

Der MU-File XXX/DAT:1, der zwischen 1 und 200 Records gross ist, wird in insgesamt 8 Felder gelesen.

Diese Records werden dann indirekt sortiert in Zeile 90 mit dem BASIC-Sort und der Variable IX% als indirektes Sort-Feld. Das Sortierkriterium ist AM! in aufsteigender Reihenfolge, wenn mehrere Elemente gleich sind, werden die Werte von NM$ benutzt. Waehrend des Sortiervorganges werden die Elemente von IX% entsprechend der Sortierfolge eingeordnet.

Es sollte ferner erwaehnt werden, dass ein Record zwar aus 8 Elementen besteht, dass aber fuer den Sortiervorgang nur 2 Elemente herangezogen werden.

Nach dem Sortiervorgang werden die sortierten Felder wieder zurueckgeschrieben. Da hierfuer der gleiche File wieder benutzt wird, ist es empfehlenswert, sich vorher eine Kopie zu machen, falls es beim Schreiben des sortierten Files zu einem Fehler kommt.

In diesem Beispiel sehen Sie, dass ein IGEL auch ueber mehrere Zeilen gehen darf (Zeile 60 und 70).

5.. Beispiel - Sequentielle Records schreiben in FF-Files.

Die FF-Files waren gedacht als Ersatz fuer die Field Item Files (TRSDOS Random Files). Das FIELD-Statement wird nicht benutzt bei FF-Files, dennoch kann der Benutzer eine Pseudo-Field Operation durchfuehren. LSET und RSET werden nicht mehr benutzt bei FF-Files fuer das Setzen von Variablen, dennoch kann ein Anwender wenn er wuenscht die Variablen nach der Pseudo-Field Operation die Stringvariablen damit setzen. Fuer numerische Variable darf LSET und RSET nicht benutzt werden. Ebenso werden die Funktionen MKD$, MKI$, MKS$, CVD, CVI und CVS nicht mehr benutzt.

```
10  CLEAR 2000
20  OPEN"O",1,"XXX/DAT:1","FF",63
30  GOSUB 10000          ' DATEN VORBEREITEN
40  IF RN% = 0 THEN CLOSE : END
50  PUT 1,,,(20)NM$,AN%,AM!,DT$,(15)ST$,IG%,FP!,DP$;
60  GOTO 30
```

Der File ist damit eroeffnet fuer sequentielle Ausgabe von Records mit je 63 Bytes Laenge.

Der File-Positionierungsparameter im PUT-Statement ist null, was bedeutet, dass mit jedem PUT jeweils der naechste Record geschrieben wird.

Die Routine 10000 muss vom Anwender selbst geschrieben werden um die Daten aufzubereiten. Sollen keine Records-

mehr in den File geschrieben werden, setzt man RN% auf 0. Anderfalls setzt man die Variable RN% auf # 0, und laedt die 8 zu schreibenden Variablen NM$, AN%, DT$, ST$, IG%, FP! und DP# mit den Daten, die in den File geschrieben werden sollen.

Der IGEL ist im PUT-Statement und besteht aus den 8 Variablen. Jede Variable entspricht einem Bestandteil eines Records, alle Variablennamen werden voneinander mit Komma getrennt und mit einem Semikolon abgeschlossen.

Jeder der Strings NM$ und ST$ wird im File durch die Anzahl der Zeichen, die als Vorsatz vor dem Variablennamen stehen, dargestellt. Bei jedem PUT wird der Inhalt der Variablen, z. B. NM$, in den File geschrieben. Sollte einmal NM$ laenger sein als 20 Zeichen, so werden alle Zeichen, die nach dem 20. liegen, abgeschnitten. Die Variable selbst bleibt aber unveraendert, die Zeichen werden nur nicht in den File geschrieben. Wenn NM$ kuerzer ist als 20 Variable, so wird der Record, nicht aber die Variable selbst mit Leerzeichen aufgefuellt. Gleiches gilt auch fuer die 15 Zeichen lange Variable ST$.

Jede der Integer-Zahlen AN% und IG% wird mit 2 Bytes Laenge in den File geschrieben. Sie haben das gleiche Format wie im BASIC.

Jede der Gleitkommazahlen AM! und DF! wird mit 4 Bytes Laenge in den File geschrieben. Sie haben das gleiche Format wie im BASIC.

Jede der Gleitkommazahlen mir doppelter Genauigkeit wird mit 8 Bytes Laenge den File geschrieben. Sie haben das gleiche Format wie im BASIC.

6. Beispiel - Sequentielle Records lesen von FF-Files.

```
 10  CLEAR 2000
 20  OPEN"I",1,"XXX/DAT:1","FF",63
 30  IF EOF(1) THEN END
 40  GET 1,,,(20)NM$,AN%,AM!,DT$,(15)ST$,IG%,FP!,DP#;
 50  GOSUB 10000
 60  GOTO 30
```

Dieses Beispiel hat die umgekehrte Funktion des Beispieles 5. Es werden die gleichen Variablen und die der gleiche IGEL benutzt. Die Daten-Records werden nacheinander gelesen und verarbeitet ueber die Routine 10000, die vom Anwender geschrieben werden muss.

Nach jedem Lesen enthaelt AM$ einen 20 Character langen String, ST$ einen 15 Character langen String.

7. Beispiel - Sequentielle Records lesen und Update im FF-File.

```
 10  CLEAR 2000
 20  OPEN"R",1,"XXX/DAT:1","FF",63
 30  IF EOF(1) THEN END
 40  GET 1,,120
 50  GOSUB 10000
 60  PUT 1,#,120
 70  GOTO 30
120  (20)NM$,AN%,AM!,DT$,(15)ST$,IG%,FP!,DP#;
```

Es wird hier der gleiche File benutzt, der in Beispiel 5 erzeugt wurde. Er wurde eroeffnet fuer Lesen und Schreiben. Die Records werden nacheinander vom File in die BASIC-Variablen gelesen und eine oder mehrere davon veraendert ueber die Routine 10000. Danach wird der Record wieder zurueckgeschrieben.

Der Postionierungsparameter im PUT ist ein #. Das bedeutet, dass vor jedem PUT der File auf den Beginn des letzten Records zurueckgesetzt wird. Diese letzte Adresse vom GET ist die REMRA.

Sowohl GET als auch PUT benutzen den gleichen IGEL auf Zeile 120.

Nach jedem GET enthaelt NM$ einen String mit 20 Charactern Laenge, ST$ enthaelt einen String von 15 Charactern Laenge. In der Routine 10000 kann nun der Anwender die Laenge eines oder beider Strings veraendern. Waehrend des PUT wird jedoch AM$ wieder auf eine Laenge von 20 Charactern gesetzt, entweder wird rechts abgeschnitten oder mit Leerzeichen aufgefuellt. Gleiches geschieht mit ST$ auf 15 Zeichen Laenge.

8. Beispiel - Sequantielle Records lesen und Update im FF-File.

```
 10  CLEAR 2000
 20  OPEN"D",1,"XXX/DAT:1","FF",63
 30  GOSUB 10000          ' RECORD BESTIMMEN
 40  RN% = 0 THEN END     ' FERTIG
 50  GET 1,RN%,120
 60  GOSUB 15000          ' RECORD VERAENDERN
 70  IF RN% <> 0 THEN PUT 1,RN%,120
 80  GOTO 30
120  (20)NM$,AN%,AM!,DT$,(15)ST$,IG%,FP!,DP#;
```

Dieses Beispiel ist aehnlich wie Beispiel 7, ausser dass die Records wahlfrei gelesen werden und der Anwender selbst bestimmt, ob ein Record veraendert wird.

Es wird hier der gleiche File benutzt, der in Beispiel 5 erzeugt wurde. Es gilt folgender Ablauf fuer die Records:

- In Routine 10000 kann die Nummer des Records eingegeben werden, der als naechstes bearbeitet werden soll. Bei der Rueckkehr aus dieser Routine enthaelt RN% die Recordnummer, wenn RN%=0 ist, wird das Programm beendet.
- Der Record wird gelesen mit RN% als Positionierungsparameter.
- In der Routine 15000 kann der gelesene Record angezeigt werden. Es koennen ein oder mehrere Bestandteile dieses Records geaendert werden. Setzt man RN% auf 0, so wird der Record nicht zurueckgeschrieben, andernfalls wird RN% nicht veraendert.
- Wenn RN% # 0 ist, wird der Record zurueckgeschrieben. RN% wird dabei als Recordnummer benutzt. Es sei darauf hingewiesen, dass der Positionierungsparameter fuer das PUT auch '#' haette sein koennen.

Der erste Parameter im OPEN-Statement ist ein "D" anstelle eines "R". Man haette auch "R" benutzen koennen, der Parameter "D" bewirkt aber, dass der File, falls RN% aus irgend einem Grunde in einer Routine veraendert wird, nicht vergroessert werden kann, es kann also nicht hinter EOF geschrieben werden.

9. Beispiel - Sequentielles Schreiben von Records in einen MU-File und sequentielles Schreiben von Records in einen FF-File, der als Index-File fuer den MU-File dient.

Es gibt sicherlich sehr viele Faelle, in denen man Strings von variabler Laenge in einen File schreiben moechte, bei denen nicht mit Leerzeichen aufgefuellt und auch nichts abgeschnitten werden soll, wie dies bei Field- oder Fixed-Item Files getan wird. Wenn man dann wahlfrei auf Records zugreifen moechte, muss man sich eine Index-Tabelle anlegen, mit der man dann direkt auf die Records zugreifen kann und diese gegebenenfalls veraendert.

```
 10  CLEAR 2000
 20  DIM AN%(4000),RB!(4000)  ' 2 FELDER MIT INDEX
 30  OPEN "O",1,"XXX/DAT:1","MU"  ' HAUPTFILE DATEN
 40  RC% = 0
 50  GOSUB 10000  '  DATEN FUER NAECHSTEN RECORD AUFB.
 60  IF RC% = 0 THEN 105    '  FERTIG
 70  RC%=RC%+1 : IF RC%>4000 THEN PRINT
     "FILE ZU GROSS" : GOTO 105
 80  RB!(RC%) = LOC(1)!   '  RBA - WERT NAECHST. RECORD
 90  PUT 1,,,NM$,AN%(RC%),AM!,DT$,ST$,IG%,FP!,DP$;
100  GOTO 50
105  CLOSE
110  IF RC%=0 THEN PRINT "KEINE DATEN IM RECORD" : END
120  CMD"O",RC%,AN%(1),RB!(1)    '  INDEX SORTIEREN
130  OPEN "O",1,"XXX/NDX:1","FF",6  ' OPEN INDEX FILE
140  FOR X = 1 TO RC%
150  PUT 1,,,AN%(X),RB!(X);    ' INDEX SCHREIBEN
160  NEXT X : CLOSE : END
```

Dieses Beispiel haette man auch so umschreiben koennen, dass der Index unmittelbar nach den Daten auf den File geschrieben wird, dies haette aber einige Nachteile mit sich gebracht. Da in diesem Falle beide Files auf der gleichen Floppy liegen, haette der Arm des Laufwerkes sich staendig hin und her bewegt, die Ausfuehrungszeit waere laenger gewesen. Aus diesem Grunde wurden die Indizes in ein Feld geschrieben und erst nach Beendigung der Verarbeitung in den File uebertragen.

In diesem Beispiel steht in dem Feld AN% eine Kontonummer, die nur einmal auftritt.

Das Programm arbeitet wie folgt:

- Fuer jeden Daten-Record:
  - Der Programmierer schreibt sich selbst die Routine 10000, in der RN% auf 0 gesetzt wird, wenn keine neuen Daten mehr eingegeben werden. Andernfalls wird RN% auf ungleich 0 gesetzt, und die Daten des Records in die richtige Variable geschrieben, inclusive der Kontonummer in das richtige Feld (AN%(x)).
  - Der RBA-Wert des Daten-Records innerhalb des Haupt-Files wird ermittelt ueber die Zeile 80 mit LOC(1)! und in das RBA-Feld RB!(x) abgespeichert.
  - Der Record wird in den Daten-File geschrieben.

  Die zwei Felder AN% und RB% werden direkt sortiert. Da es sich um einen direkten Sort handelt, werden die Werte beider Felder in aufsteigender Reihenfolge der Kontonummer in die Felder zurueckgespeichert. Dadurch, dass die Kontonummer jeweils nur einmal auftritt und die RBA-Werte im zweiten Sortierkriterium aufsteigend sind, werden sie nie ueberprueft.

Der Index-File wird durch das Schreiben der Index-Records erzeugt. Die Index-Records werden sequentiell in aufsteigender Reihenfolge der Kontonummern geschrieben.

Die gleichen Ergebnisse haette man erhalten, wenn man die Zeilen 80 und 90 wie folgt geschrieben haette:

```
 90  PUT 1,,,NM$,AN%(RC%),AM!,DT$,ST$,IG%,FP!,DP$;
 80  RB!(RC%) = LOC(1)$    '  RBA - WERT LETZTER RECORD
```

10. Beispiel - Sequentielles Lesen und Veraendern von Records in einen indizierten MU-File.

```
  10  CLEAR 2000
  20  DIM AN%(4000),RB!(4000)   ' 2 FELDER MIT INDEX
  30  OPEN "I",1,"XXX/NDX:1","FF",6   ' OPEN INDEX FILE
  50  IF EOF(1) THEN 100
  60  RC%=RC%+1
  70  IF RC%>4000 THEN PRINT "FILE ZU GROSS" : END
  80  GET 1,,,AN%(RC%),RB!(RC%);    ' INDEX LESEN
  90  GOTO 50
 100  IF RC%=0 THEN PRINT "KEINE KONTEN" : END
 110  OPEN "D",1,"XXX/DAT:1","MU"   ' HAUPTFILE DATEN
 120  GOSUB 10000    '  NAECHSTEN RECORD AUSWAEHLEN
 130  IF RC% = 0 THEN END      '  FERTIG
 140  FOR X = 1 TO RC%
 150  IF RN% = AN%(X) THEN 170   '  INDEX SUCHEN
 160  NEXT X : PRINT "FALSCHE KONTONUMMER" : GOTO 120
 170  GET 1,!RB!(X),300          '  RECORD LESEN
 180  IF AN% <> RN% THEN PRINT "FALSCHE DATEN IM RECORD"
      : END
 185  GOSUB 15000     '  DATEN ANZEIGEN UND VERAENDERN
 190  IF RN% <> 0 THEN PUT 1,0,300   ' ZURUECKSCHREIBEN
 200  GOTO 120
 300  NM$,AN%(RC%),AM!,DT$,ST$,IG%,FP!,DP$;
```

Es wird angenommen, dass Sie ungefaehr eine Anwendung haben, die die diesem Programm entspricht, mit dem Sie sequentielle Daten ueber eine Index lesen koennen, anzeigen, und ggf. veraendern und zurueckschreiben koennen.

Es werden die zwei Files von Beispiel 9 hier benutzt. Zuerst wird der Index-File eroeffnet in der Inhalt in die Felder AN% und RB! gelesen.

Fuer jeden Record wird folgendes ausgefuehrt:

- Der Anwender schreibt seine Routine 10000, in der die Kontonummer eingegeben wird. Beim RETURN enthaelt RN% die Kontonummer oder 0. Bei 0 wird das Programm abgebrochen.
- Das Feld AN% mit den Kontonummern wird durchsucht nach der eingegebenen Nummer. Wird sie nicht gefunden, wird eine Fehlermeldung ausgegeben.
- Der entsprechende Datenrecord wird eingelesen vom Hauptfile, dazu wird der entsprechende RBA-Wert aus dem Feld RB! benutzt. RN% wird dann verglichen mit dem Wert von AN% aus dem File, und wenn ungleich wird eine Fehlermeldung ausgegeben und das Programm beendet.
- Vom Anwender wieder stammt die Routine 15000, in der die Daten angezeigt und, falls noetig, veraendert werden. Wird dieser Record nicht veraendert, so muss RN% auf 0 gesetzt werden. Der Record wird dann nicht zurueckgeschrieben. Bei der Rueckkehr aus dieser Rou-

tine wird der Record, wenn RN% ungleich 0 ist, zurueckgeschrieben. Das PUT wird mit '#' positioniert, das bedeutet, dass der Record an die gleiche Stelle geschrieben wird (der File wird mit '#' auf REMRA positioniert).

11. Beispiel - Sequentielles Schreiben verschiedener Arten von Records in einen MU-File.

In einer moeglichen Anwendung sollen verschiedene Arten von Records geschrieben werden. Normalerweise wuerde man verschiedene Arten von Records in verschiedene Files schreiben. Wenn es jedoch der Fall ist, dass die einzelnen Records verschiedenen Typs sich aufeinander beziehen, ist es ggf. sinnvoller, wenn diese Records auch in einen File geschrieben werden um die Anzahl der Zugriffe gering zu halten. In MU, MI und FI-Files ist es erlaubt, grundsaetzlich unterschiedliche Records auf einen File zu schreiben, waehrend es bei MF und FF-Files nur dann erlaubt ist, wenn die Records gleiche Laenge haben.

Als Anwendungsbeispiel werden im englischen Orginal die Buchungen fuer eine Versicherung benutzt. Diese Buchungssaetze koennen aus einem Hauptsatz bestehen, einem oder mehreren Records mit Buchungen, einem oder mehreren Records mit Zahlungen und einen oder oder mehrere Records mit Forderungen. Jeder Dieser 4 verschiedenen Records hat also ein anderes Format.

Im unten aufgefuehrten Beispiel werden 3 verschiedene Formate benutzt, einen Hauptrecord und 2 verschieden Unterrecords. Es besteht hier kein echter Zusammenhang zwischen den Records, es sind im Prinzip Beispiele, die aus der Luft gegriffen sind. Sie sollen auch nur die Moeglichkeit von verschiedenen Recordarten und ihre Verarbeitung im MU-File darstellen.

In dem untenstehenden Beispiel besteht jeder Satz von Records aus einem Hauptrecord und keinem oder mehreren der 2 verschiedenen Unterrecords.

```
  10  CLEAR 2000
  20  OPEN "O",1,"XXX/DAT:1","MU"
  30  GOSUB 10000   '  RECORD ERZEUGEN
  40  IF RN% = 0 THEN CLOSE : END ' FERTIG
  50  PUT 1,,,"1",NM$,AN%(RC%),AM!,DT$,ST$,IG%,FP!,DP$;
      '  TYP 1 SCHREIBEN
  60  GOSUB 11000     '  TYP 2 ODER 3 ERZEUGEN
  65  IF X$ <> "2" THEN 90
  70  PUT 1,,,"2",SA$,SB$,LN$,PD!;   '  TYP 2 SCHREIBEN
  80  GOTO 60
  90  IF X$ <> "3" THEN 30
 100  PUT 1,,,"3",SJ$,DF$,IP%,IA%,PD!;   '  TYP 3
      SCHREIBEN
 110  GOTO 60
```

Die Routine 10000 wird vom Programmierer erstellt, sie erzeugt die Records vom Typ 1 und setzt RN% auf ungleich 0, wenn noch Saetze zu schreiben sind, andernfalls wird RN% auf 0 gesetzt. Fuer jeden Satz von Records gibt es nur einen Record vom Typ 1, es koennen aber mehrere von den anderen Typen sein.

Die Routine 11000 wird vom Programmierer erstellt, sie erzeugt die Records vom Typ 1 und setzt RN% auf ungleich 0, wenn noch Saetze zu schreiben sind, andernfalls wird RN% auf 0 gesetzt. Fuer jeden Satz von Records gibt es nur einen Record vom Typ 1, es koennen aber mehrere von den anderen Typen sein.

Jedes PUT-Statement enthaelt seinen eigenen IGEL. Beachten Sie, dass der erste Eintrag in jedem Falle ein Ausdruck ist anstelle eines Variablennamens. Es haette auch eine Variable sein koennen, die dann den Recordtyp haette enthalten muessen. Es wurde aber absichtlich dieser Weg gewaehlt um Ihnen zu zeigen, dass ein IGEL auch Ausdruecke verarbeiten kann und nicht nur Variablenlisten.

12. Beispiel - Sequentielles Schreiben verschiedener Arten von Records in einen MU-File und update dieser Records.

Der in Beispiel 11 erzeugte File wird hier benutzt. In diesem Beispiel wird gezeigt, wie in Marked Item Files (gilt ebenso fuer Fixed Item Files) Records teilweise eingelesen und verarbeitet werden koennen. Es werden die ersten drei, der 5. und 7. Bestandteil des Records vom Typ 1 eingelesen, die Records von Typ 2 werden ueberlesen, nur die Records von Typ 3 werden komplett gelesen und gegebenenfalls veraendert.

```
 10  CLEAR 2000
 20  OPEN "D",1,"XXX/DAT:1","MU"
 30  IF EOF(1) THEN END
 40  GET 1,,,RT$; '  RECORDTYP EINLESEN
 50  IF RT$ <> "1" THEN PRINT "FALSCHER RECORDTYP"    -
     : END
 60  GET 1,#,,NM$,AN%,,DT$,,IG%;   ' EINLESEN
 70  IF EOF(1) THEN END
 80  GET 1,,,RT$; '  RECORDTYP EINLESEN
 90  IF RT$ = "2" THEN 70   ' TYP 2 UEBERLESEN
100  IF RT$ <> "3" THEN 50
110  GET 1,#,200    ' REST VON TYP 3 EINLESEN
120  GOSUB 11000    ' RECORD GGF. VERAENDERN
130  IF RN% <> 0 THEN PUT 1,$,200   '  ZURUECKSCHREIBEN
110  GOTO 70
200  SJ$,DF$,IP%,IA%,FG!;
```

Die GET-Statements in den Zeilen 40 und 80 lesen nur ein Bestandteil des Records, der File bleibt dabei positioniert auf dem 2. Bestandteil dieses Records. Sowohl REMRA als auch REMBA stehen auf dem ersten Bestandteil des Files.

Die GET-Statements in den Zeilen 60 und 110 beginnen dort einzulesen, wo der Record von den GET's in den Zeilen 40 und 80 positioniert wurde. Die GET's in den Zeilen 60 und 110 positionieren nicht erst auf den naechsten Record vor dem Lesen. Bei diesen GET's wird die Adresse REMBA vor dem Lesen auf den 2. Bestandteil des Records gesetzt. Dadurch ergibt sich, dass bei dem PUT auf Zeile 130 der File auf die REMBA-Adresse des 2. Bestandteils zurueckgesetzt wird. Der Wert REMRA wird bei den GET's in den Zeilen 60 und 110 nicht veraendert.

Das GET in der Zeile 60 liest das 2., das 3., das 5. und das 7. Bestandteil des Records vom Typ 1 und laesst den File auf dem 8. Bestandteil positioniert. Das 4. und das 6. Bestandteil wird ueberlesen, mit der Ausfuehrung des naechsten GET's in Zeile 80 uebergeht den Rest des Records (8 und 9).

Der Rest des Records vom Typ 2 wird uebergangen mit der Ausfuehrung des GET's in Zeile 80.

Die Routine 11000 wird vom Programmierer selbst geschrieben. Sie veraendert die Daten des Records vom Typ 3. In diesem Falle wird RN% auf ungleich 0 gesetzt, werden keine Daten veraendert, wird RN% auf 0 gesetzt.

Bei der Rueckkehr aus der Routine 11000 wird der Record vom Typ 3, wenn RN% ungleich 0 ist, zurueckgeschrieben. Beachten Sie, dass nur vom 2. Bestandteil des Records an geschrieben wird. Das erste Bestandteil dieses Record wird nicht veraendert, da im PUT auf Zeile 130 der File auf die Position gebracht wird, wo er beim letzten Einlesen in Zeile 110 begonnen hat.

Erinnern Sie sich daran, dass beim Veraendern von MF und MU Records alle Bestandteile vom ersten geschriebenen an bis zum Ende zurueckgeschrieben werden muessen. Es ist nicht unbedingt notwendig, dass alle Bestandteile mit einem PUT geschrieben werden, dies kann auch in mehreren Durchgaengen erfolgen. Bei jedem PUT wird allerdings der Rest des Records mit Nullen aufgefuellt.

Bei Record-segmentierten Files wird die Position des naechsten Records mit der EOF-Position verglichen. Im Falle des PUT's in Zeile 130 ist die zurueckgelassene Position gleichzeitig die Startposition des naechsten Records. Bei den Records von Typ 1 und 2 jedoch entweder auf dem 8. oder dem 2. Bestandteil des Records. In diesen Faellen wird die Startposition des naechsten Records errechnet und dieser Wert fuer die EOF-Abfrage benutzt.

13. Beispiel - Sequentielles Schreiben in einen MF-File (markierte Bestandteile in einem File mit fester Recordlaenge).

Eine moegliche Anwendunggibt es fuer den Fall, dass ein File Strings enthaelt und diese bis zu einer gewissen maximalen Laenge bei Bedarf zu veraendern sind. Da in einem MU-File Strings nicht oder nur begrenzt zu verlaengern sind beim Veraendern, muss der Anwender entweder auf Field-Item Files, auf FF-Files oder zuletzt auf MF-Files zurueckgreifen. Die Eigenschaften von Field-Item Files wurden bereits besprochen, so wird in diesem Fall entweder auf FF- oder MF-Files zurueckgegriffen.

Der Vorteil von MF-Files ist der, dass Strings nicht aufgefuellt werden mit Leerzeichen bis zur maximalen Recordlaenge. Jeder String wird mit der Anzahl der Zeichen geschrieben, die er enthaelt und wird nicht erweitert. Der Record wird danach mit Nullen bis zu seiner maximalen Laenge aufgefuellt. Beim Lesen eines solchen Records werden diese Fuellbytes uebergangen, der Anwender sieht also nichts davon. Diese Fuellbytes koennen den Programmierer also auch dann nicht stoeren, wenn er Strings miteinander vergleicht.

Der Nachteil der MF-Files gegenueber dem FF-File ist der, dass ein MF-File durch die Markierungsbytes mehr Platz benoetigt als ein FF-File. In diesem Beispiel ist ein Record um 13% groesser als der gleiche Record im FF-File von Beispiel 5. Beide Records enthalten die gleichen Daten, ausser dass der Record im MF-File nicht mit Leerzeichen aufgefuellt ist.

```
10  CLEAR 2000
20  OPEN "O",1,"XXX/DAT:1","MF",71
30  GOSUB 10000   '  DATEN ERZEUGEN
40  IF RN% = 0 THEN CLOSE : END ' FERTIG
45  IF LOF(1)<RN% THEN PRINT "FALSCHE RECORD-#":GOTO 30
50  PUT 1,,T(20)NM$,AN%,AM!,DT#,(15)ST$,IG%,FP!,DP#
60  GOTO 30
```

Ein File wird eroeffnet fuer die sequentielle Ausgabe von Records mit einer Laenge von 71 Bytes. Entsprechend dem IGEL in Zeile 50 werden diese 71 Bytes aufgeteilt in

Bestandteile von 21, 3, 5, 9, 16, 3, 5 und 9 Bytes Laenge (jedes Bestandteil hat ein Markierungsbyte).

Vom Programmierer stammt die Routine 10000, die RN% auf 0 setzt, wenn keine neuen Saetze mehr erzeugt werden. Andernfalls wird RN% auf ungleich 0 gesetzt und die Daten fuer den neuen Record in die Variablen NM$, AN%, AM!, DT#, ST$, IG%, FP! und DP# gespeichert.

Die Daten stehen so im File, wie es bereits im Beispiel 1 fuer den MU-File, ausser dass das SOR-Byte nicht benutzt wird und dass beide Strings in ihrer Laenge begrenzt sind, NM$ auf 20 Zeichen und ST$ auf 15 Zeichen. Ist eine der Variablen groesser als angegeben, so wird nur der linke Teil mit der angegebenen Laenge uebertragen, der restliche Teil wird abgeschnitten.

Genaugenommen ist es nicht unbedingt notwendig, dass im IGEL in Zeile 50 eine Laengenangabe fuer die maximale Stringlaenge gemacht wird. Man haette ebenso den IGEL von Beispiel 1, Zeile 50, benutzen koennen. Andererseits ist es aber so, dass ohne Laengenangabe ein Record unter Umstaenden nur bedingt veraendert werden kann.

Mit der LOF-Funktion auf Zeile 45 wird ueberprueft, ob der gewuenschte Record sich innerhalb des Files befindet.

14. Beispiel - Wahlfreies Lesen und Veraendern in einem MF-File.

```
 10 CLEAR 2000
 20 OPEN "D",1,"XXX/DAT:1","MF",71
 30 GOSUB 10000   '  SATZAUSWAHL
 40 IF RN% = 0 THEN CLOSE : END ' FERTIG
 50 GET 1,RN%,120  '  RECORD LESEN
 60 GOSUB 15000   '  GGF. VERAENDERN
 70 IF RN% <> 0 THEN PUT 1,RN%,120   '  ZURUECKSCHREIBEN
 80 GOTO 30
120 (20)NM$,AN%,AM!,DT#,(15)ST$,IG%,FP!,DP#;
```

Der File wird fuer wahlfreies Lesen und Schreiben eroeffnet. Der erste Parameter im OPEN-Statement wurde mit "D" angegeben, damit der File nicht erweitert werden kann.

Vom Programmierer stammt die Routine 10000, in der eingegeben werden kann, welcher Record als naechstes bearbeitet wird. In dieser Routine wird in RN% die Satznummer eingegeben, bei 0 wird der File geschlossen und das Programm beendet.

Danach wird der entsprechende Record gelesen. In der Stringvariablen NM$ steht dann ein String in der Laenge von 0 bis 20 Zeichen, in ST$ ein String mit 0 bis 15 Zeichen, abhaengig davon, was beim PUT in diesen Record geschrieben wurde.

Ebenfalls vom Anwender stammt die Routine 15000, in der gegebenenfalls die Daten der Variablen NM$, AN%, AM!, DT#, ST$, IG%, FP! und DP# geaendert werden koennen. erfolgt keine Aenderung, wird RN% auf 0 gesetzt.

Ist nach der Routine 15000 RN% <> 0, so wird der Record zurueckgeschrieben. Hat eine der Stringvariablen eine neue Laenge, so erhaelt das entsprechende Bestandteil des Record auch diese neue Laenge.

15. Beispiel - Sequentielles Schreiben in einen MI-File.

MI- und FI-Files bestehen aus einer langen Serie von einzelnen Bestandteilen. Auch wenn der Programmierer logisch zusammengehoerende Gruppen in Records eingeteilt hat, weiss das BASIC nichts ueber diese Einteilung, wenn beim OPEN nicht die Recordlaenge angegeben wird (wie bei Field-Item, MF- oder FF-Files), die Records werden auch nicht getrennt durch ein SOR-Zeichen (Start of Record) wie im MU-File, oder ein EOL-Zeichen (Return, 0DH) wie im PRINT/INPUT-File. Da dem BASIC nichts ueber die logische Einteilung im MI- und FI-File bekannt ist, kann es auch nicht automatisch auf den naechsten Record positioniert werden, wie es beispielsweise in der Zeile 80 des Beispiels 12 gemacht wird, wo der Rest der Records vom Typ 1 und 2 ueberlesen wurde.

Benutzen Sie den Code von Beispiel 11 ohne Aenderung, so koennen Sie damit durch ein anderes OPEN-Statement einen MI-File erzeugen, der aus 3 verschiedenen Recordtypen besteht. Aendern Sie das OPEN-Statement von Beispiel 11 ab in

```
20  OPEN "O",1,"XXX/DAT:1","MI"
```

und erzeugen Sie damit einen File, der jetzt in Beispiel 16 benutzt wird.

16. Beispiel - Sequentielles Lesen eines MI-Files.

Der File von Beispiel 15 wird hier verwendet. Beachten Sie, dass Sie zwar wissen, dass der File 3 verschiedene Recordtypen enthaelt, das BASIC jedoch nicht. Aus diesem Grunde muss man, um von einem Record zu dem naechsten zu gelangen, den Rest des Records komplett einlesen. Dazu koennen jedoch mehrere GET-Statements benutzt werden.

Ein MI-File kann nicht veraendert werden. Die Einschraenkung wurde gemacht, da bei diesem Filetyp unterschiedliche Stringlaengen nicht behandelt werden koennen.

```
  10  CLEAR 2000
  20  OPEN "I",1,"XXX/DAT:1","MI"
  30  IF EOF(1) THEN END   '  LEERER FILE
  40  GET 1,,,RT$;   '  RECORDTYP
  50  IF RT$ <> "1" THEN PRINT "FALSCHE DATEN" : END
  60  GET 1,,,NM$,AN%(RC%),AM!,DT$,ST$,IG%,FP!,DP$;
      '  TYP 1 LESEN
  70  IF EOF(1) THEN END
  80  GET 1,,,RT$;   '  RECORDTYP
  90  IF RT$ <> "2" THEN 150
 100  IF RT$ <> "3" THEN 50
 110  GET 1,*,,SJ$,DF$,IP%,IA%,PD!;   '  TYP 3 LESEN
 120  GOSUB 11000   '  ABARBEITEN
 140  GOTO 70
 150  GET 1,,,SA$,SB$,LN$,PD!; : GOTO 70
```

Beachten Sie, dass die Zeile 60 keinen Positionierungsparameter hat, dagegen hat die Zeile 60 im Beispiel 12 den Parameter *. In Beispiel 12 wurde der '*' dazu benutzt, dass der File an der gleichen Stelle positioniert blieb und sich nicht auf den naechsten Record stellte. Da bei MI- und FI-Files dem BASIC keine Records bekannt sind, hat die Positionierung mit '*' oder eine Null-Angabe die gleiche Funktion. Der File bleibt in beiden Faellen an der gleichen Stelle stehen. So koennte im oben angegebenen Beispiel in den Zeilen 40, 60, 80, 110 und 150 anstelle des Null-Parameters ebensogut ein Stern stehen. Da es aber einfacher ist keinen Stern und nur die Kommas einzugeben,

wurde diese Darstellungsweise gewaehlt. Denken Sie aber daran, dass bei MU-, MF- und FF-Files der 'M' eine andere Bedeutung hat als ein Null-Parameter.

17. Beispiel - Sequentielles Schreiben von Records in einen FI-File und sequentielles Schreiben von Index-Records am Ende des Files.

Der FI-File ist sehr flexibel. Er hat im Prinzip die gleichen Eigenschaften wie der FF-File, hat aber im Gegensatz zu diesem die Moeglichkeit der variablen Recordlaenge. Genau wie bei den MI-Files weiss auch in diesem Falle das BASIC nichts ueber die Einteilung in Records. Ein FI-File kann eine Mischung aus den unterschiedlichsten Recordarten haben. In diesem Beispiel und in Beispiel 18 werden insgesamt fuenf verschiedene Recordtypen benutzt: die 3 verschiedenen Records aus den Beispielen 11, 12, 15 und 16, die Index-Records aus dem Beispiel 9 und 10 mit dem FF-File und zusaetzlich noch 2 Records mit 2 Bytes Laenge.

Gehen wir in diesem Beispiel wieder davon aus, dass die Variable AN% im Record vom Typ 1 wieder eine Kontonummer ist, die nur einmal auftritt. Der File wird zuerst beschrieben von den Records vom Typ 1, 2 und 3. Der Wert RBA und die Kontonummer wird von jedem Satz von Typ 1 in einem Feld gerettet. Nachdem alle Datenrecords geschrieben sind kommt ein Endekennzeichen mit einem Byte Laenge, als naechstes kommen die 2 sortierten Felder mit den Werten RBA und der Kontonummer. Zuletzt wird die Anzahl der Index-Records geschrieben.

Dieses Beispiel ist aehnlich wie die Beispiele 9 und 11, die meisten dieser Kommentare gelten auch hier. Der Unterschied ist der, dass alles ueber einen FI-File abgehandelt wird, und nicht ueber einen MU- und einen FF-File.

```
 10  CLEAR 2000
 20  DIM AN%(4000),RB!(4000)  ' 2 FELDER MIT INDEX
 30  OPEN "O",1,"XXX/DAT:1","FI"  ' DATEN/INDEX-FILE
 40  RC% = 0
 50  RT$ = "1" : GOSUB 10000 ' DATEN TYP 1 AUFB.
 60  IF RN% = 0 THEN 170     '  FERTIG
 70  RC%=RC%+1 : IF RC%>4000 THEN PRINT
     "FILE ZU GROSS" : GOTO 170
 80  RB!(RC%) = LOC(1)!   '  RBA NAECHST. RECORD
 90  PUT 1,,,(1)RT$,(20)NM$,AN%(RC%),AM!,
     DT$,(15)ST$,IG%,FP!,DP$;   '  TYP 1 SCHREIBEN
100  GOSUB 15000   '  TYP 2 ODER 3 ERZEUGEN
110  IF RT$ = "2" THEN 150
120  IF RT$ <> "3" THEN 50
130  PUT 1,,,(1)RT$,(3)SA$,(32)SB$,(14)LN$,PD!; ' TYP 3
140  GOTO 100
150  PUT 1,,,(1)RT$,(3)SA$,(32)SB$,(14)LN$,PD!; ' TYP 2
160  GOTO 100
170  IF RC%=0 THEN PRINT "KEINE DATEN IM RECORD" : END
175  RT$ = "0" : PUT 1,,,(1)RT$; ' ENDEZEICHEN
180  CMD"O",RC%,AN%(1),RB!(1)   '  INDEX SORTIEREN
190  FOR X = 1 TO RC%
200  PUT 1,,,AN%(X),RB!(X);    ' INDEX SCHREIBEN
210  NEXT X
220  PUT 1,,,RC%;   ' ANZAHL INDEX-WERTE SCHREIBEN
230  CLOSE : END
```

Der Vorteil beim Schreiben von Index-Saetzen in den gleichen File wie die eigentlichen Main-Records liegt darin, dass man fuer ein Copy oder Backup nur einen File hat und die Index-Daten in jedem Falle zu den Main-Records passen. In den Beispielen 9 und 10 haette man auch alles in einen File (MU) schreiben koennen.

Da es sich bei dem File um einen Fixed-Item-File handelt, muessen alle String-Variablen im IGEL die Laengenangabe enthalten. Ist der String in der Variablen laenger oder kuerzer als angegeben, wird er entweder abgeschnitten oder mit Leerzeichen aufgefuellt.

Gegenueber dem Beispiel 9 ist das Programm bis auf die nachfolgenden Zeilen 80 und 90 gleich:

```
80  PUT 1,,,(20)NM$,AN%(RC%),AM!,DT$,(15)ST$,IG%,FP!,DP$;
90  RB!(RC%) = LOC(1)$   '  RBA-WERT DES RECORDS
```

In Zeile 175 muss noch ein Endekennzeichen geschrieben werden, es wird in Beispiel 18 benutzt.

18. Beispiel - Wahlfreies Lesen und Veraendern von Daten-Records in einem MI-File.

Hier wird der File benutzt, der in Beispiel 17 erzeugt wurde. Die letzten 2 Bytes des Files werden gelesen, um eine Angabe zu erhalten wie viele Inex-Records und damit Records vom Typ 1 vorhanden sind. Die Index-Records werden in die 2 Felder eingelesen. Die Ausgewaehlten Daten-Records werden dann vom File gelesen und gegebenenfalls die Variablen DF$ und IA% vom Typ 3 veraendert und zurueckgeschrieben.

Dieses Beispiel ist eine Mischung der Beispiele 10 und 12, ausser dass sowohl die Daten- als auch die Index-Records in einem File vom Typ FI stehen. Es werden nur 2 Bestandteile des Recordtyps 3 veraendert.

```
 10  CLEAR 2000
 20  DIM AN%(4000),RB!(4000)   ' 2 FELDER MIT INDEX
 30  OPEN "D",1,"XXX/DAT:1","FI"   ' DATEN/INDEX-FILE
 40  X!=LOC(1)%-2   '  RBA LETZTE 2 BYTES ERRECHNEN
 50  GET 1,!X!,,RC%;   '  ZAEHLER EINLESEN FUER INDEX
 60  GET 1,!$(X!-6*RC%)   '  AUF ERSTEN INDEX POS.
 70  FOR X = 1 TO RC%   '  INDEX-DATEN EINLESEN
 80  GET 1,,,AN%(X),RB!(X);
 90  NEXT X
100  GOSUB 10000   '  KONTONUMMER EINGEBEN
110  IF RN% = 0 THEN END   '  FERTIG
120  FOR X = 1 TO RC%   '  KONTONUMMER IM FELD SUCHEN
130  IF RN% = AN%(X) THEN 150
140  NEXT X : PRINT "KONTONUMMER FALSCH" : GOTO 100
150  GET 1,!RB!(X),,(1)RT$,(20)NM$,AN%,AM!,
     DT$,(15)ST$,IG%,(12)$;   '  RECORD TYP 1 LESEN
160  IF RT$ <> "1" THEN PRINT "FALSCHER INDEX" : END
170  IF RN% <> AN% THEN PRINT "FALSCHE DATEN" : END
180  GET 1,,,(1)RT$;   '  RECORDTYP EINLESEN
190  IF RT$ = "3" THEN 230
195  IF RT$="1" OR RT$="0" THEN 100   '  ENDE ABPRUEFEN
200  IF RT$ <> "2" THEN PRINT "FALSCHE DATEN" : END
210  GET 1,#,,(3)SA$,(32)SB$,(14)LN$,PD!;
     '   TYP 2 UEBERLESEN
220  GOTO 180
230  GET 1,,,(40)SJ$,DF$,(2)$,IA%,FG!;   ' TYP 3 LESEN
240  GOSUB 11000   '  TYP 1 ODER 3 BEARBEITEN
250  IF RN% <> 0 THEN PUT 1,$,,(40)$,DF$,(2)$,IA%,(4)$;
260  GOTO 180
```

In Zeile 40 wird ueber die Funktion LOC(1)% der RBA-Wert von EOF ermittelt. Ueber diesen Wert kann man dann den RBA-Wert der letzten 2 Bytes errechnen, in denen die Anzahl der Index-Records steht und damit gleichzeitig die Anzahl

der Records vom Typ 1. Dieser Integerwert wird in RC% gelesen in Zeile 50.

In Zeile 60 wird die Anzahl der Index-Records benutzt, um vom Ende her den RBA-Wert des ersten Index-Wertes zu errechnen. Der File wird dann dort positioniert. Beachten Sie, dass der File mit einem GET-Statement und dem entsprechenden Positionierungsparameter positioniert wird. Der Parameter ist !$rba, was bedeutet, dass es sich um eine reine Positionierung handelt. Es darf in diesem Falle kein IGEL oder IGELSN folgen. Es findet auch keine Datenuebertragung statt.

Wie ueblich stammt die Routine 10000 vom Anwender. In dieser Routine wird die Kontonummer eingegeben, deren Hauptsatz und Folgesaetze eingelesen werden sollen. Um das Programm abzubrechen kann man RN% auf 0 setzen, andernfalls enthaelt RN% die Kontonummer.

Auf Zeile 150 wird der File auf den Wert positioniert, der mit der Kontonummer im Index-Feld gefunden wurde. Der Record vom Typ 1 wird dann gelesen. Da es unwesentlich ist, was in den Variablen ST$, FP! und DP# steht, koennen diese ueberlesen werden. Da es sich aber um einen Fixed Item File handelt, muss dem BASIC mitgeteilt werden, wie viele Bytes zu ueberlesen sind. Dazu wird die Funktion im Format (len)$ benutzt. Die Angabe (15)$ bewirkt, dass die 15 Bytes ueberlesen werden, die normalerweise zu der Variablen ST$ gehoeren. Mit (12)$ werden 4 Bytes ueberlesen, die zu der Variablen FP! gehoeren und die 8 Bytes der Variablen DP#. Zuletzt sei erwaehnt, dass der Ausdruck (12)$ in einem IGEL fuer einen FF-File haette weggelassen werden koennen, da sonst keine anderen Variablen folgen in diesem Record. Mit dem naechsten GET ohne Positionierungsparameter waere der FF-File automatisch auf den naechsten Record positioniert worden. Im FI-File ist das aber nicht moeglich, da das BASIC intern nichts ueber die Filestruktur weiss, aus diesem Grunde muss die Angabe (12)$ erfolgen.

In der Zeile 210 muss der File auch wieder auf den naechsten Record positioniert werden, da der Rest des Records von Typ 2 ueberlesen werden soll. Man haette die auch ueber folgendes Kommando errechen koennen:

```
120  GET 1,,,(53)$;  '  REST VON TYP 2 UEBERLESEN
```

Auch wiederum vom Programmierer stammt die Routine 11000, in der Daten der Records vom Typ 1 und 3 bearbeitet werden. Wird im Record vom Typ 3 nichts veraendert, kann man RN% auf 0 setzen. Andernfalls wird entweder DF$ oder IA% veraendert, und RN% auf ungleich 0 belassen oder gesetzt.

Das PUT auf Zeile 250 positioniert den File auf den Wert, der als REMBA-Position vom GET in Zeile 230 abgespeichert wurde. Beachten Sie, dass nur 2 Bestandteile des Records im File ersetzt werden (DF$ und IA%). Vergleichen Sie diese Zeile mit der Zeile 130 des Beispiels 12. Die anderen Bestandteile dieses Records werden uebersprungen und nicht veraendert. Der Programmierer muss bei der Angabe von (len)$ und (len)$ auf die richtige Anzahl der Bytes achten.

Die Benutzung des Ausdrucks (len)$ im IGEL der Zeilen 150 und 250 wurde nur deshalb gewaehlt, um dem Anwender ein Beispiel fuer (len)$ zu geben. Ebenso haette man einen eleganteren IGEL angeben koennen:

```
150   GET 1,#,,(1)RT$,(20)NM$,AN%,AM!,DT$,(15)ST$,
          IG%,FP!,DP#;
```

```
250 IF RN%<>0 THEN PUT 1,#,,(40)SJ$,DF$,IP%,IA%,FG!;
```

PAGE IS
MISSING

C.0 ZUSAMMENFASSUNG DER BEFEHLE

C.1 DOS-BEFEHLE

Kommando	Parameter	Bedeutung
APPEND	FNAME1,<TO,>FNAME2	File anhaengen
ATTRIB	FNAME1 <,INV><,VIS> <,PROT=xxx> <,ACC=passw1> <,UPD=passw2> <,ASE=yn><,ASC=yn> <,UDF=yn>	Fileattribute aendern
AUTO	<DOS-Kommando>	Startkommando
BASIC2	-	Eintritt in Lev. 2
BLINK	<,YN>	Cursor
BOOT	-	Reset
BREAK	<,yn>	Break-Taste
CHAIN	FNAME<,LABEL>	Chain-File ausf.
CHNON	<,NYD>	Chain-Modus
CLEAR	<,START=adr1><,END=adr2> <,MEM=adr3>	Clear Memory
CLOCK	<,yn>	Uhrzeit
CLS	-	Clear Screen
COPY	FNAME1<,TO>,FNAME2 <,SPDN=DI3><,DPDN=DI4>	kopieren Format 1
COPY	@FNAME1<,TO>,FNAME2 <,SPDN=DI3><,DPDN=DI4>	Format 2
COPY	<:>DI1,FNAME1<TO,>,FNAME2 <,SPDN=DI3><,DPDN=DI4>	Format 3
COPY	<:>DI1,@FNAME1<TO,>,FNAME2 <,SPDN=DI3><,DPDN=DI4>	Format 4

Kommando	Parameter	Bedeutung
COPY	<:>DI1<=tc1><,TO>, <:>DI2<=tc2> MM/TT/YY<,Y><,N> <,NDMW><,FMT><,NFMT> <SPDN=DI3><,DPDN=DI4> <,SPW=passwort1> <,NDPW=passwort3> <,DDND><,ODN=name1> <,ODN=name1><,KDN><,KDD> <,NDN=name2><,SN=name3> <,USD><,BDU><,UBB>	Format 5
COPY	<:>DI1<=tc1><,TO>, <:>DI2<=tc2> MM/TT/JJ,CBF<,Y><,N> <,USR><,/EXT><,UPD> <,ILF=FNAME3><,XLF=FNAME4> <,CFWO><,NDMW><,FMT> <SPDN=DI3><,DPDN=DI4><,NFMT> <,SPW=passwort1> <,ODPW=passwort2> <,NDPW=passwort3><,DDND> <,ODN=name1><,KDN><,KDD> <,NDN=name2> <,SN=name3><,USD><,UBB> <,DDSL=ln1><,DDGA=gc1>	Format 6
CREATE	FNAME<,LRL=ln1> <,REC=anz> <,ASE=yn><,ASC=yn>	File erzeugen
DATE	mm/dd/yy	Datum
DEBUG	<,yn>	DEBUG aktivieren
DIR	<:><DI><,S> <,A><,I><,U><,/EXT><,P>	Directory anzeigen
DO	FNAME<,LABEL>	wie CHAIN
DUMP	FNAME,SADR,EADR <,Entry><,Reloc>	Speicherinhalt auf File schreiben
ERROR	xx	Fehler anzeigen
FORMAT	DN2<=tc2>,NAME2,mm/dd/yy PASSWORT<,N><,Y><,NDMW> <,DDND><,ODN=name1><,KDN> <,DDSL=ln1><,DDGA=gc1> <,DPDN=dn4><,PFST=tn3> <,PFTC=tc3>	Formatieren

Kommando	Parameter	Bedeutung
FORMS	<,WIDTH=xxx><,LINES=yyy>	Druckerformat
FREE	<,P>	freier Platz
HIMEM	<,adr1>	hoechste Adresse
JKL	-	Bildschirminhalt dr.
KILL	FNAME	File loeschen
LC	<,yn>	Tastatur umschalten
LCDVR	<,yn(,yn)>	Bildschirm umsch.
LIB	-	
LIST	FNAME<,Startzeile(,Anz)>	File listen
LOAD	FNAME	Z-80 Programm laden
MDBORT		MINI-DOS abbrechen
MDCOPY	FNAME1<,TO>,FNAME2	kop. im MINI-DOS
MDRET		MINI-DOS beenden
PAUSE	Msg	Pause im Chaining Msg anzeigen
PDRIVE	Passwort:DN1,DN2 <,TI=TYP1><,TD=TYP2> <,DTC=tc1><,DDST=TN1> <,DDGA=GC1><,A>	Laufwerksparameter
PRINT	FNAME1<,Start(,Anz)>	File drucken
PROT	Passwort1:DN1 <,NAME=name1> <,DATE=mm/dd/yy> <,LOCK><,UNLOCK> <,PW=Passwort2> <,RUF>	Diskettenparameter

Kommando	Parameter	Bedeutung
PURGE	dn <,/typ><,USR>	Files loeschen
R	-	letztes Kommando wiederholen
RENAME	FNAME1<,to>,FNAME2	File umbenennen
ROUTE	CLEAR dev1<,dev2><,dev3>...	Ein- Ausgabe umleiten
SETCOM	OFF <,WORD=w><,BAUD=b> <,STOP=s><,PARITY=p> <,WAIT><,NOWAIT>	RS232 Parameter setzen (nur Modell III)
STMT	Msg	Msg ausgeben im DO
SYSTEM	Passwort:DN1 <,AA=aa><,AB=ab><,AC=ac> <,AD=ad><,AE=ae><,AF=af> <,AG=ag><,AH=ah><,AI=ai> <,AJ=aj><,AK=ak><,AL=al> <,AM=am><,AN=an><,AO=ao> <,AP=ap><,AQ=aq><,AR=ar> <,AS=as><,AT=at><,AU=au> <,AV=av><,AW=aw><,AX=ax> <,AY=ay><,AZ=az> <,BA=ba><,BB=bb><,BC=bc> <,BD=bd><,BE=be><,BF=bf> <,BG=bg><,BH=bh><,BI=bi> <,BJ=bj><,BK=bk><,BL=bl> <,BM=bm><,BN=bn>	Systemparameter
TIME	<hh:mm:ss>	Zeit
VERIFY	<,yn>	Disk kontr.
WRDIRP	dn1	Directory geschuetzt schreiben auf dn1

Kommando	Bedeutung
CMD"A"	nicht belegt
CMD"B"	nicht belegt
CMD"C"	Alle Leerzeichen und REM's entfernen
CMD"C",R	Alle REM's entfernen
CMD"C",S	Alle Leerzeichen entfernen
CMD"D",mm/dd/yy,xxx	Tag im Jahr in xxx aus Datum errechnen
CMD"D",-yy/ddd,mm/dd/yy	Datum aus Tag im Jahr errechnen
CMS"E"	letzten Fehler anz.
CMD"F=DELETE",v1,v2,...	Variable v1, v2, ... loeschen
CMD"F=KEEP",v1,v2,...	Variable v1, v2, ... nicht loeschen
CMD"F=SWAP",p1,p2	Variable p1 u. p2 vert.
CMD"F=POPF",p1,p2,....	Stack FOR...NEXT loeschen mit Laufvariablen p1, p2
CMD"F=POPS"	Stack GOSUB und Stack FOR ... NEXT loeschen
CMD"F=POPR"	Stack GOSUB loeschen