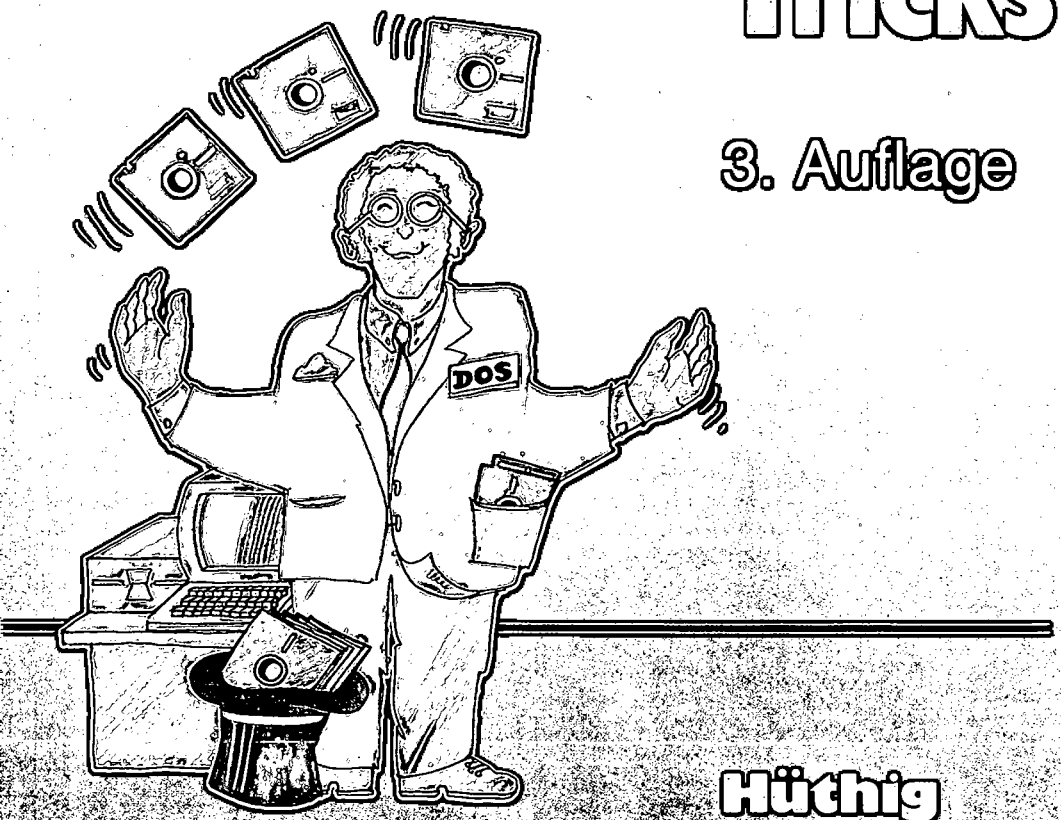
Apple DOS 3.3

Tips und Tricks

Mit ausführlichen Programmbeispielen

Ulrich Stiehl

3. Auflage



Hüthig

Ulrich Stiehl · Apple DOS 3.3 · Tips und Tricks

Ulrich Stiehl

Apple DOS 3.3

Tips und Tricks

Mit ausführlichen Programmbeispielen

3., völlig überarbeitete Auflage

Dr. Alfred Hüthig Verlag Heidelberg

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
„Begleitdiskette zu Apple DOS 3.3", DM 28,—
ISBN 3-7785-1298-6

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stiehl, Ulrich:
Apple DOS 3.3 : Tips u. Tricks ; mit ausführl. Programmbeispielen / Ulrich Stiehl. — Heidelberg : [Hauptbd.]. – 3., völlig überarb. Aufl. – 1986.
ISBN 3-7785-1297-8

Printed in Germany
Satz, Druck und Bindung: Druckerei Bitsch GmbH, Birkenau

Vorwort

Während sich die zweite Auflage dieses Buches auf die Korrektur einiger Satzfehler beschränkte, ist die dritte Auflage ab Seite 121 ff. völlig überarbeitet worden. Alle größeren Programme, deren Listings nunmehr im Photosatz produziert wurden, sind jetzt ohne jegliche Assemblerkenntnisse sofort einsetzbar. Da die neue Begleitdiskette zu diesem Buch Diversi-DOS 2C enthält, wurde auf den Abdruck der FASTBRUN-Routine zugunsten eines neuartigen RAM-Disk-Drivers verzichtet, der ein Move-Programm umfaßt, das den kompletten Inhalt der RAM-Disk auf der physischen Diskette sichert und von dort beim Programmstart automatisch eingeladen werden kann.
Die Verwendung von Diversi-DOS 2C anstelle des alten DOS 3.3 wird nachdrücklich empfohlen, da nicht nur bei Binärfiles (BLOAD/BSAVE), sondern auch bei Textfiles (READ/WRITE) die Geschwindigkeitssteigerungen enorm sind. Der Hüthig-Verlag oder genauer die Zeitschrift „Peeker" sind autorisiert, Diversi-DOS in Dateiform zu vertreiben. Benutzer von Diversi-DOS werden jedoch gebeten, 30.00 Dollar an Diversified Software Research, 34880 Bunker Hill, Farmington, MI 48018-2728 zu überweisen.

Die dritte Auflage von „Apple DOS 3.3" ist noch mehr zum Praktiker-Buch geworden, das auf unnötigen theoretischen Ballast verzichtet und statt dessen handfeste Anwendungen aufzeigt. Wer sich zusätzlich für die Theorie interessiert und über sehr gute Assemblerkenntnisse verfügt, sei auf die nicht ganz billige Neuerscheinung „B. Ruhland: DOS 3.3 – das Diskettenbetriebssystem des Apple II" verwiesen, die das alte DOS 3.3 bis zum letzten Byte detailliert beschreibt. Anwendungsbeispiele und Tips für die Praxis wird man dort allerdings nicht vorfinden.

Ulrich Stiehl

Heidelberg, im November 1985

Inhalt

Teil I: DOS für Anwender und Applesoft-Programmierer

In diesem Teil I werden die Grundlagen des DOS 3.3-Betriebssystems in leichtverständlicher Form für Benutzer von Anwenderprogrammen sowie Applesoft-Programmierer behandelt. Einige Fakten sind bewußt vereinfacht dargestellt, damit der Neuling nicht durch zu viele technische Details abgeschreckt wird. Allerdings setzt dieser Teil ab Kapitel 2 Grundkenntnisse von Applesoft voraus, da dieses Buch kein Applesoft-Lehrbuch darstellen kann.

1. DOS für Benutzer von Anwenderprogrammen

1.1. Betriebssystem

DOS ist die Abkürzung für Disk Operating System (= Diskettenbetriebssystem). Ein Diskettenbetriebssystem ist ein in der Regel in Maschinensprache geschriebenes Systemprogramm, das der Verwaltung von Files (= Dateien) auf der Diskette dient. Files können Programme, z.B. Basic- oder Maschinenprogramme, Speicherauszüge, z.B. Grafikbilder, sowie Dateien im engeren Sinne sein, z.B. Brieftexte, Adressen, Telefonverzeichnisse usw.
Für den Apple II Plus und den neueren Apple IIe gibt es verschiedene Diskettenbetriebssysteme, nämlich

a) das Pascal-Betriebssystem
b) das ProDOS-Betriebssystem
c) das DOS 3.3-Betriebssystem

Betriebssysteme sind zumeist mehr oder weniger eng mit Programmiersprachen verquickt. So eignet sich z.B. das Pascal-Betriebssystem grundsätzlich nur zur Anwendung in Verbindung mit der Programmiersprache Pascal. ProDOS und DOS 3.3 sind für die Programmiersprachen Basic und Assembler (6502 Maschinensprache) gedacht. DOS 3.3 kann sowohl unter Applesoft Basic als auch unter Integer Basic eingesetzt werden, während ProDOS auf Applesoft Basic beschränkt ist.

1.1.1. Wann soll ich ProDOS und wann DOS 3.3 einsetzen?

ProDOS ist für externe Massenspeicher, d.h. Diskettenlaufwerke mit großer Kapazität sowie für Fest- und Wechselplattenlaufwerke ausgelegt. Damit eignet sich Pro-

DOS besonders gut für die Verwaltung von großen Datenmassen, z.B. Adreßdateien mit 10.000 Adressen usw. Der Preis für diese größere, externe Speicherkapazität ist ein um ca. 12.000 Speicherstellen geringerer interner Speicherraum, weil ProDOS als umfangreicheres Betriebssystem entsprechend mehr Platz beansprucht.
DOS 3.3 eignet sich mehr für kleinere externe Datenspeicher, z.B. die klassischen Apple-Diskettenlaufwerke mit ca. 143.000 Zeichen externer Speicherkapazität sowie die neueren von Fremdfirmen angebotenen Laufwerke, die eine Speicherkapazität von mehr als 300.000 Zeichen haben.

1.1.2. Ist DOS gleich DOS?

Im Laufe der Zeit sind von Apple mehrere DOS-Verbesserungen erschienen, nämlich DOS 3, DOS 3.1, DOS 3.2, DOS 3.2.1 und schließlich DOS 3.3. In diesem Buch befassen wir uns ausschließlich mit DOS 3.3, da die vorangehenden Versionen, z.B. DOS 3.2 vom 16.2.1979, heute praktisch nicht mehr benutzt werden.

Aber auch DOS 3.3 ist nicht gleich DOS 3.3, denn im Laufe der Zeit sind von Apple-unabhängigen Firmen verschiedene DOS 3.3-Varianten veröffentlicht worden, z.B.

PRONTO-DOS
DAVID-DOS
HYPER-DOS
SUPER-DOS
DIVERSI-DOS
usw.

Hinweis:
Wenn in diesem Buch von Diversi-DOS die Rede ist, dann ist stets Version 2-C gemeint. Die neueren Versionen, insbesondere 4-C, sind nicht zu empfehlen, da bei diesen die Systemadressen nicht mehr stimmen.

Die beste DOS 3.3-Variante ist zweifellos Diversi-DOS, da dieses Betriebssystem bei einer Verbesserung der Zugriffsgeschwindigkeit um den Faktor 3-5 bei allen Datei-Arten einschließlich der sog. Textfiles keine Systemadreßänderungen vornimmt, so daß es mit DOS 3.3 völlig kompatibel ist, sofern der Programmierer im Falle eines Maschinenprogramms die Standard-Systemadressen benutzt und nicht wahllos im DOS „herumpokt". Diversi-DOS ist etwa so schnell wie ProDOS. Die anderen DOS 3.3-Varianten stellen in der Regel nur mäßige Verbesserungen dar, z.B. zumeist ausschließlich im Hinblick auf die Geschwindigkeit des Einladens von sog. Binärfiles (Basic-Programme, Maschinenprogramme, Grafik-Bilder).

Nachfolgend werden wir DOS 3.3 kurz als DOS bezeichnen. Alle Beschreibungen der Befehle und technischen Einzelheiten von DOS gelten auch für Diversi-DOS.

1.2. Laufwerk und Controller

Die klassischen Apple-Laufwerke sind Disk-Drives der Firma Shugart, die als weitgehend wartungsfrei und sehr zuverlässig bezeichnet werden können. Die Laufwerke sind durch ein Flachbandkabel mit einer sog. Controller-Karte verbunden, die in eine der sog. Slots („Schlitze") oder Steckplätze der Apple-Platine gesteckt werden können. Bei Apple II Plus können die Steckplätze 1-6 benutzt werden, während beim Apple IIe zusätzlich Slot 7 als möglicher Steckplatz in Frage kommt. Normalerweise sollte man die Controller-Karte in Slot 6 stecken, da zahlreiche Programme den Controller in Slot 6 erwarten. Ein einziger Controller kann mit zwei Laufwerken gleichzeitig verbunden werden, die dann entsprechend als Slot 6, Drive 1 bzw. Slot 6, Drive 2 angesprochen werden. Insgesamt können soviele Doppellaufwerke an den Apple angeschlossen werden, wie freie Steckplätze zur Verfügung stehen.

Die Apple-Laufwerke lesen eine Diskette stets von unten, d.h. der Lesekopf befindet sich sozusagen unterhalb der in das Laufwerk eingeschobenen Diskette. Man beachte, daß aus Gründen der Netzteilüberlastung zu einem bestimmten Zeitpunkt immer nur ein einziges Laufwerk eingeschaltet sein kann (erkenntlich an der rot aufleuchtenden „in use"-Lampe).

Die Pflege der Laufwerke beschränkt sich auf zweierlei:

1. kann man von Zeit zu Zeit— etwa zweimal pro Jahr— die Leseköpfe mit einer sog. Reinigungsdiskette reinigen. Hersteller von Reinigungsdisketten empfehlen eine häufigere, teilweise sogar wöchentliche Reinigung, doch scheint dies nach meinen Erfahrungen bei den Apple-Laufwerken nicht erforderlich zu sein.

2. kann man von Zeit zu Zeit die Laufwerksgeschwindigkeit neu justieren. Hierzu ist erforderlich, daß man das Gehäuse durch Entfernen der 4 Bodenschrauben abzieht und die hinten, unten rechts befindliche kleine Schraube („Potentiometer") nach rechts oder nach links dreht, je nachdem, ob die Rotationsgeschwindigkeit zu hoch oder zu niedrig ist. Die im Handel käuflichen Kopierprogramme enthalten meist eine entsprechende „Speed Adjustment"-Routine, die man zur Geschwindigkeitsregulierung laufen lassen muß.

1.3. Diskette

Disketten sind Datenträger mit direktem Zugriff, d.h. der Lesekopf kann per Programm jeden beliebigen Sektor der Diskette direkt lesen oder beschreiben. Demge-

genüber muß z.B. eine Datenkassette als Datenträger mit indirektem Zugriff solange vorwärts (oder sinngemäß rückwärts) gespult werden, bis der Lesekopf an der gewünschten Stelle angelangt ist.
Disketten können einseitig oder beidseitig mit einfacher oder doppelter Schreibdichte beschrieben werden. Für Apple-Laufwerke genügen einseitige Disketten mit einfacher Schreibdichte.
Ringverstärkte Disketten sind von Vorteil, doch darf der Ring bei Apple-Laufwerken nur ganz flach, d.h. wenig auftragend, sein. Eine optimale Zentrierung der Diskette erreicht man dadurch, daß man die Laufwerkklappe exakt in dem Moment behutsam schließt, in dem das rote „in use"-Lämpchen aufleuchtet.
Der Leseschlitz der Diskette, der sich — wie bereits erwähnt — unten befindet, sollte niemals mit (fettigen) Fingern in Berührung kommen. Ferner sollte man Disketten nicht biegen (und erst recht nicht knicken) und nicht auf den Monitor legen (wegen der Gefahr des Magnetfeldes). Schließlich sollte man Disketten keinen Temperaturschwankungen aussetzen. So kann z.B. das Verschicken von Disketten im Winter die Folge haben, daß der sich im Innern der Diskette befindliche Filz klamm wird und deshalb die magnetische Kunststoffscheibe (= die eigentliche Diskette) nicht mehr richtig rotiert.

1.3.1. Reset

Drücken Sie niemals die Reset-Taste, während ein Laufwerk eingeschaltet ist, denn wegen der damit verbundenen Stromzufuhr-Unterbrechnung wird in der Regel derjenige Sektor zerstört, auf dem der Lesekopf gerade ruhte, und zwar stets dann, wenn eine Schreiboperation stattfand. Bei Leseoperationen geschieht meistens nichts, doch wer weiß schon genau, ob gerade gelesen oder geschrieben wird? Deshalb Finger weg von der Reset-Taste, solange das „in-use"-Lämpchen an ist! Im äußersten Notfall Laufwerkklappe öffnen, Diskette halb oder ganz herausziehen und erst dann Reset drücken!

1.4. Speicherkapazität

Eine Diskette ist in Tracks (= konzentrische Spuren) und jede Spur ihrerseits in Sektoren eingeteilt. Die DOS-Diskette enthält 35 Spuren mit je 16 Sektoren. In einem einzelnen Sektor kann man 256 Zeichen (oder Bytes) speichern. Auf eine DOS-Diskette passen damit 35 mal 16 mal 256 = 143.360 Bytes (oder Zeichen). Eine Diskette umfaßt insgesamt 35 mal 16 = 560 Sektoren. Ein einzelne Spur enthält 16 mal

256 = 4096 Bytes oder 4 Kilobytes (1 Kilobyte = 1024 Zeichen), womit eine Diskette 35 mal 4 = 140 Kilobytes beinhaltet.

Die 35 Spuren sind von 0 bis 34 (oder hexadezimal von $00-$22) numeriert. Die Sektoren sind von 0-15 (oder hexadezimal von $00-$0F) numeriert. Die Spur 0 ist die äußerste und die Spur 34 die innerste Spur. Zwischen äußerster und innerster Spur beträgt der Abstand nicht einmal 1 cm, so daß die auf der Diskette genutzte Lesezone sehr schmal ist.

Auf den Spuren 0-2 befindet sich normalerweise das Betriebssystem, und auf der Spur 17 (hexadezimal $11) befindet sich der sog. Catalog, d.h. das Disketteninhaltsverzeichnis. Damit stehen auf einer normalen Diskette nur 35 minus 4 = 31 Spuren für reine Datenspeicherung zur Verfügung. Die Nettokapazität beträgt damit 31 mal 4096 = 126.976 Zeichen oder 31 mal 4 = 124 Kilobytes. Es besteht die Möglichkeit, DOS-lose Datendisketten zu erstellen, bei denen die Spuren 1 und 2 für Daten genutzt werden können, so daß dann die Nettokapazität 33 mal 4096 = 135.168 Zeichen beträgt.

1.5. Booten

„Bootstrap" bedeutet auf englisch eigentlich Stiefelschlaufe und „by one's own bootstrap" besagt soviel wie, daß man sich wie Münchhausen am eigenen Schopfe herauszieht. Ein Bootstrap-Programm oder kurz Boot-Programm ist ein sog. Urlader, der in aufeinanderfolgenden Stufen (ähnlich wie beim Raketenstart) das Betriebssystem einlädt. Das Boot-Programm befindet sich als nur 256 Bytes langes Programm in der Controller-Karte und wird im Falle von Slot 6 durch folgende Befehle aktiviert:

a) von Applesoft (Cursor Ü) aus durch PR#6 (bei Slot 5 PR#5 usw.)
b) vom Monitor (Cursor *) aus durch 6 Ctrl-P (bei Slot 5 5 Ctrl-P usw.)

Dies gilt für den Fall, daß der Apple bereits eingeschaltet war. Das reine Einschalten des Apple bewirkt automatisch den Boot-Vorgang (durch ein entsprechendes Monitorprogramm, das bei $FAA6 beginnt und dann nach $C600 im Falle von Slot 6 springt). Ferner bewirkt beim eingeschalteten Apple IIe die Tastenkombination Ctrl-Offener Apfel-Reset den Start des Boot-Programms.

Heutzutage gibt es kaum noch Apple II Plus Geräte mit weniger als 48K Speicherkapazität, und Apple IIe Geräte haben ohnehin stets 64K Speicherkapazität, so daß so-

genannte Master-Disketten, die das DOS in den jeweils höchstmöglichen Speicherbereich verschoben hatten, keine Bedeutung mehr haben.
Nach Beendigung des Boot-Vorgangs befindet sich das DOS in dem Bereich $9600-$BFFF (oder dezimal 38400-49151). Der für Programm und Daten freie Arbeitsspeicher beträgt damit ca. 36.000 Zeichen. Die Speicherverteilung sieht genauer gesagt folgendermaßen aus (technische Details siehe Teil II dieses Buches):

1. $0000-$00FF: Zero-Page oder Nullseite
2. $0100-$01FF: Stack oder Prozessor-Stapel
3. $0200-$02FF: Tastatureingabe-Puffer
4. $0300-$03CF: frei für kurze Maschinenprogramme
5. $03D0-$03FF: DOS-Vektoren
6. $0400-$07FF: Bildschirmspeicher
7. $0800-$95FF: frei für Programm und Daten
8. $9600-$9CFF: 3 DOS-Datenpuffer
9. $9D00-$AAC8: DOS-Befehlsinterpreter
10. $AAC9-$B5FF: DOS File Manager (Entry bei $AAFD)
11. $B600-$BFFF: DOS RWTS (Entry bei $BD00)

Wenn das DOS in die Language Card (= die oberen 16K RAM) des Apple IIe oder Apple II Plus durch einen sog. DOS-Mover geschoben wird, sieht die Speicherverteilung in der Regel so aus:

1. $0800-$BEFF: frei für Programm und Daten
2. $BF00-$BFFF: DOS-Driver für Bank-Selecting
3. $D000-$DFFF: Bank 1 der Language Card frei
4. $D000-$FFFF: Bank 2 der Language Card DOS

Die DOS RWTS würde dann z.B. bei $F600 beginnen. Der für Programm und Daten freie Speicherraum beträgt dann ca. 47.000 Zeichen.

1.6. Initialisierung (INIT)

Die Initialisierung— auch Formatierung genannt— bereitet die fabrikneue Leerdiskette zur Datenaufnahme vor, indem softwaremäßig ein bestimmtes Bit-Muster auf die Diskette geschrieben wird, damit der Lesekopf später die einzelnen Spuren und Sektoren finden kann (Apple-Disketten sind soft-sektoriert, d.h. softwaremäßig in Sektoren eingeteilt. Gegensatz: hard-sektoriert mit z.B. 16 Sektoren-Löchern).

Nach dem Booten einer bereits vorliegenden Programmdiskette, z.B. der „System Master"-Diskette, wird das sog. Hello-Programm geladen. Dies ist dasjenige (Applesoft-)Programm, unter dessen Name die Programmdiskette zuvor initialisiert wurde. Ein solches Hello-Programm kann z.B. so aussehen:

10 HOME : PRINT „DISKETTE ANGELEGT AM 1.3.84"

Anwender, die keinerlei Programmiererfahrung haben, verfahren bitte wie folgt:

1. „System Master"-Diskette in Laufwerk 1 einlegen
2. PR#6— gefolgt von der Return-Taste— eingeben
3. Leerdiskette in Laufwerk 1 einlegen
4. NEW— in Großbuchstaben, gefolgt von der Return-Taste— eingeben
5. 10 HOME : PRINT „DISKETTE ANGELEGT AM ..."
   — gefolgt von der Return-Taste— eingeben
6. INIT ZUNAME— gefolgt von der Return-Taste— eingeben

Damit wird die Leerdiskette unter Ihrem Namen initialisiert und zur Datenaufnahme vorbereitet.

1.7. CATALOG, LOCK, UNLOCK, VERIFY

Geben Sie jetzt den Befehl CATALOG— gefolgt von Return— ein. Sie sehen dann den Catalog oder das Disketteninhaltsverzeichnis (auch Directory genannt), das etwa im Falle des Namens Meier folgendermaßen aussieht:

DISK VOLUME 254

A 002 MEIER

„A" steht für Applesoft. „002" steht für die Anzahl der Sektoren (2), die das Applesoft-Programm namens „MEIER" einnimmt. Um die Länge einer Datei überschlägig zu berechnen, multiplizieren Sie die Anzahl der Sektoren minus 1 mit 256. Hier gilt: (2 minus 1) mal 256 = 256, d.h. die Datei MEIER ist bis zu 256 Zeichen lang.

Tippen Sie nun— stets gefolgt von der Return-Taste—

LOCK MEIER

und dann

CATALOG

Sie sehen jetzt

DISK VOLUME 254

*A 002 MEIER

Vor dem „A" steht also jetzt ein Sternchen, was besagt, daß die Datei „ge-LOCKt" oder schreibgeschützt wurde (to lock = zuschließen). Mit dem LOCK-Befehl kann man also eine Datei vor dem Überschreiben schützen.

Tippen Sie nun

UNLOCK MEIER

und dann

CATALOG

Sie sehen dann wieder den ursprünglichen Catalog, d.h. das Sternchen wurde wieder entfernt. Mit dem UNLOCK-Befehl (to unlock = aufschließen) läßt sich also der Schreibschutz wieder aufheben.

Der LOCK-Befehl ist nur ein relativer Schutz vor dem Überschreiben einer Datei, denn wenn Sie z.B. jetzt INIT MEIER erneut eingeben würden, so würde die Diskette durch den Formatierungsvorgang völlig überschrieben, d.h. man kann auch bereits initialisierte Disketten erneut initialisieren. Wenn sie ganz sicher gehen wollen, daß die Daten auf der Diskette nicht mehr verändert werden, dann überkleben Sie die seitliche Schreibschutzeinkerbung auf der Diskette mit dem in der Disketten-Box befindlichen Klebestreifen. Dann kann die Diskette weder überschrieben noch durch Unachtsamkeit aus Versehen neu initialisiert werden.

Tippen Sie nun

VERIFY MEIER

und danach

VERIFY MUELLER

Im ersteren Fall passiert gar nichts, während Sie im letzteren Fall die Fehlermeldung FILE NOT FOUND (= Datei nicht gefunden) erhalten. Der VERIFY-Befehl (to verify = überprüfen) überprüft erstens die Existens einer Datei und zweitens wird die Intaktheit der Datei durch vollständiges Einlesen derselben ermittelt. Es gibt also die Möglichkeit, daß zwar der Dateiname aus dem Catalog ersichtlich ist, aber der VERIFY-Befehl trotzdem zu einer Fehlermeldung führt (dann meist I/O ERROR), was besagt, daß die Datei physisch defekt ist, z.B. wenn Zigarettenasche auf die Diskette gefallen ist.

1.8 Kopieren von Disketten und Dateien

Die „System Master"-Diskette enthält zwei wichtige Kopierprogramme, nämlich

1. COPYA zum Duplizieren kompletter Disketten, und
2. FID zum Kopieren einzelner Dateien.

Beide Programme sind in dem „DOS Handbuch" von Apple ausführlich beschrieben, so daß hierauf nicht näher eingegangen zu werden braucht. COPYA wird mit
RUN COPYA
gestartet, während FID mit
BRUN FID
gestartet wird.

Anstelle von COPYA kann man auch das erheblich schnellere Programm COPY.IIE verwenden, dessen Listing im Teil II dieses Buches abgedruckt ist.

2. DOS für Applesoft-Programmierer

2.1. Dateinamen

Während maschinensprachlich die meisten ASCII-Zeichen als Dateinamen möglich wären, sollte man normalerweise bei der Wahl der Dateinamen folgende Regeln beachten:

1. Ein Dateiname sollte nur aus Großbuchstaben, Ziffern, Leertaste und Punkt bestehen.
2. Ein Dateiname sollte stets mit einem Großbuchstaben beginnen.
3. Ein Dateiname darf höchstens 30 Zeichen lang sein

Danach wäre z.B. „+1 Datei" ein illegaler Dateiname, weil er mit keinem Großbuchstaben beginnt und außerdem noch Kleinbuchstaben enthält. Beispielsweise kann der Applewriter IIe infolge eines Programmierfehlers keine Dateinamen einlesen, die den Bindestrich enthalten, z.B. „DATEI-1" (Grund: Der Applewriter versucht, den „-" in einen Großbuchstaben umzuwandeln, was natürlich nicht geht). Wenn man die obigen Regeln beachtet, entstehen niemals Konflikte bei Anwenderprogrammen.
Auf eine Diskette passen insgesamt 105 Dateinamen und damit theoretisch insgesamt 105 Dateien. Soviele Dateien können jedoch normalerweise niemals auf einer Diskette untergebracht werden, weil sonst jede Datei im Durchscnitt weniger als 5 Sektoren einnehmen müßte, denn eine Diskette enthält normalerweise nur 496 Daten-Sektoren (31 Spuren mal 16 Sektoren).

2.2. Befehlsnamen

Ähnlich wie bei Applesoft (z.B. HOME, NEW, PRINT usw.) gibt es bei DOS zahlreiche Befehlsnamen (z.B. CATALOG, LOCK, UNLOCK, VERIFY usw.). Im einzelnen sind folgende Gemeinsamkeiten und Unterschiede zu beachten:

a) Applesoft-Befehlsnamen werden speicherintern als sog. Tokens (= 1-Byte-Verschlüsselungen) gepackt gespeichert; z.B. ist der interne Code $BA (dezimal 186) das Token für PRINT. Nur während der Eingabe einer neuen Programmzeile oder während des Listens eines Programms erscheinen die Tokens „ausgeschrieben". Umgekehrt werden die DOS-Befehlsnamen als Bestandteile eines Applesoft-Programms stets „ausgeschrieben", d.h befinden sich so im Speicher, wie man sie beim Listen sieht (als ASCII-Folge mit Bit 7 off). Dafür können DOS-Befehlsnamen als String-Variablen deklariert werden, was bei Applesoft-Befehlsnamen nicht möglich ist. Beispiel:

```
10 L$ = „LIST" : REM nicht erlaubt!
20 C$ = „CATALOG": REM erlaubt
```

b) Genau wie bei Applesoft müssen auch bei DOS die Befehlsnamen in Großbuchstaben geschrieben werden, also List und Catalog falsch, LIST und CATALOG richtig. DOS-Befehlsnamen sind 2 - 8 Zeichen lang.

2.3. Direkte und indirekte DOS-Befehle

Ähnlich wie bei Applesoft gliedert man bei DOS die Befehle in zwei Gruppen:

2.3.1. Direkt- UND Indirektbefehle

Direkt- UND Indirektbefehle lassen sich SOWOHL über die Tastatur ALS AUCH aus Applesoft-Programmen (sowie Integer- und Assembler-Programmen) heraus geben.

Direktbefehl (Tastatureingabe-Modus, Nicht-RUN-Modus):

Der Tastatureingabe-Modus (immediate mode) liegt dann vor, wenn sich das „Prompt" Ù (oder beim amerikanischen Apple die eckige Klammer) am linken Bildschirmrand befindet und dahinter der „Cursor" (Schreibmarke) blinkt.

Der DOS-Direktbefehl unterscheidet sich vom Applesoft-Direktbefehl durch folgendes:

— Im Applesoft können mehrere Direktbefehle gleichzeitig in einer Zeile eingegeben werden, z.B.

HOME : PRINT 7 + 3 : LIST 10
Bei DOS-Befehlen ist dies nicht möglich; z.B. würde
CATALOG : CATALOG
zu einer Fehlermeldung führen.

— Darüber hinaus können DOS-Direktbefehle nur nach einem Return eingegeben werden, d.h. der Cursor (Ü bei Applesoft) muß sich am linken Bildschirmrand befinden.

Indirektbefehl (Programm-Modus, RUN-Modus):

Einem DOS-Indirektbefehl muß stets ein Return oder CHR$(13) und ein Ctrl-D oder CHR$(4) vorangehen und der Befehl muß in Anführungszeichen gesetzt und als PRINT-Befehl ausgeführt werden, der mit einem Return endet.

Richtige Beispiele:

```
10 D$ = CHR$(4) : PRINT D$;,,CATALOG"
```

oder

```
10 D$ = CHR$(4) : PRINT D$,,CATALOG" : REM Semikolon zwischen D$ und ,,CATALOG" darf fehlen!
```

oder

```
10 PRINT CHR$(4);,,CATALOG"
```

oder

```
10 PRINT CHR$(4),,CATALOG"
```

oder

```
10 PRINT ,,DCATALOG" : REM Hier steht das D für ein eingegebenes Ctrl-D
```

oder

```
10 X$ = CHR$(4) + ,,CATALOG" : PRINT X$ : REM Variablen-Zuweisung erlaubt!
```

oder

```
10 D$ = CHR$(4) : C$ = ,,CATALOG" : PRINT D$;C$
```

oder

```
10 PRINT CHR$(4) ,,CATALOG" : PRINT CHR$(4) ,,CATALOG" : REM
```

Mehrfachbefehle in einer Programmzeile erlaubt!

Falsche Beispiele:

```
10 PRINT ,,CATALOG" : REM Ctrl-D fehlt!
```

oder

```
10 CATALOG : REM PRINT, Ctrl-D und Anführungszeichen fehlen!
```

oder

```
10 PRINT CHR$(4);,,CATALOG"; : REM Semikolon nach CATALOG unter-
```

drückt Return!

Folgende DOS-Befehle können sowohl in Direkt- wie Indirektmodus gegeben werden:

LOAD
RUN
SAVE
BLOAD
BRUN
BSAVE
CATALOG
VERIFY
RENAME
DELETE
LOCK
UNLOCK
MON
NOMON
PR# + Slot

IN # + Slot
EXEC
CLOSE

CHAIN (nur Integer-Basic!)
INIT (!)
MAXFILES (!)
FP (!)
INT (!)

Bei den mit Ausrufezeichen markierten Befehlsnamen ist Vorsicht im Programm-Modus geboten. Näheres steht bei den einzelnen Befehlserläuterungen.

2.3.2. NUR Indirektbefehle

Reine Indirektbefehle lassen sich nur aus dem Programm heraus geben. Es sind dies die Befehle:

OPEN
READ
WRITE
POSITION
APPEND

Es handelt sich hierbei ausschließlich um sog. Textfile-Befehle.

2.4. Parameter Slot, Drive, Volume

DOS-Befehle können durch sog. Parameter näher bestimmt werden. Es sind dies die allgemeinen Parameter Slot (S), Drive (D) und Volume (V) sowie einige befehlsspezifische Parameter wie L(ength) usw., die bei den einzelnen Befehlen behandelt werden. Die Reihenfolge der Parameter ist gleichgültig. Das Betriebssystem merkt sich die zuletzt definierten Parameter, so daß man die Parameter nicht ständig wiederholen muß. Beispiele:

CATALOG, S6,D1 oder
CATALOG, D1,S6

Wie ersichtlich, werden die Parameter vom Befehlsnamen durch Kommas abgegrenzt. Aus diesem Grunde darf ein Dateiname niemals ein Komma enthalten!

1. Slot = S (Steckplatz; Werte 1-7)

Der Slot-Parameter ist im Bereich 1-7 (beim Apple II Plus wegen einer möglichen RGB-Karte in Slot 7 nur im Bereich 1-6) möglich, z.B.

CATALOG, S4

2. Drive = D (Laufwerk, Werte 1-2)

Der Drive-Parameter kann 1 oder 2 sein, z.B.

CATALOG, D1
CATALOG, D2

3. Volume = V (Disketten-Nummer, Werte 0-254)

Eine Diskette kann eine Nummer erhalten, die ausschließlich beim Initialisieren festgelegt wird und nachträglich praktisch nicht mehr geändert werden kann (weil die Volume-Nummer im sog. Adreßfeld, einem Art Vorspann zu jedem 256-Byte-Sektor, beim Formatieren verankert wird.)

Der zulässige Bereich für Volume-Nummern ist 1-254, z.B.

INIT MEIER, V1
INIT MEIER, V10

Sonderfall:

INIT MEIER, V0
INIT MEIER (ohne Parameter)

ergibt stets die Volume-Nummer 254.

Der Nutzen der Volume-Nummer ist gering, da DOS nicht in der Lage ist, automatisch zu ermitteln, auf welchem Slot und Drive sich eine mit einer Volume-Nummer spezifizierte Diskette befindet. (Dies ist z.B. bei Pascal und ProDOS in bezug auf die Volume-Names möglich.)

Die Volume-Nummer spielt dagegen bei Festplattenlaufwerken (z.B. Corvus) eine große Rolle, da z.B. die Corvus in 50-100 Volumes eingeteilt ist, je nach Größe des Plattenspeichers.

Die Parameter S, D und V sind bei fast allen Befehlen möglich. Ausnahmen:

- bei CATALOG wird der V-Parameter ignoriert
- bei bestimmten Befehlen sind die Parameter weder sinnvoll noch möglich, z.B. MON, NOMON, MAXFILES
- bei CLOSE sind Parameter nicht erforderlich (und auch nicht zulässig), da dieser Befehl als einziger vollautomatisch alle Dateien auf allen Laufwerken schließen kann.
- als sehr lästig erweist es sich, daß bei den READ- und WRITE-Befehlen keine Parameter zulässig sind (wie man dies umgeht, siehe weiter unten)

Gesamtaufstellung (in Beispielen):

```
INIT,S6,D1,V1
CATALOG,S5,D2 (kein V)
RENAME ALT,NEU,S6,D1,V1
DELETE ALT,S2,D2,V100
LOCK DATEI,S6,D1,V1
UNLOCK DATEI,S6,D1,V1
VERIFY DATEI,S6,D2,V0

SAVE PROGRAMM,S6,D1,V5
LOAD PROGRAMM,S5,D2,V0
RUN HELLO,S6,D1,V254
BSAVE BILD,A$4000,L$2000,S6,D1,V1
BLOAD BILD,S6,D1,V1
BRUN MASCHINENPROGRAMM,S6,D1,V1
EXEC EXECUTIVE.FILE,S6,D2,V10
CHAIN INTEGER.PROGRAMM,S6,D1,V1

MON (kein S,D,V)
NOMON (kein S,D,V)
MAXFILES 3 (kein S,D,V)
FP (normal kein S,D,V)
INT (normal kein S,D,V)
PR#6 (nur S)
IN#2 (nur S)
```

```
10 PRINT CHR$(4);,,OPEN DATEI,S6,D1,V1"
10 PRINT CHR$(4);,,CLOSE DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,WRITE DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,READ DATEI" (kein S,D,V)
10 PRINT CHR$(4);,,APPEND DATEI,S6,D1,V1"
10 PRINT CHR$(4);,,POSITION DATEI" (kein S,D,V)
```

2.5. Einfache DOS-Befehle

Als einfach bzw. unkompliziert können alle diejenigen DOS-Befehle bezeichnet werden, die nicht mit Textfiles zusammenhängen.

2.5.1. INIT, CATALOG, LOCK, UNLOCK, VERIFY, DELETE, RENAME

INIT dient — wie bereits geschildert — zum Initialisieren fabrikneuer oder zum Reinitialisieren bereits früher initialisierter (= formatierter) Disketten:

```
10 PRINT ,,ZU FORMATIERENDE LEERDISKETTE EINLEGEN!"
20 INPUT ,,WELCHE VOLUME-NUMMER (1-254):";X$
30 PRINT CHR$(4),,INIT DATENDISK";,, ,V";VAL(X$)
40 PRINT CHR$(4),,DELETE DATENDISK"
50 GOTO 20
```

Dieses kleine Programm würde jeweils nach der gewünschten Volume-Nummer fragen und dann die eingelegte Leerdiskette initialisieren und im Anschluß daran den Hello-Namen (= Boot-Programm-Namen) wieder löschen. Wenn sich auf einer Diskette kein Hello-Programm mehr befindet, bootet die Diskette trotzdem korrekt, doch erfolgt dann die Meldung ,,FILE NOT FOUND", weil dieses Programm ja jetzt nicht mehr existiert.

Wie aus diesem Beispiel ersichtlich, ist es zwar möglich, aus einem Programm heraus zu initialisieren, doch sollte man unbedingt eine Warnmeldung einbauen, damit nicht aus Versehen eine wertvolle Datendiskette durch die Initialisierung zerstört wird.

Die Befehle CATALOG, LOCK, UNLOCK und VERIFY sind bereits bekannt.

Auch die Befehle RENAME (Dateiname umbenennen) und DELETE (Datei löschen) verstehen sich von selbst. Die Syntax ist bei RENAME:

RENAME ALTER NAME, NEUER NAME

RENAME ist im DOS nicht narrensicher implementiert mit der Folge, daß es möglich ist, mit RENAME zwei gleiche Dateinamen auf der Diskette zu erzeugen. Nehmen wir an, die Diskette enthalte bereits die Files

PROGRAMM.1
PROGRAMM.2

Mit RENAME PROGRAMM.1,PROGRAMM.2 würde man dann auf der Diskette zwei gleiche Dateinamen erzeugen:

PROGRAMM.2
PROGRAMM.2

Da die LOAD-, BLOAD-Befehle usw. die Suche nach dem zuerst gefundenen Dateinamen abbrechen, würde man das ehemalige, eigentliche PROGRAMM.2 niemals mehr einladen können. Man muß dann schleunigst den oberen Dateinamen mit RENAME PROGRAMM.2,PROGRAMM.1 wieder zurückbenennen.

Demo-Programm:

```
100 HOME : INPUT ,,DISKETTE EINLEGEN ";X$
110 PRINT CHR$(4),,SAVE XXX"
120 PRINT CHR$(4),,CATALOG"
130 PRINT CHR$(4),,LOCK XXX"
140 PRINT CHR$(4),,CATALOG"
150 PRINT CHR$(4),,UNLOCK XXX"
160 PRINT CHR$(4),,CATALOG XXX"
170 PRINT CHR$(4),,RENAME XXX,YYY"
180 PRINT CHR$(4),,CATALOG"
190 PRINT CHR$(4),,DELETE YYY"
200 PRINT CHR$(4),,CATALOG"
210 PRINT ,,JETZT KOMMT 'FILE NOT FOUND'-MELDUNG"
220 PRINT CHR$(4),,VERIFY YYY"
```

Warum kommt am Ende dieses Programmes eine Fehlermeldung?

2.5.2. FP und INT

FP steht für Floating Point Basic (= Fließkomma-Basic = Applesoft) und INT steht für Integer Basic (= Ganzzahl-Basic). Der Apple IIe sowie der Apple II Plus haben Applesoft im ROM (im Bereich $D000-$F7FF), während der Uralt-Apple II Integer-Basic im ROM hatte. Die „System Master"-Diskette lädt automatisch nach dem Booten Integer-Basic in die Language Card (RAM-Bereich $D000-$F7FF). Mit INT kann man dann zu Integer umschalten, und mit FP wieder zu Applesoft zurückschalten. Da Integer-Basic meistens nicht mehr benutzt wird, hat der FP-Befehl heute eine andere Bedeutung. Wenn man sich im Applesoft-Modus befindet und FP eingibt, dann wird

1. die Speicherstelle 2048 (hexadezimal $0800) gelöscht (Dies macht NEW nicht. Wäre hier ein anderer Wert außer 0, dann würde RUN nicht mehr funktionieren.)
2. HIMEM (Highest Memory = Obergrenze des freien Speichers) auf 38400 (hexadezimal $9600) zurückgesetzt, womit gleichzeitig MAXFILES 3 eingestellt wird, und
3. der NEW-Befehl ausgeführt.

FP ist damit wirkungsvoller als NEW allein. FP ist praktisch der einzige Befehl, der HIMEM normalisiert. (Falls sich das DOS selbst in der Language Card befindet, bewirkt FP in der Regel HIMEM $BF00.) Der String-Pool von Applesoft beginnt übrigens bei HIMEM-1, also bei $95FF.

FP und INT als Indirektbefehle bewirken den sofortigen Programmabbruch und das Löschen des jeweiligen Programms, z.B.

```
1000 PRINT CHR$(4),,FP"
```

2.5.3. MAXFILES

MAXFILES heißt „maximum number of files" = maximale Puffer- und damit Dateianzahl. Für jede geöffnete Datei wird ein 595 Bytes umfassender Zwischenpuffer angelegt. Die normalerweise 3 automatisch nach dem Booten angelegten Puffer befinden sich in dem Bereich

$9600-$9CF8 (dezimal 38400-40184); (40184-38400 + 1) : 595 = 3

Wenn Befehle wie CATALOG, LOAD, VERIFY usw. ausgeführt werden, wird vorübergehend 1 Puffer benötigt. Der zuerst belegte Puffer ist der Puffer ab $9600.

Nach Beendigung des CATALOG-Befehls usw. wird der Puffer wieder frei. Mit dem Befehl MAXFILES + Zahl, z.B.

```
MAXFILES 5
```

kann man die Anzahl der Puffer festlegen. Es sind maximal 16 Puffer möglich, wobei z.B. bei MAXFILES 4 HIMEM um 595 Bytes niedriger gesetzt wird als $9600, d.h. je mehr Puffer, desto weniger freier Speicherraum. (Wenn DOS in die Language Card verschoben wurde, hat man in der Regel nur maximal 5 Puffer zur Verfügung, die automatisch ständig benutzt werden können, so daß hier der MAXFILES-Befehl wirkungslos ist.)
Im einzelnen gilt für normales DOS in den unteren 48K:

MAXFILES 1 = HIMEM: 39590 = $9AA6
MAXFILES 2 = HIMEM: 38995 = $9853
MAXFILES 3 = HIMEM: 38400 = $9600 (Normalfall)

Der momentane Wert von HIMEM läßt sich übrigens ermitteln mit:

```
PRINT PEEK (115) + PEEK (116) · 256
```

MAXFILES darf aus einem Programm heraus nur dann geändert werden, wenn zuvor entweder keine Strings definiert wurden oder der Variablen-Speicher mit CLEAR gelöscht wurde. MAXFILES definiere man deshalb am besten als erste Zeile des Applesoft-Programms, z.B.:

```
10 CLEAR : PRINT CHR$(4),,MAXFILES 1"
```

MAXFILES spielt insbesondere bei Textfiles eine Rolle, da für jeden durch OPEN geöffneten Textfile ein Puffer eingerichtet wird, z.B.:

```
10 CLEAR : PRINT CHR$(4),,MAXFILES 4"
20 PRINT CHR$(4),,OPEN DATEI.1"
30 PRINT CHR$(4),,OPEN DATEI.2"
40 PRINT CHR$(4),,OPEN DATEI.3"
50 PRINT CHR$(4),,OPEN DATEI.4"
60 PRINT ,,Achtung:"
70 PRINT ,,Jetzt kommt Meldung"
80 PRINT ,,NO BUFFERS AVAILABLE"
90 PRINT CHR$(4),,CATALOG"
```

Da hier nur 4 Puffer angelegt wurden, war kein Puffer mehr für den CATALOG-Befehl zur Verfügung.

2.5.4. MON und NOMON

MON kommt von „to monitor" = überwachen. MON bedeutet DOS-Befehle überwachen, NOMON DOS-Befehle nicht mehr überwachen. Mit Überwachen ist gemeint, daß bei einem laufenden Programm die DOS-Befehle sowie bei Textfiles darüber hinaus die eingelesenen oder gespeicherten Daten am Bildschirm sichtbar werden. Es gibt folgende spezifische Parameter:

C = Command = DOS-Befehle werden sichtbar
I = Input = eingelesene Textfiles werden sichtbar (während des Einlesens)
O = Output = gespeicherte Textfiles werden sichtbar (während des Speicherns)

Beispiele:

MON C,I,O
NOMON C,I,O (auch NOMONCIO, d.h. Kommas entbehrlich)
MON C
NOMON C
usw.

Man beachte, daß die Befehle MON und NOMON ohne Parameter wirkungslos sind.

2.5.5. PR#S und IN#S

Diese Befehle steuern den Output (PR # + Slot) und Input (IN # + Slot) von Peripheriegeräten. Beispiele:

- mit PR#1 wird der Drucker aktiviert, falls in Slot 1 ein Drucker-Interface steckt und der Drucker bereits eingeschaltet wurde, d.h. auf ON LINE steht.
- mit PR#6 oder auch IN#6 wird das DOS gebootet, falls sich ein DOS-Controller in Slot 6 befindet.
- mit PR#3 wird die 80-Zeichen-Karte eingeschaltet, falls sich eine solche in Slot 3 befindet, usw.

Aus dem Programm heraus sollten diese Befehle stets mit Ctrl-D ausgeführt werden.

da sonst die sog. DOS-Vektoren und damit das DOS ,,abgehängt" werden.

```
10 PRINT CHR$(4),,PR#1"
```

würde den Drucker einschalten und gleichzeitig das DOS angeschlossen lassen.

```
10 PR#0 : IN#0 : PR#1
```

würde den Drucker zwar ebenfalls einschalten, aber das DOS komplett abhängen.

```
10 PR#0 : IN#0 : CALL 1002
```

würde dann nach Beendigung des Druckvorgangs das DOS wieder ordnungsgemäß ,,anhängen".
Das planmäßige, vorübergehende ,,Abhängen" des DOS mit PR#0 : IN#0 und das spätere ,,Anhängen" mit PR#0 : IN#0 : CALL 1002 hat den Vorteil, daß die Bildschirm- und Druckergeschwindigkeit während des abgeschalteten DOS größer ist. Weiteres Beispiel:

```
10 PR# 0 : IN#0 : REM DOS ABGEHÄNGT
20 FOR X = 1 TO 10
30 PRINT ,,AAAAAAAAAA"
40 NEXT X
50 PR#1
60 FOR X = 1 TO 10
70 PRINT ,,AAAAAAAAAA" : NEXT X
80 PR#0 : IN#0
90 PRINT CHR$(4) ,,CATALOG" : REM WIRKUNGSLOS
100 PR#0 : IN#0 : CALL 1002 : PRINT CHR$(4) ,,CATALOG"
```

2.5.6. LOAD, RUN, SAVE (Basicfiles)

LOAD, RUN und SAVE sind die DOS-Befehle zum Laden (= LOAD), Laden UND Starten (= RUN) sowie zum Speichern (= SAVE) von Applesoft- und Integer-Programmen (= Basicfiles). Die Handhabung dieser Befehle ist denkbar einfach, da das DOS die Berechnung der Länge des jeweiligen Programms selbst übernimmt. Beispiele:

```
LOAD HELLO
```

```
RUN HELLO
SAVE HELLO

10 PRINT ,,DIES IST PROGRAMM.1"
1000 PRINT CHR$(4) ,,RUN PROGRAMM.2"
```

Programme lassen sich aus einem Program heraus starten. Allerdings gehen dadurch alle Variablen von Programm.1 verloren.

```
10 PRINT ,,DIES IST EIN PROGRAMM, DAS SICH SELBST SPEICHERT"
20 PRINT CHR$(4) ,,SAVE PROGRAMM"
```

Programme lassen sich aus dem Programm heraus selbst speichern, doch ist dies normalerweise nicht besonders sinnvoll.

Basic-Programme werden normalerweise ab 2048 + 1 (= $0800 + 1) in den RAM-Speicher geladen. Start- und Endadresse eines Applesoft-Programms kann man bei Bedarf manuell ermitteln durch folgende Formeln:

Startadresse: PRINT PEEK (103) + PEEK (104) · 256
Endadresse: PRINT PEEK (175) + PEEK (176) · 256

Auf der Diskette werden Basic-Programme als Speicherauszüge ab $0801 bis zum Programmende inclusive 4 Hex-Codes $00 nach dem Programmende und mit einem 2 Bytes langen Vorspann, der die Länge beinhaltet (in hexadezimal, Low Byte — High Byte), gespeichert. (Nach den 4 Hex $00 wird infolge eines DOS-Fehlers 1 weiteres Byte gespeichert.)

Basicfiles sind aus dem Catalog wie folgt ersichtlich:

A steht für Applesoft
I steht für Integer

Dagegen steht B für Binärfile und T für Textfile.

2.5.7. BLOAD, BRUN und BSAVE (Binärfiles)

Diese Befehle beziehen sich auf sog. Binärfiles, d.h. auf Maschinenprogramme sowie reine Speicherauszüge, z.B. Hires-Bilder, Grafik-Shapes usw. BLOAD steht für bi-

näres Laden, BRUN für binäres RUN (ausschließlich von Maschinenprogrammen) und BSAVE steht für binäres Speichern.
Der BSAVE-Befehl verlangt als Parameter die Startadresse (A) sowie die Länge (L) in entweder dezimaler oder hexadezimaler Schreibweise (auch gemischt möglich), z.B.:

BSAVE BILD, A$2000, L$2000 oder
BSAVE BILD, A8192, L8192 oder
BSAVE BILD, A$2000, L8192 oder
BSAVE BILD, A8192, L$2000

Angenommen, im RAM ab $0300 sei Nachstehendes gespeichert:

0300: 01 02 03 04 05 06

Wollte man dies auf der Diskette speichern, so würde folgender Befehl genügen:

BSAVE TEST,A$0300,L$0006 oder kürzer
BSAVE TEST,A$300,L$6 (Führende Nullen können weggelassen werden.)

Der entsprechende Disketten-Sektor würde dann so aussehen:

00 03 06 00 01 02 03 04 05 06

Die ersten zwei Bytes sind die Startadresse (00 03 — 0300 — $0300 — also zuerst Low Byte, dann High Byte)
Die nächsten zwei Bytes sind die Länge (06 00 — 00 06 — $0006 — wieder Low Byte first)

Für vierstellige, hexadezimale Zahlen im Bereich $0000-$FFFF gilt folgende Umrechnungsformel:

Low Byte + (High Byte · 256)

Mit BLOAD wird der Binärfile genau an diejenige Stelle eingeladen, die beim vorangehenden BSAVE spezifiziert wurde. Man kann einen Binärfile allerdings auch an eine andere Stelle bloaden durch Spezifikation einer neuen Startadresse. Leider gibt es beim DOS keinen Befehl, der die Startadresse und Länge eines Binärfiles anzeigt, so daß man mit den folgenden Formeln die Berechnung selbst vornehmen muß:

BLOAD-Startadresse: PRINT PEEK (43634) + PEEK (43635) · 256
BLOAD-Länge: PRINT PEEK (43616) + PEEK (43617) · 256 (vgl. S. 67!)

Wenn das DOS in die Language Card verschoben wurde, wird in der Regel anstelle des INIT-Befehls ein entsprechender Befehl implementiert, z.B. PAD bei Diversi-DOS.

Der BRUN-Befehl darf nur bei Maschinenprogrammen angewandt werden. BRUN lädt zunächst den Binärfile und springt dann zur Startadresse.

Die Länge eines Binärfiles darf 32767 (= $7FFF) normalerweise nicht überschreiten. Die Startadresse kann im Bereich 0-65535 ($0000-$FFFF) liegen.

Als freie Bereiche für Binärfiles kommen in Frage:

$0200-$02FF (nur für vorübergehend benötigte Startprogramme, da dies der Tastatureingabepuffer ist)
$0300-$03CF
$0803-$095FF (Maschinenprogramme sollte man besser ab $803 statt $800 beginnen lassen, da Applesoft dann „denkt", daß kein Programm geladen wurde.)

Wenn sich DOS in der Language Card befindet, kann man auch noch den Bereich $9600-$BEFF benutzen. Umgekehrt, wenn sich DOS nicht in der Language Card befindet, stehen die oberen 16K des RAM-Speichers ebenfalls für Maschinenprogramme zur Verfügung.

Zusammenfassend gilt für Applesoft-, Integer- und Binär-Files folgender „Vorspann" auf dem ersten Datensektor des Files auf der Diskette:

1. Integer: Low Byte + High Byte der Länge,
danach Programm,
danach 4 Hex-Codes 00

2. Applesoft: Low Byte + High Byte der Länge,
danach Programm,
danach 4 Hex-Codes 00

3. Binärfile: Low Byte + High Byte der Startadresse,
Low Byte + High Byte der Länge,
danach eigentlicher Binärfile

2.5.8. CHAIN

Der CHAIN-Befehl sei hier nur der Vollständigkeit halber erwähnt. Er gilt nur für das kaum noch benutzte Integer-Basic und dient dem Zweck, aus einem Integer-Programm heraus ein weiteres Integer-Programm mit RUN zu starten, ohne daß dadurch die Variablen zerstört werden, z.B.:

```
1000 PRINT ,,D CHAIN ZWEITPROGRAMM" : REM D = CTRL-D
```

Für Applesoft befindet sich auf der ,,System Master"-Diskette eine ähnliche Maschinen-Routine, die jedoch nicht immer funktioniert. Auch bei Applesoft-Compilern wie TASC usw. funktioniert das ,,Chaining" nur teilweise. Für Besitzer von RAM-Karten, z.B. der Apple IIe 64K Karte, gibt es eine viel sicherere Lösung. Man speichert die Variablen auf der schnellen RAM-Disk, startet das nächste Programm-Modul und lädt schließlich die zwischengespeicherten Variablen von der RAM-Karte wieder zurück. Ein RAM-Disk-Driver ist im Teil II dieses Buches als Listing abgedruckt.

2.6. Textfile-Befehle

2.6.1. OPEN, READ, WRITE, CLOSE

Informationstheoretisch gesehen gibt es zwei grundsätzlich verschiedene Arten von Dateien, nämlich

1. Programm-Dateien, z.B. Applesoft-, Integer- und Assemblerprogramme, sowie
2. Daten-Dateien, z.B. Adressen, Briefe, Bilder, Meßwerte usw.

Basicfiles (A, I) sind stets Programm-Dateien. Binärfiles sind demgegenüber teils Programm-Dateien (z.B. Assemblerprogramme) und teils Daten-Dateien, z.B. Hires-Bilder, binär abgespeicherte Zahlen-Arrays usw.

Die typische und zugleich wichtigste Daten-Datei für sowohl Applesoft- wie auch Maschinen-Programme ist der sog. Textfile. Textfile bedeutet wörtlich Text-Datei (String-Datei, Zeichenketten-Datei), doch können Textfiles auch Zahlen-Dateien beinhalten.

Beispiel: Strings

AAAAA
BBBBB

```
10 PRINT CHR$(4) ,,OPEN STRINGS"
20 PRINT CHR$(4) ,,WRITE STRINGS"
30 PRINT ,,AAAAA"
40 PRINT ,,BBBBB"
50 PRINT CHR$(4) ,,CLOSE"
```

```
10 PRINT CHR$(4) ,,OPEN STRINGS"
20 PRINT CHR$(4) ,,READ STRINGS"
30 INPUT A$
40 INPUT B$
50 PRINT CHR$(4) ,,CLOSE"
```

Beispiel: Zahlen

11111
22222

```
10 PRINT CHR$(4) ,,OPEN ZAHLEN"
20 PRINT CHR$(4) ,,WRITE ZAHLEN"
30 PRINT 11111
40 PRINT 22222
50 PRINT CHR$(4) ,,CLOSE"
```

```
10 PRINT CHR$(4) ,,OPEN ZAHLEN"
20 PRINT CHR$(4) ,,READ ZAHLEN"
30 INPUT Z1
40 INPUT Z2
50 PRINT CHR$(4) ,,CLOSE"
```

Aus den Beispielen können wir folgendes ersehen:

1. Ein Textfile, gleichviel ob er neu angelegt wird oder bereits existiert, d.h. bereits früher angelegt wurde, wird mit OPEN + DATEINAME geöffnet. Wenn eine Datei bereits angelegt war und nunmehr durch neue Werte überschrieben werden soll, so empfiehlt es sich, falls der neue Datei-Inhalt kleiner als der alte

wird, die Altdatei zunächst zu löschen, da sonst die Neudatei auf der Diskette den gleichen Raum einnehmen wird wie die Altdatei. Beispiel:

```
10 REM ALTDATEI MIT 1000 STRINGS
20 PRINT CHR$(4) ,,OPEN TEST"
30 PRINT CHR$(4) ,,WRITE TEST"
40 FOR X = 1 TO 1000 : PRINT ,,AAAAA" : NEXT X
50 PRINT CHR$(4) ,,CLOSE"

10 REM NEUDATEI MIT 500 STRINGS
20 PRINT CHR$(4) ,,OPEN TEST"
30 PRINT CHR$(4) ,,DELETE TEST"
40 PRINT CHR$(4) ,,OPEN TEST"
50 PRINT CHR$(4) ,,WRITE TEST"
60 FOR X = 1 TO 500 : PRINT ,,AAAAA" : NEXT X
70 PRINT CHR$(4) ,,CLOSE"
```

Bemerkungen:

a) Die Zeile 20 bei dem Neudatei-Programm kann entfallen, falls man sicher ist, daß sich auf der Diskette bereits eine Datei dieses Namens befindet. Ist man sich jedoch unsicher, ob die TEST-Datei bereits existiert, dann kann man mit OPEN — DELETE — OPEN die ,,FILE NOT FOUND"-Fehlermeldung umgehen.

b) Hätte man die Neudatei nicht zunächst gelöscht, dann würde die Neudatei weiterhin auf der Diskette denselben Raum wie die Altdatei einnehmen, weil DOS nach dem CLOSE die nunmehr überflüssigen, restlichen Altdatei-Datensektoren nicht von selbst löscht.

2. Nach dem OPEN-Befehl kann wahlweise ein WRITE- oder READ-Befehl folgen, je nachdem, ob auf die Datei geschrieben oder von ihr gelesen werden soll.

3. Eine Datei wird normalerweise mit PRINT-Befehlen beschrieben und mit INPUT-Befehlen gelesen.

4. Eine Datei muß mit dem CLOSE-Befehl stets geschlossen werden, auch dann, wenn nur von ihr gelesen wurde. Zwar gehen bei Dateien, aus denen nur gelesen wird, keine Daten auf der Diskette verloren, wenn der CLOSE-Befehl vergessen wurde, doch bleibt dann der für diese Datei bereitgestellte DOS-Puffer belegt. Öffnet man z.B. bei Maxfiles 3 den vierten Textfile, ohne zuvor eine der

ersten drei geöffneten Textfiles wieder zu schließen, folgt die Meldung „NO BUFFERS AVAILABLE" (= keine Puffer mehr frei). Ein fehlender CLOSE-Befehl kann auch nach Beendigung des Programms manuell nachgeholt werden, indem man über die Tastatur CLOSE tippt. CLOSE allein schließt ALLE noch offenen Dateien, CLOSE + DATEINAME schließt nur diese letztere Datei. Beispiel:

```
10 PRINT CHR$(4) „OPEN DATEI.1"
15 PRINT 1
20 PRINT CHR$(4) „OPEN DATEI.2"
25 PRINT 2
30 PRINT CHR$(4) „OPEN DATEI.3"
35 PRINT 3
40 PRINT CHR$(4) „CLOSE DATEI.1" : REM Schließt nur diese Datei.1
```

Was aus den Beispielen nicht unmittelbar evident, jedoch für die Korrektheit der Textfile-Befehle erforderlich ist, ist der Umstand, daß die OPEN-, READ- und WRITE-Befehle nur dann wirksam werden, wenn ihnen nicht nur ein Ctrl-D, sondern auch ein Ctrl-M bzw. Return vorausgeht.

Falsches Beispiel 1:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 PRINT „AAAAA"; : REM Semikolon!
40 PRINT CHR$(4) „CLOSE"
```

Kommentar: Der letzte PRINT-Befehl vor dem CLOSE-Befehl darf nicht mit einem Semikolon abschließen, da sonst der CLOSE-Befehl nicht ausgeführt wird. Beim Beispiel 1 würde auf die Diskette statt „AAAAA" in Wirklichkeit „AAAAA-CLOSE" geschrieben, d.h. Ctrl-D + „CLOSE" würde an den String „AAAAA" angehängt werden!

Richtiges Beispiel 1:

```
10 PRINT CHR$(4) „OPEN TEST"
20 PRINT CHR$(4) „WRITE TEST"
30 FOR X = 1 TO 3 : PRINT „AAAAA"; : NEXT X
40 PRINT : PRINT CHR$(4) „CLOSE"
```

Kommentar: Mehrere PRINT-Befehle mit Semikolon zur Unterdrückung des Returns sind zulässig, doch muß der letzte PRINT-Befehl ein Return beinhalten.

Falsches Beispiel 2:

```
10 PRINT ,,SPEICHERN J/N “;: GET X$
20 IF X$ = ,,N“ THEN END
30 PRINT CHR$(4) ,,OPEN TEST“
usw.
```

Kommentar: Der GET-Befehl unterdrückt das Return mit der Folge, daß ,,OPEN TEST“ am Bildschirm angezeigt, aber nicht vom DOS als Befehl erkannt wird.

Richtiges Beispiel 2:

```
10 PRINT ,,SPEICHERN J/N “; : GET X$
20 IF X$ = ,,N“ THEN END
30 PRINT : PRINT CHR$(4) ,,OPEN TEST“
```

Kommentar: Hier steht in Zeile 30 ein zusätzliches PRINT, so daß der DOS-Syntax Genüge getan wird.

2.6.2. Struktur eines Textfiles

Wenn auf die Diskette folgende Datei geschrieben wird

```
10 PRINT CHR$(4) ,,OPEN FILE“
20 PRINT CHR$(4) ,,WRITE FILE“
30 PRINT ,,aaaaa“
40 PRINT ,,bbbbb“
50 PRINT ,,ccccc“
60 PRINT CHR$(4) ,,CLOSE“
```

dann hat diese Datei die Struktur:

aaaaa	+ Return	Feld 1
bbbbb	+ Return	Feld 2
ccccc	+ Return	Feld 3
	+ Ctrl-0	Endmarker

Durch Return abgegrenzte ASCII-Folgen (Strings oder Zahlen) werden als Felder bezeichnet.

Return hat den ASCII-Code 13 oder 141 (hexadezimal $0D oder $8D) und Ctrl-0 hat den ASCII-Code 0 oder 128 (oder hexadezimal $00 oder $80). ASCII ist die Abkürzung für American Standard Code for Information Interchange, d.h. amerikanischer Datenaustausch-Code. Die Tabelle im Anhang zu diesem Buch enthält eine erweiterte ASCII-Code-Aufstellung mit den entsprechenden hexadezimalen, binären und dezimalen Code-Werten. Bereits ein flüchtiger Blick auf diese Tabelle genügt um festzustellen, daß die Tabelle sozusagen doppelt belegt ist, d.h. die Werte von 0-127 werden durch die Werte von 128-255 wiederholt. Bei der ersten Hälfte ist das linke Bit der 8-Bit-Binärverschlüsselung stets 0, bei der zweiten Hälfte stets 1. Dieses linke Bit wird Bit 7 genannt, da die Bits von rechts nach links — beginnend mit 0 — gezählt werden. Beispiel:

76543210 = Bit-Positionen
11110000 = $F0 = p

Insgesamt gibt es also 2 mal 128 verschiedene Zeichen bzw. Codes. Während Applesoft im RAM-Speicher Strings in der Regel mit „Bit 7 off" (Bit 7 = 0; linke Hälfte) speichert, werden Strings auf der Diskette in der Regel mit „Bit 7 on" (Bit 7 = 1; rechte Hälfte) gespeichert. Return hat damit den Wert 141 (oder $8D). Dagegen wird als Ausnahme Ctrl-0 auf der Diskette stets als 0 (oder $00 oder 00000000) gespeichert. Ctrl-0 (auch NUL genannt) ist der Endmarker (= die Endmarke = die Endmarkierung) für einen Textfile. Symbolisch würde die obengenannte Datei auf der Diskette also wie folgt gespeichert:

a a a a a R b b b b b R c c c c c R 0

R = Return = Ctrl-M
0 = Ctrl-0 = NUL

Und hexadezimal verschlüsselt würde die Datei so aussehen:

E1 E1 E1 E1 E1 8D E2 E2 E2 E2 E2 8D E3 E3 E3 E3 E3 8D 00

(Die Leertasten sind in beiden Fällen nur aus Gründen der besseren Lesbarkeit eingefügt.)

Bei der ASCII-Tabelle sind die Werte 160-254 (oder $A0-$FE) die sogenannten sichtbaren oder druckbaren Zeichen:

$A0-$AF:	diverse Sonderzeichen
$B0-$B9:	Ziffern
$BA-$C0:	diverse Sonderzeichen
$C0-$DA:	großes Alphabet
$DB-$E0:	länderspezifische Zeichen (Umlaute)
$E1-$FA:	kleines Alphabet
$FB-$FE:	länderspezifische Zeichen (Umlaute)

Daneben gibt es noch die Ctrl-Zeichen = Kontroll-Zeichen = Steuerzeichen = normalerweise nicht sichtbare und keinesfalls druckbare Zeichen:

$80-$9F:	diverse Kontroll-Zeichen
$FF:	DEL-Zeichen (Rückwärts-Löschzeichen)

ferner

$00:	Endmarker für Textfiles

Von den Kontroll- oder Steuerzeichen haben zwei für DOS eine besondere Bedeutung:

1. Ctrl-D = $84 = DOS-Befehlserkennungszeichen
2. Ctrl-M = $8D= Feldbegrenzungszeichen

Ctrl-D und Ctrl-M sollten möglichst niemals Bestandteil eines Strings sein, da dies sonst zu Schwierigkeiten bei Applesoft-Programmen führt.

Zahlen = Strings auf der Diskette?

Textfiles können sowohl Zahlen wie Strings enthalten. Zahlen werden jedoch auf der Diskette genauso wie Strings gespeichert, d.h. genau in der Form, wie man Zahlen am Bildschirm sieht und nicht etwa in binär verschlüsselter Form, wie sie Applesoft speicherintern abgelegt (in sog. gepackter 5-Byte-Verschlüsselung bei Fließkommazahlen und gepackter 2-Byte-Verschlüsselung bei Ganzzahlen). Beispiel:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT ,,12345" : REM DIES IST EIN STRING!
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
```

```
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT Z
80 PRINT CHR$(4) ,,CLOSE"
```

Hier wird ,,12345" als String gespeichert und als Zahl durch INPUT Z eingelesen.

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT 12345 : REM DIES IST EINE ZAHL!
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT S$
80 PRINT CHR$(4) ,,CLOSE"
```

Hier wird 12345 als Zahl gespeichert und als String durch INPUT S$ eingelesen.

Diese beiden Beispiele machen deutlich, daß man als Strings gespeicherte Zahlen als Zahlen-Variablen einlesen kann und umgekehrt. Voraussetzung ist natürlich, daß die Strings nur Zahlzeichen enthalten, also die Ziffern von 0-9 sowie Dezimalpunkt, Plus- oder Minuszeichen sowie E (= Exponent-Zeichen).

2.6.3. Komma und Semikolon: PRINT und INPUT

An die WRITE- und READ-Befehle schließen sich üblicherweise die entsprechenden PRINT- und INPUT-Befehle an. Um mit Textfiles effektiv umgehen zu können, muß man sich zunächst einmal mit den PRINT- und INPUT-Befehlen, die eigentlich Applesoft-Befehle sind, vertraut machen.
PRINT + Zahl oder PRINT + String, z.B.
PRINT A : PRINT A$ oder
PRINT 1 : PRINT ,,AAAAA"
bewirken normalerweise die Bildschirmanzeige der Zahlen bzw. Strings. Ist ein Drucker angeschlossen (mit PR # 1), werden die PRINT-Werte gleichzeitig an den Drucker geschickt. Ist jedoch eine DOS-Datei geöffnet, dann bewirken die PRINT-Befehle das Speichern der Werte auf der Diskette.

Nach der Basic-Syntax lassen sich 3 PRINT-Arten unterscheiden:

1. PRINT + Return, z.B. PRINT A

2. PRINT + Semikolon, z.B. PRINT A;
3. PRINT + Komma, z.B. PRINT A,

Das Semikolon am Ende eines PRINT-Statements unterdrückt das Return und das Komma am Ende eines PRINT-Statements bewirkt eine Tabulierung am Bildschirm.

Der INPUT-Befehl ist normalerweise für die manuelle Tastatureingabe aus einem Applesoft-Programm heraus gedacht. Ist jedoch eine DOS-Datei geöffnet, dann bewirkt der INPUT-Statement das Einlesen eines Strings oder einer Zahl von der Diskette und die Zuweisung zu der spezifizierten Variablen. Während PRINT statt Variablen auch Konstanten als Parameter haben kann, z.B.

PRINT 1 statt PRINT A, wobei A = 1 oder
PRINT „AAA" statt PRINT A$, wobei A$ = „AAA"

muß INPUT als Parameter stets eine Variable haben, also

INPUT A oder INPUT A$

Nach der Basic-Syntax lassen sich 2 INPUT-Arten unterscheiden:

1. INPUT + Return, z.B. INPUT A : INPUT B : INPUT C
2. INPUT + Komma, z.B. INPUT A, B, C

Für den ersten Fall gibt es bei Applesoft-Programmen entweder die Möglichkeit, daß mehrere INPUT-Statements — getrennt durch Doppelpunkt — in derselben Programmzeile stehen

10 INPUT A : INPUT B : INPUT C

oder daß die INPUT-Statements auf mehrere Programmzeilen verteilt werden

10 INPUT A
20 INPUT B
30 INPUT C

Im zweiten Fall müssen die Variablen alle in einer einzigen Zeile stehen und durch Return bei der letzten Variablen abgeschlossen werden:

10 INPUT A, B, C

2.6.4. Komma bei Mehrfachfeldern

Da PRINT-Befehle mit Kommas eigentlich eine Tabulierung bewirken, die natürlich auf der Diskette nichts zu suchen hat, sollte man die READ- und WRITE-Befehle nach meinen Erfahrungen am besten gar nicht mit Kommas implementieren, da man sonst wunderliche Überraschungen erleben kann.

Falsches Beispiel 1:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT 1, 2, 3
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT A, B, C
80 PRINT CHR$(4) ,,CLOSE"
```

Kommentar: Auf der Diskette entsteht eine Datei mit der Struktur

123R

d.h. die Ziffern sind nicht durch Kommas voneinander abgetrennt, so daß insgesamt nur eine einzige Zahl 123 auf der Diskette gespeichert wird mit der Folge, daß bei Einlesen nach der Variablen A für B und C keine Werte mehr zur Verfügung stehen.

Falsches Beispiel 2:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT 1 ,,," 2 ,,," 3
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT A : INPUT B : INPUT C
80 PRINT CHR$(4) ,,CLOSE"
```

Kommentar: Auf der Diskette entsteht eine Datei mit der Struktur

1,2,3R

d.h. die Ziffern sind durch Kommas getrennt. Da jedoch der nackte INPUT-Statement (ohne folgendes Komma) ein Return und kein Komma erwartet, bewirkt Beispiel 2 ebenso wie Beispiel 1 eine DOS-Fehlermeldung.

Richtiges Beispiel 3a:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT 1 ,,," 2 ,,," 3
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT A, B, C
80 PRINT CHR$(4) ,,CLOSE"
```

Richtiges Beispiel 3b:

```
10 A = 1 : B = 2 : C = 3
20 PRINT CHR$(4) ,,OPEN TEST"
30 PRINT CHR$(4) ,,WRITE TEST"
40 PRINT A ,,," B ,,," C
50 PRINT CHR$(4) ,,CLOSE"
60 PRINT CHR$(4) ,,OPEN TEST"
70 PRINT CHR$(4) ,,READ TEST"
80 INPUT A, B, C
90 PRINT CHR$(4) ,,CLOSE"
```

Kommentar: Bei den Beispielen 3a und 3b entstehen wie bei dem Beispiel 2 auf der Diskette Files mit der Struktur

1,2,3R

Die in Anführungszeichen gesetzen Kommas bewirken, daß diese Kommas auch auf der Diskette mitgespeichert werden (im Gegensatz zum Beispiel 1, wo die Kommas nicht in Anführungszeichen gesetzt wurden). Liest man nun diese Datei mit Komma-INPUT-Statements ein, erfolgt eine korrekte Wertzuweisung zu den Variablen A, B und C. Es ist also möglich, mit Kommas zu arbeiten, doch ist das folgende Programm erheblich narrensichere, zumal dann eine einheitliche File-Struktur ,,Feld + Return" stets gewährleistet wird:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 PRINT 1 : PRINT 2 : PRINT 3
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) ,,READ TEST"
70 INPUT A : INPUT B : INPUT C
80 PRINT CHR$(4) ,,CLOSE"
```

Dieses Beispiel erscheint auf den ersten Blick umständlicher als die obigen Beispiele. Man muß jedoch berücksichtigen, das das Speichern und Einlesen von Textfiles üblicherweise nicht einer Anhäufung von Einzelvariablen darstellt, z.B.:

```
10 INPUT A1 : INPUT A2 : INPUT A3 : INPUT A4 : INPUT A5 : REM usw.
```

denn dies wäre bei einer großen Anzahl von Variablen völlig unökonomisch. Vielmehr werden Textfiles sinnvollerweise als Arrays gespeichert bzw. eingelesen, z.B.:

```
10 DIM A(1000)
20 PRINT CHR$(4) ,,OPEN DATEI"
30 PRINT CHR$(4) ,,READ DATEI"
40 FOR X = 1 TO 1000 : INPUT A(X) : NEXT X
50 PRINT CHR$(4) ,,CLOSE"
```

Auf diese Weise lassen sich mit wenigen Programmzeilen Tausende von Zahlen oder Strings einlesen.

2.6.5. Semikolon: GET und PRINT CHR$(X)

Semikolons (oder Semikola) bewirken beim PRINT-Befehl die Unterdrückung des Returns sowohl bei der Bildschirmanzeige und Druckerausgabe wie auch bei der Speicherung von Textfiles auf der Diskette. In Applesoft ist es praktisch nicht möglich und auch nicht sinnvoll, Textfiles anzulegen, die keinerlei Returns enthalten. Da vor dem CLOSE-Befehl stets ein Return stehen muß, muß ein mit Applesoft erstellter Textfile zwangsweise ein Return enthalten, z.B.

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 FOR X = 1 TO 10000 : PRINT ,,AAAAAAAAAA"; : NEXT
40 PRINT : PRINT CHR$(4) ,,CLOSE"
```

Diese TEST-Datei würde aus 10 mal 10000 = 100.000 A's + einem einzigen Return am File-Ende bestehen. Eine solche fast Return-lose Datei könnte man jedoch mit INPUT-Statements nicht mehr einlesen, da der INPUT-String maximal 239 Zeichen lang sein darf. Mit anderen Worten: Applesoft-Programmierer können grundsätzlich nur Textfiles bewältigen, die in Felder eingeteilt sind, wobei die praktikable Feldlänge in der Regel ca. 230 Zeichen inkl. Return nicht überschreiten sollte.
Ein String als solcher darf bis zu 255 Zeichen lang sein, doch kann ein solcher String nicht mit INPUT eingegeben werden wegen der Obergrenze von 239 Zeichen. Die Definition eines Strings in einer Programmzeile ist sogar auf ca. 230 Zeichen begrenzt, je nachdem, ob die Zeilennummer ein- oder mehrstellig ist:

```
100A$ = ,,12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890"
```

Ein einzelner String mit bis 255 Zeichen kann also nur durch Verkettung entstehen:

```
10 A$ = B$ + C$
```

Dateien, die keine (oder fast keine) Returns enthalten, lassen sich von einem Applesoft-Programm mit GET-Befehlen einlesen. Bei Dateien der obigen Größenordnung ist jedoch der GET-Statement völlig undiskutabel, das es eine schiere Ewigkeit dauern würde, bis diese Datei eingelesen wäre. Probieren Sie es selber einmal aus!

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,READ TEST"
30 FOR X = 1 TO 10000
40 FOR Y = 1 TO 10
50 GET X$
60 NEXT Y,Z
70 PRINT : PRINT CHR$(4) ,,CLOSE"
```

Wie lange hat es gedauert?

Übrigens gibt es kein exaktes Gegenstück zum GET-Befehl, denn während der Get-Befehl tatsächlich nur zum Einlesen von je einem einzelnen Zeichen von der Diskette verwendet werden kann, eignet sich der PRINT-Befehl sowohl zum Speichern einzel-

ner Zeichen als auch längerer Zeichenketten (= Strings). Bisweilen ist es erforderlich, daß eine Folge einzelner Zeichen als ASCII-Code in Form von

PRINT CHR$(X);

auf die Diskette gesendet werden müssen, z.B. wenn Binärfiles in Textfiles umgewandelt werden müssen. Ein Beispiel ist hierfür die Umwandlung von Applewriter 1.1-Binärfiles in normale Textfiles, was an dem nachfolgenden Programm demonstriert werden soll:

2.6.5.1. Applewriter 1.1 Binärfile-Konverter

```
100 REM = = = APPLEWRITER 1.1: BINÄR- IN TEXT-FILE = = =
110 HOME : HTAB 14: VTAB 3: INVERSE : PRINT ,,APPLEWRITER 1.1"
120 HTAB 1: VTAB 7: PRINT ,,UMWANDLUNG VON B-FILES IN T-FILES":
    NORMAL
130 PRINT : PRINT : PRINT ,,START J/N ";
140 GET X$: IF X$ = ,,N" THEN HOME : END
150 IF X$ < > ,,J" THEN 140
160 PRINT : PRINT : PRINT ,,LAUFWERK-NR. (1 ODER 2): ";
170 GET X$: IF X$ < > ,,1" AND X$ < > ,,2" GOTO 170
180 D = VAL (X$)
190 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D"D
200 INPUT ,,BSAVE-FILE: TEXT.";Y$
210 PRINT ,,VERTIPPT J/N ";: GET X$: IF X$ < > ,,N" THEN 190
220 HOME : PRINT : PRINT CHR$ (4),,BLOAD TEXT."Y$
230 INVERSE : PRINT ,,ABSPEICHERN DES TEXT-FILES": NORMAL
240 PRINT : PRINT : PRINT ,,LAUFWERK-NR. (1 ODER 2): ";
250 GET X$: IF X$ < > ,,1" AND X$ < > ,,2" THEN 250
260 D = VAL (X$)
270 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D"D
280 INPUT ,,TEXT-FILE: ";Y$
290 PRINT ,,VERTIPPT J/N ";: GET X$: IF X$ < > ,,N" GOTO 270
300 HOME : HTAB 12: INVERSE : PRINT ,,BITTE WARTEN": NORMAL :
    POKE 34,2: HOME
310 PRINT CHR$ (4),,MON O"
320 PRINT CHR$ (4),,OPEN"Y$: PRINT CHR$ (4),,DELETE"Y$
330 PRINT CHR$ (4),,OPEN"Y$: PRINT CHR$ (4),,WRITE"Y$
340 AD = 6401:Z1 = 32:Z2 = 64:Z3 = 96:Z4 = 128:Z5 = 192:Z6 = 224
```

```
350 A = PEEK (AD):AD = AD + 1
360 IF A = Z3 THEN 420
370 IF A > = Z6 THEN A = A − Z2
380 IF A > = Z5 THEN A = A + Z1
390 IF A < Z1 THEN A = A + Z5
400 IF A < Z2 THEN A = A + Z4
410 PRINT CHR$ (A);: GOTO 350
420 PRINT : PRINT CHR$ (4),,CLOSE": PRINT CHR$ (4),,NOMON O":
    PRINT CHR$ (4),,CATALOG"
430 POKE 34,0
440 INVERSE : PRINT ,,ERNEUT J/N";: NORMAL : PRINT ,, ";
450 GET X$: IF X$ = ,,J" GOTO 110
460 IF X$ < > ,,N" GOTO 450
```

Der PRINT CHR$(A)-Befehl steht bei diesem Programm in Zeile 410. Auch dieses Programm ist ein Beispiel dafür, wie erschreckend lange es dauert, wenn der Zugriff auf einen Textfile auf der Einzelzeichen-Ebene erfolgt.

2.6.6. Komma, Doppelpunkt und Anführungszeichen bei Strings

Da Komma, Doppelpunkt und Anführungszeichen syntaktische Funktionen bei dem INPUT-Befehl übernehmen (technisch am besten beschrieben in dem alten ,,Basic Programming Reference Manual", S. 66), kann ein auf der Diskette gespeicherter String diese Zeichen nicht alle gleichzeitig enthalten. Es gibt nur zwei Möglichkeiten:

1. Entweder man verzichtet auf die Anführungszeichen (ASCII-Code 34) und kann dann Komma und Doppelpunkt als Bestandteile von Strings auf Textfiles speichern,
2. oder man verzichtet auf Komma und Doppelpunkt und kann dann Anführungszeichen als Bestandteile von Strings auf Textfiles speichern.

Um es klarzustellen: Bei Assembler-Programmen gelten diese Beschränkungen nicht, wohl aber bei reinen Applesoft-Programmen.

Da Komma und Doppelpunkt wichtiger als die doppelten Anführungszeichen (= Gänsefüßchen) sind, zumal ja auch noch die einfachen An- und Abführungszeichen existieren (ASCII-Codes 96 und 39; letzerer zugleich Apostroph), kann man zu dem folgenden, wenig bekannten Trick greifen:

1. Alle auf den WRITE-Befehl folgenden PRINT-Statements werden mit CHR$(34) eingeleitet:

```
10 A$.=,,aaa,aaa:aaa"
20 PRINT CHR$(4) ,,OPEN TEST" : PRINT CHR$(4) ,,WRITE TEST"
30 PRINT CHR$(34) A$
40 PRINT CHR$(4) ,,CLOSE"
```

Ein Semikolon zwischen CHR$(34) und A$, z.B.
30 PRINT CHR$(34) ; A$
dient nur der besseren Lesbarkeit und ist funktional entbehrlich.

2. Bei allen auf den READ-Befehl folgenden INPUT-Statements ist keine besondere Syntax zu beachten, doch muß man wissen, daß bei

```
10 PRINT CHR$(4) ,,OPEN TEST" : PRINT CHR$(4) ,,READ TEST"
20 INPUT A$
30 PRINT CHR$(4) ,,CLOSE"
```

der Variablen A$ nur der reine String aaa,aaa:aaa und nicht etwa Anführungszeichen + aaa,aaa:aaa zugewiesen wird.

Der einzige Nachteil dieses Verfahrens ist der, daß nunmehr jedes Feld auf der Diskette 1 Byte mehr Speicherraum (wegen des Anführungszeichens) beansprucht. Bei langen Strings fällt dies jedoch kaum ins Gewicht. Das nachfolgende Programmbeispiel demonstriert das CHR$(34)-Verfahren:

```
100 REM = = = KOMMA-KOLON-TEST = = =
110 HOME : INVERSE : PRINT ,,KOMMA-KOLON-READ-WRITE-TEST":
    NORMAL
120 PRINT : PRINT : PRINT ,,STARTEN J/N ";: GET X$: IF X$ = ,,N"
    THEN END
130 IF X$ < > ,,J" THEN 110
140 DIM A$(3)
150 X$ = ,,STIEHL,ULRICH:HEIDELBERG"
160 FOR X = 1 TO 3:A$(X) = X$: NEXT
170 HOME : INVERSE : PRINT ,,SO HEISST DER STRING:": NORMAL
180 PRINT : PRINT X$: PRINT : INVERSE : PRINT ,,3 X SPEICHERN:":
    NORMAL : PRINT
190 PRINT CHR$ (4),,MONIO"
```

```
200 PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$ (4),,DELETE
    STRINGTEST": PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$
    (4),,WRITE STRINGTEST"
210 FOR X = 1 TO 3
220 REM = = = ENTSCHEIDENDE ZEILE
230 REM = = = MIT CHR$(34) VOR A$
240 PRINT CHR$ (34) A$(X)
250 NEXT
260 PRINT CHR$ (4),,CLOSE"
270 PRINT : INVERSE : PRINT ,,3 X EINLESEN:": NORMAL : PRINT
280 PRINT CHR$ (4),,OPEN STRINGTEST": PRINT CHR$ (4),,READ
    STRINGTEST"
290 FOR X = 1 TO 3
300 INPUT A$(X)
310 NEXT
320 PRINT CHR$ (4),,CLOSE"
330 PRINT CHR$ (4),,NOMONIO"
340 PRINT : INVERSE : PRINT ,,DAS WURDE EINGELESEN:": NORMAL :
    PRINT
350 FOR X = 1 TO 3: PRINT A$(X): NEXT
```

2.6.7. OPEN-Files

Die Programmierung wird komplizierter, wenn gleichzeitig mehrere Dateien geöffnet sind oder wenn eine Datei nicht geschlossen werden kann, weil laufend oder in Intervallen auf die Datei zugegriffen wird.

Der uns bereits bekannte einfachste Fall liegt dann vor, wenn die PRINT- oder INPUT-Befehle von WRITE-CLOSE- bzw. READ-CLOSE-Befehlen unmittelbar eingeschlossen werden. Was passiert jedoch, wenn zwischendurch Tastatureingaben erwartet oder Hinweise am Bildschirm angezeigt werden sollen? In diesem Fall muß der READ- oder WRITE-Befehl vorübergehend inaktiviert werden, damit nicht etwa der Bildschirm-Hinweis plötzlich auf der Diskette gespeichert wird. Die vorübergehende Inaktivierung von READ-WRITE-Befehlen ist denkbar einfach durch einen reinen PRINT CHR$(4)-Befehl zu realisieren. (Übrigens führt jeder ANDERE DOS-Befehl, z.B. PRINT CHR$(4) ,,CATALOG", wegen des Ctrl-D zu einer Inaktivierung des momentanen Read- oder Write-Zustandes.)

Beispiel 1:

```
10 PRINT CHR$(4) ,,OPEN DATEI"
20 PRINT CHR$(4) : REM INAKTIVERT DOS
30 PRINT ,,STRING EINGEBEN!"
40 INPUT X$
50 PRINT CHR$(4) ,,WRITE DATEI"
60 PRINT X$ : GOTO 20
```

Hier würde eine fehlende Zeile 20 zur Folge haben, daß neben der Variablen X$ auf der Diskette ungewollt jeweils der String ,,STRING EINGEBEN!" mit abgespeichert werden würde.

Beispiel 2:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,WRITE TEST"
30 FOR X = 1 TO 100 : PRINT X : NEXT
40 PRINT CHR$(4) ,,CLOSE"
50 PRINT CHR$(4) ,,OPEN TEST"
60 PRINT CHR$(4) : REM INAKTIVIERT DOS
70 PRINT ,,ZAHL EINLESEN J/N ": INPUT X$
80 IF X$ = ,,N" THEN PRINT CHR$(4) ,,CLOSE" : END
90 PRINT CHR$(4) ,,READ TEST" : INPUT Z : PRINT Z : GOTO 60
```

Hier würde eine fehlende Zeile 60 zur Folge haben, daß bei Zeile 70 mit INPUT X$ die nächste Zahl automatisch von der Diskette käme, ohne daß man eine Chance hätte, durch Tastatureingabe von ,,N" das Programm abzubrechen.

Die vorübergehende DOS-Inaktivierung, die übrigens nicht mit dem DOS-,,Abhängen" durch PR#0 : IN#0 zu verwechseln ist, hat übrigens den Vorteil, daß eventuelle DOS- oder Applesoft-Fehlermeldungen nicht im Falle des WRITE-Aktivzustandes aus Versehen auf der Diskette gespeichert werden.

Wenn mehrere Dateien gleichzeitig geöffnet sind, kommen oft noch die Parameter Slot, Drive und Volume zum Tragen. Beispiel:

```
10 PRINT ,,A-DATEI:" : PRINT
20 INPUT ,,DATEINAME, SLOT, DRIVE, VOLUME: "; DA$, SA, DA, VA
30 PRINT ,,B-DATEI:" : PRINT
```

```
40 INPUT ,,DATEINAME, SLOT, DRIVE, VOLUME: "; DB$, SB, DB, VB
50 PRINT CHR$(4) ,,OPEN"DA$ ,,,S"SA ,,,D"DA ,,,V"VA
60 READ CHR$(4) ,,READ"DA$
70 PRINT CHR$(4) ,,OPEN"DB$ ,,,S"SB ,,,D"DB ,,,V"VB
80 READ CHR$(4) ,,READ"DB$
```

Ein im meinen Augen lästiger Fehler des DOS-Betriebssystems ist der, daß die READ- und WRITE-Befehle im Gegensatz zum OPEN-Befehl keine Parameter zulassen. Deshalb ist es ohne Poke-Befehle nicht möglich, auf mehrere, sich in verschiedenen Drives und/oder Slots befindliche Dateien gleichzeitig zuzugreifen, ohne die jeweilige Datei vorher zu schließen.

Da Apple-Besitzer in der Regel nur über 2 Laufwerke verfügen, stellt sich das Problem des Slot-Wechsels selten, und Volume-Nummern werden ohnehin kaum benutzt. Dagegen ist die Notwendigkeit des Drive-Wechsels offenkundig. Angenommen, man wollte zwei in sich sortierte Textfiles, die so groß sind, daß sie nicht komplett in den RAM-Speicher passen, von Drive 1 nach Drive 2 mischen (= Merge- oder Mischvorgang = Vereinigung zweier in sich sortierter Teildateien zu einer neuen ebenfalls in sich sortierten großen Gesamtdatei). Zu diesem Zweck müssen vorübergehend alle drei Dateien offen sein.

Drive-Wechsel:

Mit POKE -21912,1 kann man Drive 1 als aktives Drive deklarieren.
Mit POKE -21912,2 kann man Drive 2 als aktives Drive deklarieren.

Slot-Wechsel:

Mit POKE -21910,4 kann man Slot 4 als aktiven Slot deklarieren.
Mit POKE -21910,6 kann man Slot 6 als aktiven Slot deklarieren, usw.

Diese Pokes gelten für 48K-DOS, also nicht für den Fall, daß sich DOS in der Language Card befindet.

2.6.7.1. „Schein-Mischen" (Demo-Programm)

```
100  PRINT CHR$(4) ,,OPEN A1, D1"
110  PRINT CHR$(4) ,,WRITE A1"
120  FOR X = 1 TO 1000
```

```
130 PRINT ,,AAAAAAAAAAAAAAA" : NEXT
140 PRINT CHR$(4) ,,CLOSE"
150 PRINT CHR$(4) ,,OPEN A2"
160 PRINT CHR$(4) ,,WRITE A2"
170 FOR X = 1 TO 1000
180 PRINT ,,AAAAAAAAAAAAAAA" : NEXT
190 PRINT CHR$(4) ,,CLOSE"
200 REM SCHEIN-MERGEN
210 DIM A1$ (100), A2$ (100)
220 PRINT CHR$(4) ,,OPEN A1" *
230 PRINT CHR$(4) ,,OPEN A2" *
240 POKE -21912,2 : REM DRIVE 2
250 PRINT CHR$(4) ,,OPEN M" *
260 FOR X = 1 TO 10
270 POKE -21912,1 : REM DRIVE 1
280 PRINT CHR$(4) ,,READ A1"
290 FOR Y = 1 TO 100 : INPUT A1$(X) : NEXT
300 PRINT CHR$(4) ,,READ A2"
310 FOR Y = 1 TO 100 : INPUT A2$(X) : NEXT
320 POKE -21912,2 : PRINT CHR$(4) ,,WRITE M": REM DRIVE 2
330 FOR Y = 1 TO 100 : PRINT A1$(X) : PRINT A2$(X) : NEXT
340 NEXT X
350 PRINT CHR$(4) ,,CLOSE"
```

Ohne die angegebenen Poke-Befehle wäre es bei den geöffneten Dateien A1, A2 und M nicht möglich gewesen, den Drive-Wechsel aus dem Programm heraus vorzunehmen.

2.6.7.2. „Echtes Mischen" (Registerprogramm)

Das nachfolgende komplizierte Programm ist ein Teil-Modul zu meinem Registerprogramm, das vor allem für Register am Ende von Büchern (Stichwort + Seitenzahl) Verwendung findet. Die Dateistruktur ist folgende:

1. Feld: Anzahl der Stringpaare (als 6stelliger String)
2. Feld: CHR$(34) + 1. Stichwort
3. Feld: CHR$(34) + 1. Seitenzahl
4. Feld: CHR$(34) + 2. Stichwort
5. Feld: CHR$(34) + 2. Seitenzahl

* Wie ein Leser richtig festgestellt hat, sind die Pokes entbehrlich, wenn die OPEN-Befehle mit kompletten Slot-Drive-Volume-Parametern erfolgen.

usw. bis zu den 2 letzten Feldern, die ,,ULI", ,,ULI" als programmtechnische Endmarker haben.

Beispiel:

```
000004
,,Computer
,,10
,,DOS 3.3
,,30
,,Apple IIe
,,20
,,formatieren
,,70
,,ULI
,,ULI
```

Es sind also sowohl die Stichwörter als auch die Seitenzahlen als Strings gespeichert, da das Programmpaket auch für andere Register, z.B. Glossare, verwendet werden kann. Es werden automatisch bis zu 20 in sich bereits sortierte, beliebig große Dateien paarweise von Drive 1 eingelesen, gemischt und als neue Misch-Datei auf Drive 2 abgespeichert. Das paarweise Mischen muß solange vollzogen werden, bis nur noch eine einzige ,,Monster"-Enddatei übrigbleibt, die übrigens mehr als 300.000 Zeichen Umfang haben kann. Da das Mischen ein langwieriger Prozeß ist, ist das Applesoft-Programm mit dem TASC compiliert (mit Origin $0806). Außerdem wird Diversi-DOS benutzt, das in die Language Card geschoben wird. Aber auch dies reicht noch nicht aus, da Sortieren und Mischen nach DUDEN erfolgen soll, dessen Alphabetisierungsrichtlinien sich bekanntlich nicht mit der ASCII-Code-Reihenfolge decken. Deshalb erfolgt die Umcodierung durch ein Assemblerprogramm, das die Strings auch nicht mehr mit dem INPUT-Befehl, sondern mit dem Monitor-GETLN-Befehl ($FD6A) einliest, umcodiert und dann in der ,,Schein-String-Zeile" 150 ablegt, so daß das compilierte Applesoft-Programm diesen String übernehmen kann. Auf diese Weise — Diversi-DOS, DOS in der LC, Compilierung, Assemblerroutinen — ist das Registermischprogramm etwa 10-20 mal schneller, als ein entsprechendes ,,normales" Applesoft-Programm unter DOS 3.3 wäre. Was allein 10mal schneller bedeutet, wird einem klar, wenn man statt 1 Minute z.B. 10 Minuten am Bildschirm wartet.

Wenn DOS oder Diversi-DOS in der Language Card liegt, ist das Umschalten von Drive 1 nach Drive 2 bei geöffneten Textfiles besonders kompliziert und kann wegen

des erforderlichen „Bank-Selecting" nur durch ein kleines Assemblerprogramm realisiert werden. Das Umschalten geschieht in dem nachstehenden Programm durch die Befehle CALL 826 (= Drive 1) und CALL 830 (= Drive 2). Ferner kommt in dem Programm CALL 849, das das Sortierwort unmodifiziert mit GETLN einliest, sowie CALL 875 vor, das die Umcodierung vornimmt.

```
100 REM = = = REGISTERMISCHER = = =
110 REM !INTEGER*
120 REM !DEFCOMMONA,D
130 REM !DEFCOMMOND$(20),D$
140 GOTO 190
150 M$ = ,,": M$ = LEFT$ (,,?----------
    ----------",MM): RETURN
160 CALL 849:MM = PEEK (255): GOSUB 150:M0$ = M$
170 CALL 875:MM = PEEK (255): GOSUB 150:C0$ = M$
180 INPUT N0$: RETURN
190 POKE 846,10: POKE 847,24: REM $180A:?---
200 ONERR GOTO 220
210 GOTO 280
220 PRINT CHR$ (4): PRINT CHR$ (4),,CLOSE": IF PEEK (222) = 9 THEN
    PRINT CHR$ (4),,DELETE"A0$
230 TEXT : HOME : PRINT ,,FEHLER-NR. " PEEK (222)
240 PRINT CHR$ (7)
250 PRINT ,,W = WEITER ";
260 GET D$: IF D$ < > ,,W" THEN 260
270 REM MENUE
280 HOME : INVERSE : PRINT ,, REGISTERMISCHER ": NORMAL
290 PRINT : PRINT ,,(MAX.10 SORTIERTE DATEIPAARE MISCHBAR)"
300 PRINT : PRINT ,,(AUSGANGSDATEIEN,D1 -> ZIELDATEIEN,D2)"
310 PRINT : PRINT ,,1 MISCHEN VON DATEIPAAREN": PRINT : PRINT
    ,,5 UEBERTRAGEN EINER RESTDATEI": PRINT : PRINT ,,0
    HAUPTMENUE": PRINT
320 INVERSE : PRINT ,,VON ULRICH STIEHL - VERSION V. 01.02.84":
    NORMAL : PRINT
330 GET D$: ON D$ = ,,1" GOTO 350: ON D$ = ,,5" GOTO 940: ON D$ =
    ,,0" GOTO 540: GOTO 330
340 REM DATEIENEINGABE
350 CLEAR : DIM D$(20)
360 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D1": INVERSE :
    PRINT ,,RETURN ALLEIN = EINGABEENDE": NORMAL
```

```
370 D = 0
380 D = D + 1: INPUT ,,A-DATEI:";D$(D)
390 D = D + 1: INPUT ,,B-DATEI:";D$(D)
400 IF D$(D - 1) = ,," AND D$(D) = ,," AND D = 2 THEN 280
410 IF D$(D - 1) = ,," AND D$(D) = ,," THEN D = D - 2: GOTO 440
420 IF D$(D - 1) = ,," OR D$(D) = ,," THEN 360
430 IF D < 19 THEN 380
440 PRINT ,,RICHTIG J/N ";
450 GET D$: ON D$ = ,,N" GOTO 350: IF D$ < > ,,J" GOTO 450
460 GOSUB 530: PRINT : FOR X = 1 TO D: PRINT CHR$
    (4),,UNLOCK"D$(X),,,D1": NEXT
470 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D2": INPUT ,,ZIEL-
    DATEINAME:";D$: ON D$ = ,," GOTO 470: PRINT ,,RICHTIG J/N ";
480 GET X$: ON X$ = ,,N" GOTO 470: IF X$ < > ,,J" GOTO 480
490 REM MISCHSTART
500 A = - 1: HOME
510 A = A + 2: IF A + 1 < = D THEN 570
520 HOME : PRINT ,,MISCHEN FERTIG !": GOTO 240
530 HOME : HTAB 14: VTAB 10: INVERSE : PRINT ,,BITTE WARTEN":
    NORMAL : RETURN
540 HOME : INVERSE : PRINT ,,PROGRAMMDISKETTE IN DRIVE 1":
    NORMAL : PRINT CHR$ (7): INPUT ,,HAUPTMENUE EINLESEN
    JA/N ";D$: IF D$ < > ,,JA" THEN 280
550 GOSUB 530: PRINT : PRINT CHR$ (4),,BRUN
    REGISTERSTARTER.OBJ,D1"
560 REM = = = MISCHEN = = =
570 CLEAR :L = 150: DIM T$(150),I$(150)
580 GOSUB 530: HTAB 1: VTAB 1: PRINT D$(A): PRINT D$(A + 1)
590 REM ERSTER STRING
600 A1$ = D$(A):A2$ = D$(A + 1)
610 A0$ = D$ + ,,.M" + STR$ ((A + 1) / 2)
620 PRINT : PRINT CHR$ (4),,OPEN"A1$,,,D1": PRINT CHR$
    (4),,READ"A1$: INPUT X$:X = VAL (X$)
630 PRINT CHR$ (4),,OPEN"A2$: PRINT CHR$ (4),,READ"A2$: INPUT
    Y$:Y = VAL (Y$)
640 CALL 830: REM LAUFWERK 2
650 PRINT CHR$ (4),,OPEN"A0$: PRINT CHR$ (4),,WRITE"A0$:X = X
    + Y:X$ = STR$ (X):Y$ = ,,000000":X$ = LEFT$ (Y$,6 - LEN (X$)) +
    X$: PRINT X$
660 CALL 826: REM LAUFWERK 1
```

```
670 PRINT CHR$ (4),,READ"A1$: GOSUB 160:M1$ = M0$:C1$ = C0$:N1$
    = N0$
680 PRINT CHR$ (4),,READ"A2$: GOSUB 160:M2$ = M0$:C2$ = C0$:N2$
    = N0$
690 B = 1
700 IF C1$ > C2$ GOTO 780
710 REM = = = = C1$<=C2$ = = = =
720 T$(B) = M1$:I$(B) = N1$
730 B = B + 1: IF B > L THEN GOSUB 900
740 PRINT CHR$ (4),,READ"A1$: GOSUB 160:M1$ = M0$:C1$ = C0$:N1$
    = N0$
750 IF M1$ = ,,ULI" AND N1$ = ,,ULI" THEN PRINT CHR$
    (4),,CLOSE"A1$:A$ = A2$:T$(B) = M2$:I$(B) = N2$: GOTO 840
760 GOTO 700
770 REM = = = = C1$>C2$ = = = =
780 T$(B) = M2$:I$(B) = N2$
790 B = B + 1: IF B > L THEN GOSUB 900
800 PRINT CHR$ (4),,READ"A2$: GOSUB 160:M2$ = M0$:C2$ = C0$:N2$
    = N0$
810 IF M2$ = ,,ULI" AND N2$ = ,,ULI" THEN PRINT CHR$
    (4),,CLOSE"A2$:A$ = A1$:T$(B) = M1$:I$(B) = N1$: GOTO 840
820 GOTO 700
830 REM A$=RESTDATEI
840 B = B + 1: IF B > L THEN GOSUB 900
850 PRINT CHR$ (4),,READ"A$: INPUT T$(B): INPUT I$(B)
860 IF T$(B) < > ,,ULI" AND I$(B) < > ,,ULI" GOTO 840
870 L = B: GOSUB 900
880 PRINT CHR$ (4),,CLOSE": PRINT CHR$ (4),,DELETE"A1$,,D1":
    PRINT CHR$ (4),,DELETE"A2$: GOTO 510
890 REM SPEICHERN EINES L-BLOCKS
900 CALL 830
910 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO L: PRINT CHR$
    (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
920 FOR X = 1 TO L:T$(X) = ,,":I$(X) = ,,": NEXT : CALL 826:B = 1:
    RETURN
930 REM ÜBERTRAGEN
940 CLEAR :L = 150: DIM T$(150),I$(150): HOME : INVERSE : PRINT
    ,,UEBERTRAGEN EINER DATEI ": PRINT ,,VON DRIVE 1 NACH
    DRIVE 2": NORMAL
950 PRINT : PRINT ,,1 = ZURUECK": PRINT : PRINT ,,0 = UEBER-
```

```
    TRAGEN": PRINT
 960 GET D$: ON D$ = ,,1" GOTO 280: IF D$ < > ,,0" GOTO 960
 970 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D1": INPUT ,,ALTER
     DATEINAME: ";X$: ON X$ = ,," GOTO 940: PRINT
     ,,RICHTIG J/N ";
 980 GET Y$: ON Y$ = ,,N" GOTO 970: IF Y$ < > ,,J" GOTO 980
 990 HOME : PRINT : PRINT CHR$ (4),,CATALOG,D2": INPUT ,,NEUER
     DATEINAME: ";A0$: ON A0$ = X$ GOTO 940: PRINT
     ,,RICHTIG J/N ";
1000 GET Y$: ON Y$ = ,,N" GOTO 990: IF Y$ < > ,,J" GOTO 1000
1010 GOSUB 530: PRINT : PRINT CHR$ (4),,OPEN"X$,,,D1": PRINT CHR$
     (4),,READ"X$: INPUT Y$:B = VAL (Y$)
1020 PRINT CHR$ (4),,OPEN"A0$,,,D2": PRINT CHR$ (4),,DELETE"A0$:
     PRINT CHR$ (4),,OPEN"A0$: PRINT CHR$ (4),,WRITE"A0$:
     PRINT Y$
1030 IF B > L THEN 1110
1040 REM B< =L;B+1=,,ULI"
1050 CALL 826
1060 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO B + 1: INPUT T$(X):
     INPUT I$(X): NEXT
1070 CALL 830
1080 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO B + 1: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT : PRINT CHR$ (4),,CLOSE"
1090 PRINT CHR$ (4),,DELETE"X$,,,D1": GOTO 280
1100 REM B>L;D=REST
1110 A = INT (B / L):D = B + 1 − A · L
1120 FOR Z = 1 TO A
1130 CALL 826
1140 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO L: INPUT T$(X): INPUT
     I$(X): NEXT
1150 CALL 830
1160 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO L: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
1170 NEXT Z
1180 CALL 826
1190 PRINT CHR$ (4),,READ"X$: FOR X = 1 TO D: INPUT T$(X): INPUT
     I$(X): NEXT
1200 CALL 830
1210 PRINT CHR$ (4),,WRITE"A0$: FOR X = 1 TO D: PRINT CHR$
     (34);T$(X): PRINT CHR$ (34);I$(X): NEXT
```

```
1220 PRINT CHR$ (4),,CLOSE": PRINT CHR$ (4),,DELETE"X$,,,D1":
     GOTO 280
```

```
:ASM
                        1               ORG  826
                        2   *
                        3   ********************************
                        4   *
                        5   HC08B       EQU  $C08B
                        6   HC081       EQU  $C081
                        7   HC083       EQU  $C083
                        8   HEA68       EQU  $EA68
                        9   HC082       EQU  $C082
                       10   *
                       11   *
                       12   ********************************
                       13   *
033A: A9 01             14   DRIVE1      LDA  #$01       ;826
033C: D0 02             15               BNE  DRIVE
033E: A9 02             16   DRIVE2      LDA  #$02       ;830
0340: AE 83 C0          17   DRIVE       LDX  HC083
0343: AE 83 C0          18               LDX  HC083
0346: 8D 68 EA          19               STA  HEA68
0349: AD 82 C0          20               LDA  HC082
034C: 60                21               RTS
                       22   *
                       23   ********************************
                       24   *
                       25   STRLEN      EQU  255        ;$FF
                       26   PUFFER      EQU  $201       ;NACH "
                       27   GETLN       EQU  $FD6A
                       28   *
034D: 9D FF FF          29   POKER       STA  $FFFF,X    ;846/847
0350: 60                30               RTS
                       31   *
                       32   * DARF KEIN NULLSTRING SEIN,
                       33   * D.H. NUR RETURN!!!
                       34   *
0351: 20 6A FD          35   GETLN1      JSR  GETLN      ;849
0354: A2 00             36               LDX  #0
0356: BD 01 02          37   MOVESTR1    LDA  PUFFER,X
0359: 29 7F             38               AND  #$7F
035B: 9D 01 02          39               STA  PUFFER,X
035E: C9 0D             40               CMP  #$0D       ;RETURN
0360: F0 06             41               BEQ  MOVEEXIT
0362: 20 4D 03          42               JSR  POKER
0365: E8                43               INX
0366: D0 EE             44               BNE  MOVESTR1
0368: 86 FF             45   MOVEEXIT    STX  STRLEN
036A: 60                46               RTS
                       47   *
                       48   ********************************
```

```
                        49   *
036B: A2 00             50   GETLN2   LDX  #0          ;875
036D: A0 00             51            LDY  #0
                        52   *
036F: B9 01 02          53   MOVESTR2 LDA  PUFFER,Y
0372: C9 0D             54            CMP  #$0D        ;RETURN
0374: F0 4C             55            BEQ  EXIT1
                        56   *
0376: C9 7E             57            CMP  #$7E        ;ß
0378: F0 2E             58            BEQ  ESZETT
037A: C9 7D             59            CMP  #$7D        ;ü
037C: F0 2E             60            BEQ  UBUCHST
037E: C9 7C             61            CMP  #$7C        ;ö
0380: F0 2E             62            BEQ  OBUCHST
0382: C9 7B             63            CMP  #$7B        ,ä
0384: F0 2E             64            BEQ  ABUCHST
                        65   *
0386: C9 5D             66            CMP  #$5D        ;Ü
0388: F0 22             67            BEQ  UBUCHST
038A: C9 5C             68            CMP  #$5C        ,ö
038C: F0 22             69            BEQ  OBUCHST
038E: C9 5B             70            CMP  #$5B        ;Ä
0390: F0 22             71            BEQ  ABUCHST
                        72   *
0392: C9 61             73            CMP  #$61        ;a-z
0394: B0 22             74            BCS  GRBUCHST
0396: C9 5E             75            CMP  #$5E        ;^_`
0398: B0 25             76            BCS  NICHTS
039A: C9 41             77            CMP  #$41        ;A-Z
039C: B0 1D             78            BCS  MOVESTR3
                        79   *
039E: C9 30             80            CMP  #'0'
03A0: 90 1D             81            BCC  NICHTS      ;<0
03A2: C9 3A             82            CMP  #':'
03A4: 90 15             83            BCC  MOVESTR3    ;ZIFFER
03A6: B0 17             84            BCS  NICHTS
                        85   *
03A8: A9 53             86   ESZETT   LDA  #'S'        ;ß>S
03AA: D0 0F             87            BNE  MOVESTR3
03AC: A9 55             88   UBUCHST  LDA  #'U'        ;ä+Ä>A
03AE: D0 0B             89            BNE  MOVESTR3
03B0: A9 4F             90   OBUCHST  LDA  #'O'        ;ö+ö>O
03B2: D0 07             91            BNE  MOVESTR3
03B4: A9 41             92   ABUCHST  LDA  #'A'        ;ä+Ä>A
03B6: D0 03             93            BNE  MOVESTR3
03B8: 38                94   GRBUCHST SEC              ;KL>GR
03B9: E9 20             95            SBC  #$20
                        96   *
03BB: 20 4D 03          97   MOVESTR3 JSR  POKER
03BE: E8                98            INX              ;LENGTH
03BF: C8                99   NICHTS   INY              ;PUFFCNT
03C0: D0 AD            100            BNE  MOVESTR2
03C2: E0 00            101   EXIT1    CPX  #0
03C4: D0 06            102            BNE  EXIT2
```

```
03C6: A9 2F      103          LDA  #'/'        ;LEER
03C8: 20 4D 03   104          JSR  POKER
03CB: E8         105          INX
03CC: 86 FF      106  EXIT2   STX  STRLEN
03CE: 60         107          RTS
```

Näheres über die GETLN-Routine steht im Teil II, Kapitel 4. Insbesondere beachte man, daß GETLN im Gegensatz zu INPUT das Anführungszeichen mit einliest.

2.6.8. Sequentielle und Random-Files

Sequentielle Files sind Textfiles mit variablen Feldlängen und indirektem Zugriff, Random-Files sind Textfiles mit konstanten Feldlängen und direktem Zugriff. (Random bzw. genauer Random Access bedeutet wahlfreier = direkter Zugriff auf den Diskettensektor.)

Die bisher behandelten Beispiele waren — auch wenn dies nicht ausdrücklich erwähnt wurde — sogenannte sequentielle Textfiles. Ein typischer sequentieller Textfile wie z.B. bei dem oben beschriebenen Registerprogramm setzt sich aus einer beliebigen Anzahl unterschiedlich — die Betonung liegt auf unterschiedlich! — langer Felder zusammen, die wie eine Sequenz (= Folge) aneinandergereiht und lediglich durch Returns getrennt sind. DOS hängt an das Ende eines Textfiles automatisch ein Ctrl-0 als sozusagen DOS-internen Endmarker. Ein sequentielle Dateien verarbeitendes Applesoft-Programm sollte darüber hinaus eigene „Buchführung" machen, damit die „END OF FILE"-Fehlermeldung vermieden wird. Bei dem obigen Registerprogramm gibt es

1. als ersten String der Datei einen Anfangsmarker als normierte, 6stellige, rechtsbündig ausgeschlossene Zahl, die die Anzahl der Doppelstrings (Stichwort + Seitenzahl) beinhaltet, sowie
2. als letzte 2 Strings der Datei zwei Endmarker („ULI", „ULI"), damit man niemals beim Ctrl-0 ankommt.

Warum eine normierte 6stellige Zahl als String, mag man sich fragen. Da sequentielle Dateien Felder mit unterschiedlicher Länge haben und Zahlen selbst eine unterschiedliche Anzahl von Speicherstellen auf der Diskette einnehmen können (1, 10, 100, 1000 usw.), wäre es bei einer sequentiellen Datei nachträglich nicht mehr mög-

lich, den Anfangsmarker korrekt durch einen neuen Zahlenwert (z.B. bei gemischten Dateien) zu überschreiben.

Sequentielle Dateien sind Textfiles mit indirektem Zugriff analog zu Dateien, die auf Datenkassetten, Lochstreifen usw. gespeichert sind. Wenn man z.B. das 150. und nur dieses 150. Feld einer sequentiellen Datei einlesen möchte, so bleibt einem nichts anderes übrig, als alle vorangehenden 149 Felder mit einzulesen, da DOS keine Möglichkeit hat, auf sozusagen rechnerischem Wege zu ermitteln, auf welchem Sektor und auf welcher Spur sich exakt das 150. Feld befindet. Man könnte zwar auch hierfür eine „Buchhaltung" einführen, doch würde diese ihrerseits besonders bei kurzen Feldern relativ viel Platz auf der Diskette beanspruchen, von dem Zeitaufwand für die Anlage einer solchen „Buchführung" ganz zu schweigen. Aber selbst wenn all dies kein Hindernis wäre, dann wäre diese „Buchführung" trotzdem ein (für Diskettenlaufwerke) nutzloses Unterfangen, denn nachträglich läßt sich bei einem sequentiellen File kein Feld mehr erweitern oder kürzen, ohne daß die Datei insgesamt neu abgespeichert wird. Dies ist zwar bei Festplattenlaufwerken mit hoher Zugriffszeit, nicht jedoch bei normalen Diskettenlaufwerken mit vergleichsweise sehr geringer Zugriffszeit, akzeptabel.

Der große Vorteil sequentieller Dateien beruht auf der Kompaktheit der Datenspeicherung, da gewissermaßen jede Speicherstelle auf der Diskette voll ausgenutzt wird. Hinsichtlich des Verwendungszwecks eignen sich sequentielle Files für solche Dateien, bei denen man nicht auf das EINZELNE Feld zugreifen muß, sondern wo die GESAMTHEIT der Felder im Vordergrund steht. Ein typisches Beispiel für sequentielle Dateiverwaltung ist damit ein Registerprogramm, weil hier die Stichwörter wahllos (z.B. nach dem Buchumbruch) eingegeben werden und die fertig sortierte Enddatei mit der Übertragung in die Photosetzmaschine ihren Zweck erfüllt hat.

Wenn dagegen Datensätze, z.B. Kundenadressen usw., ständig gepflegt werden müssen (durch Neueingabe, Änderung und Löschung), sollte man anstelle sequentieller Dateien Random-Files anlegen. Random-Dateien sind Dateien mit direktem Zugriff, da die Records oder Datensätze eine vordefinierte Länge haben. Nehmen wir an, eine Random-Datei bestehe aus 501 Records — numeriert von 0-500 — mit je 128 Zeichen Länge, dann ist es für DOS ein leichtes auszurechnen, auf welchem Sektor und auf welcher Spur sich z.B. der 150. Record befindet. Da ein Sektor 256 Bytes umfaßt, passen in diesem Fall 2 Records auf einen Datensektor. Im 1. Datensektor befinden sich die Records 0 und 1, im 2. Datensektor die Records 2 und 3 usw. Somit befindet sich der 150. Record in der ersten Hälfte des 75. Datensektors.

Random-Files werden geöffnet mit einer Längenabgabe (L = Recordlänge) als zu-

sätzlicher Parameter zum OPEN-Befehl. Ferner muß bei jedem READ- oder WRITE-Befehl die Record-Nummer (R) spezifiziert werden. Beispiel:

```
10 PRINT CHR$(4) ,,OPEN TELEFONLISTE,L50"
20 R = 1
30 PRINT CHR$(4)
40 INPUT ,,NAME + TEL. "; T$
50 PRINT CHR$(4) ,,WRITE TELEFONLISTE,R"R
60 PRINT X$ : R = R + 1 : GOTO 30
```

Die Record-Nummern R beginnen immer bei 0 und gehen maximal bis 32767. Die Record-Länge L muß im Bereich 1-32767 liegen. Normalerweise benutzt man den Record 0 nicht für eigentliche Daten, sondern z.B. für die Speicherung der Zahl der bisher belegten Records. Ein einzelner Record besteht meist aus mehreren Feldern, z.B. Vorname, Zuname, Straße usw. In diesem Sinne ist Record das Fremdwort für Datensatz. Speicher- und programmtechnisch lassen sich 2 Typen von Datensätzen unterscheiden:

Typ 1: Felder durch Returns abgegrenzt (Auszug)

```
 10 PRINT CHR$(4) ,,OPEN ADRESSEN,L132" : REM 128 + 4 RETURNS
 20 R = 1
 30 PRINT CHR$(4)
 40 INPUT ,,VORNAME: "; V$
 50 INPUT ,,ZUNAME: "; Z$
 60 INPUT ,,STRASSE: "; S$
 70 INPUT ,,PLZ und ORT: "; P$
 80 F = LEN (V$) + LEN (Z$) + LEN (S$) + LEN (P$)
 90 IF F > 128 THEN PRINT ,,RECORD ZU GROSS" : GOTO 30
100 PRINT CHR$(4) ,,WRITE ADRESSEN,R"R
110 PRINT V$ : PRINT Z$ : PRINT S$ : PRINT P$ : R = R + 1 : GOTO 30
```

Das Einlesen würde beim Typ 1 so aussehen (Auszug):

```
500 PRINT CHR$(4) ,,READ ADRESSEN,R"R
510 INPUT V$ : INPUT Z$ : INPUT S$ : INPUT P$
520 PRINT CHR$(4)
530 PRINT V$ : PRINT Z$ : PRINT S$ : PRINT P$
540 RETURN
```

Typ 2: Record durch Return abgegrenzt (Auszug)

```
 10  PRINT CHR$(4) ,,OPEN ADRESSEN,L129" : REM 128 + 1 RETURN
 20  R = 1
 30  PRINT CHR$(4)
 40  INPUT ,,VORNAME: "; V$ : V = LEN (V$) : IF V > 32 GOTO 40
 50  INPUT ,,ZUNAME: "; Z$ : Z = LEN (Z$) : IF Z > 32 GOTO 50
 60  INPUT ,,STRASSE: "; S$ : S = LEN (S$) : IF S > 32 GOTO 60
 70  INPUT ,,PLZ und ORT: "; P$ : P = LEN (P$) : IF P > 32 GOTO 70
 80  PRINT CHR$(4) ,,WRITE ADRESSEN,R"R
 90  F = V : PRINT V$; : GOSUB 140
100  F = Z : PRINT Z$; : GOSUB 140
110  F = S : PRINT S$; : GOSUB 140
120  F = P : PRINT P$; : GOSUB 140
130  PRINT : R = R + 1 : GOTO 30
140  D = 32 - F : IF D = 0 THEN 180
150  FOR X = 1 TO D
160  PRINT CHR$(32); : REM LEERTASTE
170  NEXT X
180  RETURN
```

Das Einlesen würde beim Typ 2 so aussehen (Auszug):

```
500  PRINT CHR$(4) ,,READ ADRESSEN,R"R
510  INPUT A$ : PRINT CHR$(4)
520  PRINT LEFT$ (A$, 32)
530  PRINT MID$ (A$, 33, 32)
540  PRINT MID$ (A$, 65, 32)
550  PRINT RIGHT$ (A$, 32)
560  RETURN
```

Beim Typ 1 ist jedes einzelne Feld durch ein Return vom nachfolgenden abgegrenzt, während beim Typ 2 lediglich das letzte Feld des Records, also der Record selbst, ein Return erhält. Der Typ 1 ist trotz der zusätzlichen Returns speicherökonomischer, da keine unnötigen Leertasten am Feldende verschwendet werden. Für Bildschirmmasken sind derartig variable Feldlängen jedoch ungeeignet. Ferner kann bei solchen Feldern kein formatierter, tabellarischer Ausdruck mehr zustande kommen, denn wie will man eine mehrspaltige Tabelle einrichten, wenn z.B. die Straße mal 35 Zeichen und mal 15 lang ist. Hinzu kommt, daß die Datentypistin sozusagen erst am Schluß weiß, ob sie kürzen muß oder nicht, da ja die tatsächliche Recordlänge erst mit der Eingabe des letzten Feldes vorliegt.

Der nachfolgende kleine Random-Test zeigt, daß Random-Files natürlich nicht nur „at random", d.h. zufällig oder wahlfrei, sondern auch sequentiell gelesen werden können. Allerdings ist jeweils der R-Parameter erforderlich.

```
100 REM = = = RANDOM-TEST = = =
110 PRINT CHR$ (4),,OPENRANDOM,L200"
120 FOR X = 1 TO 200
130 PRINT CHR$ (4),,WRITERANDOM,R"X
140 PRINT X · 1.2345
150 NEXT X
160 PRINT CHR$ (4),,CLOSE"
170 INPUT ,,(1-200)?"; A
180 A = A · 1.2345
190 PRINT CHR$ (4),,OPENRANDOM,L200"
200 FOR X = 1 TO 200
210 PRINT CHR$ (4),,READRANDOM,R"X
220 INPUT B
230 IF A = B THEN PRINT A,B: GOTO 270
240 NEXT X
250 PRINT CHR$ (4),,CLOSE"
260 END
270 PRINT ,,ENDE"
280 PRINT CHR$ (4),,CLOSE"
```

2.6.8.1 Vorformatierte Random-Files

Es gibt jedoch auch die Möglichkeit, Random-Files anzulegen, die zugleich sequentielle Dateien sind. Bei dem obigen Random-Test wird jeweils nur eine einzige Zahl in einem 200stelligen Record abgelegt. In Erweiterung unserer Erläuterungen zur Struktur von Textfiles gilt für Random-Files, daß nicht nur nach dem allerletzten Record, sondern auch zwischen den einzelnen Records NUL-Codes (Ctrl-0) abgespeichert sein können. Zur Verdeutlichung folgendes Extrembeispiel:

```
10 PRINT CHR$(4) ,,OPEN TEST"
20 PRINT CHR$(4) ,,DELETE TEST"
30 PRINT CHR$(4) ,,OPEN TEST,L"100
40 PRINT CHR$(4) ,,WRITE TEST,R"1000
50 PRINT ,,1"
60 PRINT CHR$(4) ,,CLOSE"
```

Dieses Programm würde einen Random-File erzeugen, der (später) fast die ganze Diskette einnehmen würde, obgleich zunächst nur eine einzige „1" abgespeichert wäre. Wäre die Datendiskette zuvor initialisiert worden, dann hätte die TEST-Datei folgende Struktur:

100 mal 1000 (0-999) Ctrl-0's + „1" + Return + 98 Ctrl-0's

Hätte es sich dagegen um eine alte Diskette gehandelt, auf der zahlreiche Files bereits gelöscht worden wären, dann würde die TEST-Datei anstelle „sauberer" Ctrl-0's „wirren Schrott" in Form von Resten alter Programme usw. enthalten.

Besser ist es, wenn man Random-Dateien quasi vorformatiert in dem Sinne, daß man die ganze Datei z.B. mit Leertasten, Nullen o.ä. beschreibt mit der Folge, daß solch eine Random-Datei dann auch wie eine sequentielle Datei behandelt werden kann, weil das Ctrl-0 dann nur noch am Ende der Datei vorkommt. Gegenstück zum obigen Extrembeispiel:

```
10  PRINT CHR$(4) „OPEN TEST" : PRINT CHR$(4) „WRITE TEST"
20  FOR X = 0 TO 1000
30  FOR Y = 1 TO 99 : PRINT CHR$(32); : NEXT Y : PRINT
40  NEXT X
50  PRINT CHR$(4) „CLOSE"
```

Ein weiterer Grund spricht für die „Vorformatierung" von Random-Files. Nichtvorformatierte und sozusagen „at random" beschriebene Files sind auf der Diskette nicht in homogener Reihenfolge (in auf- oder absteigenden Spuren) abgespeichert mit der Folge, daß die Zugriffsgeschwindigkeit entsprechend sinkt. Hinzu kommt, daß DOS bei Random-Files nur eine „unintelligente Buchführung" macht, die sogar die Anlage von Random-Files und/oder Random-Records zuläßt, die später nie mehr vollständig beschrieben werden können! Ein Beispiel:

```
100  L = 256 · 4
110  E = 400
120  PRINT CHR$(4) „OPEN TEST,L"L
130  FOR R = 0 TO E
140  PRINT CHR$(4) „WRITE TEST,R"R
150  PRINT R + 1000
160  NEXT R
170  PRINT CHR$(4) „CLOSE"
```

(Die nachfolgenden Überlegungen setzen Kenntnisse aus dem Teil II dieses Buches voraus, so daß der Leser ggf. diesen Absatz überspringen möge.)

Bei diesem Programm beträgt die Feldlänge 4 Sektoren oder 256 mal 4 = 1024 Bytes = 1 Kilobyte. Es werden 401 (0-400) Records mit jeweils einer 4stelligen Zahl (1000-1400) + Return beschrieben. Die Datei wird in Zeile 170 „ordnungsmäßig" ohne jegliche Fehlermeldung geschlossen. Ist jedoch wirklich alles in Ordnung? Gewiß nicht! Denn eine normale DOS 3.3-Diskette mit 140 Kilobytes Bruttospeicherkapazität kann natürlich nicht 401 Kilobytes an Daten umfassen. Was ist passiert? Da mit Zeile 150 nur ein — inklusive Return — 5 Bytes langes Feld auf die Diskette geschrieben wurde, hat DOS für jeden Record nur 1 Sektor belegt, jedoch bereits 3 weitere Sektoren vorgemerkt. Dieses Vormerken geschieht dadurch, daß in der TSL für je 3 reservierte Sektoren Nullen eingetragen werden. Die erste TSL für die ersten 122 Daten-Sektoren sowie der erste eigentliche Daten-Sektor seien nachstehend auszugsweise gelistet:

Track $13, Sektor $0F: Erster TSL-Sektor

```
00 15 0F 00 00 00 00 00  T. $15, S. $0F nächste TSL
00 00 00 00 13 0E T2 S2  T. $13, S. $0E erster Daten-Sektor
T3 S3 T4 S4 13 0D T2 S2  T. $12, S. $0D zweiter Daten-Sektor
T3 S3 T4 S4 13 0C T2 S2  usw.
T3 S3 T4 S4 13 0B T2 S2
T3 S3 T4 S4 13 0A 00 00  So sieht die TSL tatsächlich aus.
00 00 00 00 13 09 00 00
00 00 00 00 13 08 00 00  usw.
```

Track $13, Sektor $0E: Erster Daten-Sektor

```
B1 B0 B0 B0 8D 00 00 00    1000 + Return
00 00 00 00 00 00 00 00    usw.
```

Der erste Daten-Sektor ist aus der ersten TSL in der zweiten Zeile des Hex-Dumps durch 13 0E (= Track $13, Sektor $0E) erkenntlich. Danach stehen auf der Diskette 6 Nullen, die hier als T2 S2, T3 S3, T4 S4 kenntlich gemacht wurden. Gemeint ist damit, daß mit den Nullen die zweiten/dritten/vierten Daten-Tracks/Sektoren vorläufig in der TSL reserviert, jedoch noch nicht in der VTOC als belegt markiert wurden. Der Random-File benötigt damit zunächst nur 401 reine Daten-Sektoren sowie zu-

sätzlich mehrere TSL-Sektoren, die jedoch insgesamt auf die Datendiskette passen. Wollte man nun jedoch die 401 bereits angelegten Records vollständig beschreiben, d.h. jeweils bis zum 1024. Byte, dann würde bereits nach weniger als 100 Records die Fehlermeldung ,,DISK FULL" angezeigt.

Aus dem Beispiel kann man lernen, daß die Anlage dynamischer Random-Files, bei denen die Inhalte der Records und/oder die Anzahl der Records selbst kontinuierlich wächst, unter DOS nicht empfehlenswert ist.

2.6.9. APPEND, POSITION, BYTE

Diese DOS-Befehle werden selten benutzt, zumal insbesondere APPEND und POSITION zeitintensiv sind, d.h. unter Umständen beim normalen DOS 3.3 mehrere Minuten dauern können.

APPEND bewirkt das Anhängen (to append) eines speicherinternen Arrays an einen speicherexternen Textfile. Die Syntax ist APPEND — WRITE — PRINT. Ein vorangehendes OPEN ist überflüssig. Man beachte, daß APPEND definitionsgemäß nur für WRITE-Operationen gedacht. Beispiel:

```
10  PRINT CHR$(4) ,,OPEN TEST"
20  PRINT CHR$(4) ,,WRITE TEST"
30  FOR X = 0 TO 99 : PRINT X : NEXT
40  PRINT CHR$(4) ,,CLOSE"
50  REM JETZT FOLGT APPEND-BEFEHL
60  PRINT CHR$(4) ,,APPEND TEST"
70  PRINT CHR$(4) ,,WRITE TEST"
80  FOR X = 100 TO 199 : PRINT X : NEXT
90  PRINT CHR$(4) ,,CLOSE"
```

Die Felder eines Textfiles werden vom DOS bei bestimmten Befehlen wie numeriert behandelt, wobei die Numerierung mit Null beginnt (0, 1, 2, 3...). Symbolisch läßt sich dies so darstellen:

```
F0...rF1...rF2...rF3...rF4...rF5...rFn...Ctrl-0
        -Z-
        aP2
        rP0
```

d.h. Feld 0, gefolgt von Return, Feld 1, gefolgt von Return usw. bis zum Feld n, gefolgt von Return. Ein Feld besteht in der Regel aus 1...n Zeichen + Return, doch wird als Grenzfall vom DOS auch das nackte Return ohne vorangehende Zeichen als Feld aufgefaßt. Wenn ein File gerade geöffnet wurde, ist der Positionszeiger auf Feld 0 gerichtet. Nach dem ersten PRINT oder INPUT richtet sich der Zeiger auf Feld 1 usw. Jedes eingelesene oder gespeicherte Return erhöht also den Positionszeiger. Beim APPEND wird (bei DOS 3.3) der File solange eingelesen, bis kein Return mehr folgt, also der Endmarker Ctrl-0 erreicht wurde. Dann wird ab dieser Stelle der Array gespeichert. Bei diesem Verfahren ist es nicht verwunderlich, daß der APPEND-Befehl so lange dauert. (Diversi-DOS ist hier etwas „schlauer", da es gleich aufgrund der TSL zum letzten Datensektor geht, wodurch der APPEND-Befehl dann nur einen Bruchteil der Zeit in Anspruch nimmt.)

Man muß unterscheiden zwischen der relativen und der absoluten Feldposition. Wenn der Positionszeiger auf Feld 0 gerichtet ist, fallen absolute und relative Feldposition zusammen. Wenn der Zeiger (Z) jedoch z.B. nach 2 INPUTs von Feld 0 und 1 auf Feld 2 gerichtet wird, befindet man sich aus der Sicht von Feld 0 in der (absoluten) Position 2 (aP2) und aus der Sicht von Feld 2 in der relativen Position 0 (rP0); siehe obiges Schaubild.

Der R-Parameter bei Random-Files bezieht sich auf die absolute Position eines Records, der sich aus einem ODER mehreren, durch Returns getrennten Feldern zusammensetzen kann. Bei sequentiellen Files ist der R-Parameter ebenfalls möglich, doch bezieht er sich nunmehr auf die relative Feldposition, d.h. auf dasjenige Feld, daß dem r-ten Return vorausgeht. Nur direkt nach dem OPEN-Befehl ist die relative mit der absoluten Feldposition identisch. Beispiel:

```
 10 PRINT CHR$(4) ,,OPEN TEST" : REM ZEIGER DANACH AUF F0
 20 PRINT CHR$(4) ,,WRITE TEST"
 30 PRINT 0 : REM ZEIGER DANACH AUF F1
 40 PRINT 1 : REM ZEIGER DANACH AUF F2
 50 PRINT 2 : REM ZEIGER DANACH AUF F3
 60 PRINT 3 : REM ZEIGER DANACH AUF F4
 70 PRINT 4 : REM ZEIGER DANACH AUF F5
 80 PRINT CHR$(4) ,,CLOSE"
 90 REM JETZT FOLGT EIGENTLICHER TEST
100 PRINT CHR$(4) ,,OPEN TEST" : REM ZEIGER DANACH AUF F0
110 PRINT CHR$(4) ,,POSITION TEST,R1" : REM RELATIV = ABSOLUT!
120 PRINT CHR$(4) ,,READ TEST"
130 INPUT R : PRINT R : REM FELD 1, WEIL 0 + 1 = 1!
```

```
140  PRINT CHR$(4) ,,POSITION TEST,R4" : REM RELATIV!
150  PRINT CHR$(4) ,,READ TEST"
160  INPUT R : PRINT R : REM FELD 5, WEIL 1 + 4 = 5!
170  PRINT CHR$(4) ,,CLOSE"
```

R0 ist das momentane Feld, R1 das nächste, R2 das übernächste usw. Aus der Sicht von z.B. dem absoluten Feld F1 ist R4 das (1 + 4 =) 5. Feld. Nach OPEN-POSITION-WRITE bzw. OPEN-POSITION-READ gilt: absolut gleich relativ. Später nach POSITION-WRITE bzw. POSITION-READ gilt: absolut ungleich relativ. Das nachfolgende Demo-Programm faßt die neuen Befehle zusammen:

```
100  REM = = = APPEND-POSITION-TEST = = =
105  REM --- MERGE.FILE DRIVE 1 ---
110  PRINT CHR$ (4) ,,OPEN MERGE.FILE,D1"
120  PRINT CHR$ (4) ,,DELETE MERGE.FILE"
130  PRINT CHR$ (4) ,,OPEN MERGE.FILE"
140  PRINT CHR$ (4) ,,WRITE MERGE.FILE"
150  Z = 0 : PRINT ,,00000" : REM FORMATIERTER FELDZÄHLER
160  PRINT CHR$ (4) ,,CLOSE MERGE.FILE"
165  REM --- EINZELFILE.1 DRIVE 2 ---
170  PRINT CHR$ (4) ,,OPEN EINZELFILE.1,D2"
180  PRINT CHR$ (4) ,,DELETE EINZELFILE.1"
190  PRINT CHR$ (4) ,,OPEN EINZELFILE.1"
200  PRINT CHR$ (4) ,,WRITE EINZELFILE.1"
210  FOR X = 0 TO 500
220  PRINT STR$ (X) + ,, AAAAAAAAAAAAAAAAAAAA"
230  NEXT X
240  PRINT CHR$ (4) ,,CLOSE EINZELFILE.1"
245  REM --- EINZELFILE.2 DRIVE 2 ---
250  PRINT CHR$ (4) ,,OPEN EINZELFILE.2,D2"
260  PRINT CHR$ (4) ,,DELETE EINZELFILE.2"
270  PRINT CHR$ (4) ,,OPEN EINZELFILE.2"
280  PRINT CHR$ (4) ,,WRITE EINZELFILE.2"
290  FOR X = 0 TO 500
300  PRINT STR$ (X) + ,, BBBBBBBBBBBBBBBBBBB"
310  NEXT X
320  PRINT CHR$ (4) ,,CLOSE EINZELFILE.2"
330  HOME
340  PRINT ,,EINGABE M = RÜCKKEHR ZU MENÜ" : POKE 34, 2
350  REM --------- MENÜ ---------
```

```
360 HOME :B$ = ,,1 = EINZELFILE.1 = 2 = EINZELFILE.2 "
370 PRINT ,,1 ABSOLUTE POSITION"
380 PRINT ,,2 RELATIVE POSITION"
390 PRINT ,,3 APPEND"
400 PRINT : INPUT ,,";K$
410 HOME: ON VAL (K$) GOTO 420, 550, 670 : GOTO 360
415 REM ----- ABSOLUT -----
420 INPUT ,,WELCHE ABSOLUTE POSITION ";A$
430 IF A$ = ,,M" GOTO 360
440 PRINT B$;: INPUT ,,";Y$
450 IF Y$ = ,,1" THEN Y$ = ,,EINZELFILE.1": GOTO 470
460 Y$ = ,,EINZELFILE-2"
470 A = VAL (A$)
480 PRINT CHR$ (4) ,,OPEN" Y$ ,,,D2" : ZEIGER AUF NULL
490 PRINT CHR$ (4) ,,POSITION" Y$ ,,,R"A
500 PRINT CHR$ (4) ,,READ" Y$
510 INPUT X$
520 PRINT CHR$ (4) ,,CLOSE" Y$
530 PRINT X$
540 GOTO 420
545 REM ----- RELATIV -----
550 INPUT ,,WELCHE RELATIVE POSITION? ";A$
560 IF A$ = ,,M" GOTO 360
570 A = VAL (A$)
580 PRINT B$;: INPUT ,,";Y$
590 IF Y$ = ,,1" THEN Y$ = ,,EINZELFILE.1": GOTO 610
600 Y$ = ,,EINZELFILE-2"
610 PRINT CHR$ (4) ,,POSITION" Y$ ,,,R"A : ZEIGER RELATIV!
620 PRINT CHR$ (4) ,,READ" Y$
630 INPUT X$
640 PRINT CHR$ (4)
650 PRINT X$
660 GOTO 550
665 REM ----- APPEND -----
670 PRINT CHR$ (4) ,,APPEND MERGE.FILE,D1"
680 PRINT CHR$ (4) ,,WRITE MERGE.FILE"
690 PRINT X$
700 PRINT CHR$ (4) ,,CLOSE MERGE.FILE"
710 PRINT CHR$ (4) ,,OPEN MERGE.FILE,D1"
720 PRINT CHR$ (4) ,,WRITE MERGE.FILE"
```

```
730  Z = Z + 1: PRINT LEFT$ ( ,,00000", 5 — LEN ( STR$ ( Z ) ) ) ; Z
740  PRINT CHR$ (4) ,,CLOSE MERGE.FILE"
750  PRINT CHR$ (4) ,,CATALOG,D2" : REM ERSETZT POKE -21912, 2
760  GOTO 360
```

Verfolgt man den obigen Programmfluß, so stellt man fest, daß die Zeile 610 auch dann durchlaufen werden kann, wenn einer der beiden Einzelfiles durch Zeile 480 nicht zuvor geöffnet wurde, d.h. der POSITION-Befehl ist auch ohne vorangehenden OPEN-Befehl zulässig, setzt jedoch dann den Zeiger auf Null, als ob OPEN vorangegangen wäre.

Der BYTE-Befehl spezifiziert bei sequentiellen Dateien die absolute bytemäßige Stelle in einem File. Nehmen wir an, ein Brief umfasse 2.001 Zeichen (0-2000). Dann würde das Programm

```
10  PRINT CHR$(4) ,,OPEN BRIEF"
20  PRINT CHR$(4) ,,READ BRIEF,B1000"
30  FOR X = 1000 TO 2000
40  GET X$ : PRINT CHR$(4); : PRINT X$; NEXT X
50  PRINT : PRINT CHR$(4) ,,CLOSE"
```

die zweite Hälfte des Briefes einlesen und am Bildschirm anzeigen. Im Gegensatz zu dem R-Parameter beim POSITION-Befehl, der sich immer nur auf NACHFOLGENDE Felder richten kann, kann der B-Parameter bei sequentiellen Textfiles ähnlich wie der R-Parameter bei Random-Files ,,vorwärts und rückwärts" angewandt werden, d.h. mit dem BYTE-Befehl könnte man z.B. eine Datei rückwärts einlesen (was natürlich nicht besonders sinnvoll wäre):

```
10  PRINT CHR$(4) ,,OPEN ALPHABET"
20  PRINT CHR$(4) ,,WRITE ALPHABET"
30  FOR X = 65 TO 91 : PRINT CHR$(X); : NEXT : REM A BIS Z
40  PRINT : PRINT CHR$(4) ,,CLOSE"
50  PRINT CHR$(4) ,,OPEN ALPHABET"
60  FOR X = 25 TO 0 : REM Z BIS A
70  PRINT CHR$(4) ,,READ ALPHABET,B",X
80  GET X$ : PRINT CHR$(4);: PRINT X$ : NEXT X
90  PRINT CHR$(4);: PRINT : PRINT CHR$(4) ,,CLOSE"
```

Man beachte, daß man bei beiden Programmen wegen des GET-Befehls, der das Return unterdrückt, eine permanente CHR$(4)-DOS-Umschaltung vornehmen muß.

Da der BYTE-Befehl nicht nach Feldern, sondern nach absoluten Diskettenspeicherstellen vorgeht und insofern Returns wie jedes andere Zeichen behandelt, ist die Anwendung dieses Befehls normalerweise nur bei Dateien mit exakt bekannter Struktur sinnvoll.

Abschließend sei erwähnt, daß der POSITION- bzw. BYTE-Befehl R- bzw. B-Parameter im Bereich 0-32767 zuläßt.

2.6.10. EXEC (Executive Textfile)

EXEC ist die Abkürzung für execution (Befehlsausführung) bzw. executive textfile (Befehlsdatei). Darunter ist ein sequentieller Textfile zu verstehen, der aus Direktbefehlen besteht, die man normalerweise manuell über die Tastatur eingeben würde, die jedoch durch Starten des Execfiles automatisch ausgeführt werden. Execfiles erstellt man am besten mit einem Textverarbeitungsprogramm, das Textfiles erzeugt, beispielsweise mit dem Applewriter IIe, oder ersatzweise mit einem Applesoft-Programm. Da bei aktiven Execfiles Tastatureingaben verboten sind — diese würden nämlich als Execbefehle interpretiert — und ferner nicht jeder DOS- oder Applesoft-Befehl Bestandteil eines Execfiles sein darf, ist die Nutzanwendung dieses File-Typs auf wenige Spezialfälle begrenzt. Vereinfachtes Beispiel:

```
10 PRINT CHR$(4) ,,OPEN EXECUTIVE"
20 PRINT CHR$(4) ,,WRITE EXECUTIVE"
30 PRINT ,,RUN PROGRAMM 1"
40 PRINT ,,RUN PROGRAMM 2"
50 PRINT CHR$(4) ,,CLOSE"
```

Danach würde der manuell über die Tastatur eingegebene Befehl

EXEC EXECUTIVE

zunächst das Programm 1 und — nach Beendigung desselben — automatisch das Programm 2 starten.

2.6.10.1. Bload-Finder

Dieser Execfile, der z.B. mit dem Applewriter auf einer Diskette unter dem Namen BLOADFIND gespeichert sein mag, bewirkt durch den Befehl EXEC BLOAD-

FIND die Ermittlung der Startadresse und Länge eines zuvor geBLOADeten Binärsfiles in dezimal und hexadezimal. Die diversen Pokes dienen der Hexadezimalumrechnung. Das „?" steht für PRINT.

```
POKE 47097, PEEK (43634) + PEEK (43616) − INT ( ( PEEK (43634) + PEEK
(43616) ) / 256) · 256: POKE 47098, PEEK (43635) + PEEK (43617) + ( ( PEEK
(43634) + PEEK (43616) ) > 255)
HOME : ?,,BLOAD-FINDER − LETZTER BLOAD:"
?,,DEZ.": ?,,ANF: "; PEEK (43634) + 256 · PEEK (43635): ?,,LEN: "; PEEK
(43616) + 256 · PEEK (43617): ?,,END: "; PEEK (47097) + 256 · PEEK
(47098)
?,,HEX.": ?,,ANF: $";: POKE 70, PEEK (43634): POKE 71, PEEK (43635): PO-
KE 58, 64: POKE 59, 249: CALL 65209: ?: ?,,LEN: $";: POKE 70, PEEK (43616):
POKE 71, PEEK (43617): CALL 65209: ?: ?,,END: $";: POKE 70, PEEK (47097):
POKE 71, PEEK (47098): CALL 65209
?: ?,, NAME EINFÜGEN": ?: ?,, BSAVE.....,A"; PEEK (43634) + 256 · ·
PEEK (43635); ,,,L"; PEEK (43616) + 256 · PEEK (43617): HTAB 1: VTAB 17
```

2.6.10.2. List-Maker

Dieser Execfile − zu starten mit z.B. EXEC LIST.MAKER − speichert das sich gerade im RAM befindliche Applesoft-Programm unter dem Namen LIST auf der Diskette als Textfile (!), der z.B. dann mit dem Applewriter bearbeitet und später wieder mit EXEC LIST in ein normales (binäres) Applesoft-Programm zurückverwandelt werden kann. Die Pokes dienen u.a. der Tastatureingabe, da der GET-Befehl nicht möglich wäre.

```
POKE 768, 44: POKE 769, 16: POKE 770, 192: POKE 771, 173: POKE 772, 0:PO-
KE 773, 192: POKE 774, 16: POKE 775, 251: POKE 776, 44: POKE 777, 16: POKE
778, 192: POKE 779, 96
63999 POKE 51, 128: POKE 33, 33: ?: ?CHR$(4) ,,OPENLIST": ?CHR$(4) ,,DE-
LETELIST": ?CHR$(4) ,,OPENLIST": ?CHR$(4) ,,WRITELIST": LIST −
63998: ?CHR$(4) ,,CLOSELIST": TEXT: RETURN
HOME: INVERSE: ?,,LIST.MAKER": NORMAL: ?: ?,,INSERT DATA DISK
AND PRESS RETURN": CALL 768: GOSUB 63999: DEL 63999, 63999
```

TEIL II: DOS für Assembler-Programmierer

In diesem Teil II werden technische Details der Datenspeicherung, diverse Tips und Tricks (auch für Applesoft-Programmierer) sowie insbesondere der Direktzugriff auf einzelne Diskettensektoren auf Maschinenebene behandelt. Da die Assemblerprogrammierung nicht jedermanns Sache ist, enthält dieser Teil eine Vielzahl ausführlich kommentierter Komplettprogramme als Quell-Code, so daß der Leser die Programme oder Programmteile problemlos in seine eigenen Programme integrieren kann. Als Assembler wurde der „Big Mac" bzw. „Merlin" verwendet. Auf seltene Pseudo-Opcodes und Macros wurde bewußt in den Assemblerprogrammen verzichtet, damit eine Abänderung des jeweiligen Source-Codes für andere Assembler (z.B. LISA, S-C-Assembler usw.) sich problemlos gestaltet.

1. Catalog, TSL und VTOC

1.1. Catalog

Die Spur 17 (oder hexadezimal $11) ist die Catalog-Spur. Die Namen der Files (= Dateien) befinden sich in den Sektoren 15-1 (oder $0F-$01), d.h. zunächst wird Sektor 15, dann Sektor 14 belegt usw. Jeder Sektor kann 7 Dateinamen umfassen, so daß 15 mal 7 = 105 Dateinamen auf die Catalog-Spur passen. (Zum Vergleich ist beim ProDOS- ebenso wie beim Pascal-Betriebssystem der Catalog auf Spur 0.)

Jeder Catalog-Sektor ist folgendermaßen aufgebaut:

Byte 00	:	unbenutzt
Byte 01	:	Track-Nummer des nächsten Catalog-Sektors ($11)
Byte 02	:	Sektor-Nummer des nächsten Catalog-Sektors (z.B. $0E)
Byte 03-0A	:	unbenutzt
Byte 0B-2D	:	1. Dateiname-Eintrag
Byte 2E-50	:	2. Dateiname-Eintrag
		usw. bis
Byte DD-FF	:	7. Dateiname-Eintrag

Der Dateiname-Eintrag ist wie folgt gegliedert (am Beispiel des 1. Eintrages erläutert):

Byte 0B	:	Track-Nummer des TSL-Sektors
Byte 0C	:	Sektor-Nummer des TSL-Sektors
Byte 0D	:	Dateityp und Lock-Flag
Byte 0E-2B	:	Dateiname
Byte 2C-2D	:	Low-Byte und High-Byte der Gesamtsektorenanzahl

Es gibt u.a. folgende Dateitypen:

X0 = Textfile (T)
X1 = Integerfile (I)
X2 = Applesoftfile (A)
X4 = Binärfile (B)

(Der R-File = relokatives Maschinenprogramm ist hier nicht berücksichtigt, da es kaum Anwendung gefunden hat. Ausnahme: Toolkit-Diskette von Apple.)

Wenn der File gelockt ist, ist X = 8, ansonsten 0. Beispiele:

80 = gelockter Textfile
02 = ungelockter Applesoftfile

Der CATALOG-Befehl durchsucht zunächst Spur 17, Sektor 15 nach dem Dateinamen. Wenn er ihn dort nicht findet, durchsucht er Spur 17, Sektor 14 usw. Ist der Name gefunden, merkt er sich den Dateityp und liest den TSL-Sektor ein.

1.2. TSL = Track-Sektor-Liste

Die TSL ist das Spuren-Sektoren-Verzeichnis des Files und wird ihrerseits in einem Sektor abgespeichert. Aus diesem Grund weist der Catalog stets (mindestens) 1 Sektor mehr als Sektorenanzahl aus. Der TSL-Sektor hat folgende Struktur:

Byte 00	:	unbenutzt
Byte 01	:	Track der nächsten TSL (00 bei 1 TSL)
Byte 02	:	Sektor der nächsten TSL (00 bei 1 TSL)
Byte 03-04	:	unbenutzt
Byte 05-06	:	Sektor-Offset
Byte 07-0B	:	unbenutzt
Byte 0C-FF	:	maximal 122 Track-Sektor-Paare

(Sektor-Offset = sektormäßiger Abstand zwischen TSL-Sektor und erstem Daten-Sektor. Meist ist der TSL-Sektor der Sektor $0F einer Spur sowie der erste Daten-Sektor der Sektor $0E derselben Spur, so daß der Offset 00 00 beträgt.)

Aus dieser Übersicht ist ersichtlich, daß eine Datei mit 122 Datensektoren (auf der Diskette mit 123 Sektoren ausgewiesen) nur 1 TSL benötigt. Große Textfiles benötigen oft mehrere TSLs, maximal bis zu 5. Aufgrund der TSL wird die gewünschte Datei von der Diskette in den Speicher geladen.

Das Ende der ab Byte 0C aufgelisteten T-S-Paare ist für DOS normalerweise durch 2 Nullen erkenntlich. DOS liest grundsätzlich keine Files ein, deren Datensektoren sich auf der Spur 0 befinden! Bei spärlich beschriebenen Random-Files können sich jedoch innerhalb des Bereichs 0C-FF mehrfach Nullen befinden, wie bereits im Kapitel 2.6.8.1. beschrieben wurde.

1.3. VTOC = Volume Table of Contents

Neben dem Catalog gibt es sozusagen als Hauptinhaltsverzeichnis das VTOC oder Volume Table of Contents. Das VTOC befindet sich auf Spur 17, Sektor 0 in Form einer Art Bit-Tabelle, wobei in jeweils 2 Bytes die belegten und unbelegten Sektoren einer Spur verschlüsselt sind. Beispiel:

```
FEDC BA98 7654 3210  =  Sektoren-Nummern
1111 0000 1010 0011     F0 A3
```

Diese Bit-Verschlüsselung würde besagen, daß die Sektoren F, E, D, C, 7, 5, 1 und 0 frei sind, während die restlichen Sektoren der Spur belegt wären. In dem VTOC können übrigens mehr als 35 Spuren verschlüsselt werden, so daß DOS 3.3 mit entsprechenden Änderungen auch für Laufwerke mit z.B. 80 Spuren verwendet werden kann. In diesem Fall befindet sich übrigens der Catalog meist auf Spur 40 ($28). Komplette VTOCs sind in den Assemblerprogrammen „DOS-lose Datendiskette" und „DOSMOVER-RAMDISK-Driver" abgedruckt. Genaue Kenntnisse der Struktur des VTOCs sind bei RWTS-Programmen meist entbehrlich. Das Handling des VTOCs überlasse man in der Regel lieber dem DOS, denn 1 falsch gesetztes Bit, und schon ist eine Diskette partiell unbrauchbar geworden!

Zu Testzwecken initialisieren wir eine Leerdiskette mit dem folgenden Hello-Programm, das wir im Anschluß an die Initialisierung laufen lassen:

```
10 REM HELLO
20 PRINT CHR$(4) „OPEN TEXTFILE" : PRINT CHR$(4) „WRITE
   TEXTFILE"
```

```
30 FOR X = 0 TO 1023 : REM 1024 FELDER
40 PRINT X + 1000; : REM 4 STELLEN
50 FOR Y = 1 TO 27 : PRINT ,,-"; : NEXT Y : REM 27 STELLEN
60 PRINT : REM RETURN = 1 STELLE
70 NEXT X : REM ZUSAMMEN 32 STELLEN
80 PRINT CHR$(4) ,,CLOSE"
```

Auf einen Textfile-Sektor passen damit 256 : 32 = 8 Felder. Der Textfile nimmt damit 1024 : 8 = 128 Daten-Sektoren ein. Da der letzte Daten-Sektor ,,glatt aufgeht" (256 Bytes), entfällt hier das Ctrl-0. Hinzu kommen noch 2 TSL-Sektoren. Das Hello-Programm selbst nimmt 2 Sektoren ein, so daß der Catalog so aussieht:

```
A 002 HELLO
T 130 TEXTFILE
```

Nachfolgend werden als Hex-Dumps der VTOC-Sektor, der erste Catalog-Sektor, der erste und letzte (= zweite) TSL-Sektor des Textfiles sowie der erste und letzte (= 128ste) Textfile-Daten-Sektor und schließlich der Hello-Programm-Sektor abgebildet.

1.4. Track-Sector-Tracer

Dieses kleine, aber sehr anschauliche Programm wird mit BRUN gestartet und zeigt bis zum nächsten Booten alle jeweils eingelesenen bzw. gespeicherten Sektoren in der Form ,,T:11 S:00" an. Da es den Anfang der RWTS ($BD00) modifiziert, funktioniert es nur bei normalem 48K DOS.

VTOC-Sektor

```
Z A P - S6,2 V$FE T$11 S$00
:PRINT
00 04110F030000FE0000000000 .....↑.....
0C 00000000000000000000000000 ............
18 00000000000000000000000000 ............
24 0000007A0000000000000000 ...z........
30 1B0100002310000100000000 ....#.......
3C 000000000000000FFFF0000 ........__..
48 FFFF0000FFFF0000FFFF0000 __..__..__..
54 FFFF0000FFFF0000FFFF0000 __..__..__..
60 FFFF0000FFFF0000FFFF0000 __..__..__..
6C FFFF0000FFFF0000FFFF0000 __..__..__..
78 FFFF0000000000003FFF0000 __.......?_..
84 00000000000000000000000000 ............
90 00000000000000000000000000 ............
9C 0000000000000003FFF0000 .........?_..
A8 FFFF0000FFFF0000FFFF0000 __..__..__..
B4 FFFF0000FFFF0000FFFF0000 __..__..__..
C0 FFFF0000000000000000000000 __..........
CC 00000000000000000000000000 ............
D8 00000000000000000000000000 ............
E4 00000000000000000000000000 ............
F0 00000000000000000000000000 ............
```

Erster TSL-Sektor

```
Z A P - S6,2 V$FE T$13 S$0F
:PRINT
00 001A0400000000000000000000 ............
0C 130E130D130C130B130A1309 ............
18 130813071306130513041303 ............
24 130213011300140F140E140D ............
30 140C140B140A14091408 1407 ............
3C 1406140514041403140214021401 ..........
48 1400150F150E150D150C150B ............
54 150A150915081507150615051 ............
60 150415031502150115001 60F ............
6C 160E160D160C160B160A1609 ............
78 160816071606160516041603 ............
84 160216011600170F170E170D ............
90 170C170B170A17091708 1707 ............
9C 170617051704170317021701 ............
A8 17001 80F180E180D180C180B ............
B4 180A1809180818071806 1805 ............
C0 18041803180218011800190F ............
CC 190E190D190C190B190A1909 ............
D8 190819071906190519041903 ............
E4 19021901190 01A0F1A0E1A0D ............
F0 1A0C1A0B1A0A1A091A0B1A07 ............
```

Catalog-Sektor

```
Z A P - S6,2 V$FE T$11 S$0F
:PRINT
00 00110E000000000000000012 ............
0C 0F02C8C5CCCCCFA0A0A0A0A0 ..HELLO
18 A0A0A0A0A0A0A0A0A0A0A0A0
24 A0A0A0A0A0A0A0A00200130F ....
30 00D4C5D8D4C6C9CCC5A0A0A0 .TEXTFILE
3C A0A0A0A0A0A0A0A0A0A0A0A0
48 A0A0A0A0A0A0A0B200000000 .....
54 00000000000000000000000000 ............
60 00000000000000000000000000 ............
6C 00000000000000000000000000 ............
78 00000000000000000000000000 ............
84 00000000000000000000000000 ............
90 00000000000000000000000000 ............
9C 00000000000000000000000000 ............
A8 00000000000000000000000000 ............
B4 00000000000000000000000000 ............
C0 00000000000000000000000000 ............
CC 00000000000000000000000000 ............
D8 00000000000000000000000000 ............
E4 00000000000000000000000000 ............
F0 00000000000000000000000000 ............
```

Zweiter und letzter TSL-Sektor

```
Z A P - S6,2 V$FE T$1A S$04
:PRINT
00 00000000007A0000000000 .....z.....
0C 1A031A021A011A001B0F1B0E ............
18 00000000000000000000000000 ............
24 00000000000000000000000000 ............
30 00000000000000000000000000 ............
3C 00000000000000000000000000 ............
48 00000000000000000000000000 ............
54 00000000000000000000000000 ............
60 00000000000000000000000000 ............
6C 00000000000000000000000000 ............
78 00000000000000000000000000 ............
84 00000000000000000000000000 ............
90 00000000000000000000000000 ............
9C 00000000000000000000000000 ............
A8 00000000000000000000000000 ............
B4 00000000000000000000000000 ............
C0 00000000000000000000000000 ............
CC 00000000000000000000000000 ............
D8 00000000000000000000000000 ............
E4 00000000000000000000000000 ............
F0 00000000000000000000000000 ............
```

```
Z A P - S6,2 V$FE T$13 S$0E
:PRINT
00 B1B0B0B0ADADADADADADADAD 1000--------
0C ADADADADADADADADADADAD ------------
18 ADADADADADADADBDB1B0B0B1 --------.1001
24 ADADADADADADADADADADAD ------------
30 ADADADADADADADADADADAD ------------
3C ADADADBDB1B0B0B2ADADADAD ---.1002----
48 ADADADADADADADADADADAD ------------
54 ADADADADADADADADADADBD ------------.
60 B1B0B0B3ADADADADADADAD 1003--------
6C ADADADADADADADADADADAD ------------
78 ADADADADADADADBDB1B0B0B4 --------.1004
B4 ADADADADADADADADADADAD ------------
90 ADADADADADADADADADADAD ------------
9C ADADADBDB1B0B0B5ADADADAD ---.1005----
A8 ADADADADADADADADADADAD ------------
B4 ADADADADADADADADADADBD ------------.
C0 B1B0B0B6ADADADADADADAD 1006--------
CC ADADADADADADADADADADAD ------------
D8 ADADADADADADADBDB1B0B0B7 --------.1007
E4 ADADADADADADADADADADAD ------------
F0 ADADADADADADADADADADAD ------------
```

Erster Daten-Sektor

```
Z A P - S6,2 V$FE T$1B S$0E
:PRINT
00 B2B0B1B6ADADADADADADAD 2016--------
0C ADADADADADADADADADADAD ------------
18 ADADADADADADADBDB2B0B1B7 --------.2017
24 ADADADADADADADADADADAD ------------
30 ADADADADADADADADADADAD ------------
3C ADADADBDB2B0B1B8ADADADAD ---.2018----
48 ADADADADADADADADADADAD ------------
54 ADADADADADADADADADADBD ------------.
60 B2B0B1B9ADADADADADADAD 2019--------
6C ADADADADADADADADADADAD ------------
78 ADADADADADADADBDB2B0B2B0 --------.2020
84 ADADADADADADADADADADAD ------------
90 ADADADADADADADADADADAD ------------
9C ADADADBDB2B0B2B1ADADADAD ---.2021----
A8 ADADADADADADADADADADAD ------------
B4 ADADADADADADADADADADBD ------------.
C0 B2B0B2B2ADADADADADADAD 2022--------
CC ADADADADADADADADADADAD ------------
D8 ADADADADADADADBDB2B0B2B3 --------.2023
E4 ADADADADADADADADADADAD ------------
F0 ADADADADADADADADADADAD ------------
```

Letzter Daten-Sektor

Programm-Sektor des Hello-Programms

```
Z A P - S6,1 V$FE RSA=$0000
:PRINT
00 D7000C080A00B248454C4C4F W.....2HELLO
0C 003B0B1400BAE72B3429224F .:...:G(4)"O
18 50454E2054455585446494C45 PEN TEXTFILE
24 223ABAE72B342922575249 54 "::G(4)"WRIT
30 4520544558544649 4C452200 E TEXTFILE".
3C 560B1E00B15BD030C1313032 V....XP0A102
48 333AB2313032342046454C44 3:21024 FELD
54 4552006E082B00BA58CB3130 ER.N.(.:XH10
60 30303B3AB234205354454C4C 00;:24 STELL
6C 454E008F083200B159D031C1 EN...2..YP1A
78 32373ABA222D223B3AB2593A 27::"-";:.Y:
84 B23237205354454C4C454E00 227 STELLEN.
90 A8083C00BA3AB25245545552 (.<.:2RETUR
9C 4E203D2031205354454C4C45 N = 1 STELLE
A8 00C4084600B2583AB25A5553 .D.F..X:2ZUS
B4 414D4D454E20333220535445 AMMEN 32 STE
C0 4C4C454E00D5085000BAE728 LLEN.U.P.:G(
CC 342922434C4F534522000000 4)"CLOSE"...
D8 010000000000000000000000 ............
E4 000000000000000000000000 ............
F0 000000000000000000000000 ............
```

```
]ASM
                        1               ORG   $300
                        2   *
                        3   *  TRACK-SECTOR-TRACER/11.03.84
                        4   *
                        5   IND         EQU   $CE
                        6   GETIOB      EQU   $3E3
                        7   RWTS        EQU   $BD00
                        8   RWTSCONT    EQU   $BD04
                        9   PRINT       EQU   $FDED
                       10   HEXOUT      EQU   $FDDA
                       11   *
0300: A9 4C             12   RWTSHOOK    LDA   #$4C         ;JMP
0302: 8D 00 BD          13               STA   RWTS
0305: A9 10             14               LDA   #<TRKSEC
0307: 8D 01 BD          15               STA   RWTS+1
030A: A9 03             16               LDA   #>TRKSEC
030C: 8D 02 BD          17               STA   RWTS+2
030F: 60                18               RTS
                       19   *
0310: 8C 5F 03          20   TRKSEC      STY   YSAVE
0313: 8D 60 03          21               STA   ASAVE
                       22   *
0316: 20 E3 03          23               JSR   GETIOB
0319: 84 CE             24               STY   IND
031B: 85 CF             25               STA   IND+1
                       26   *
031D: A5 24             27               LDA   $24         ;CH
031F: F0 05             28               BEQ   TRACK
0321: A9 8D             29               LDA   #$8D
0323: 20 ED FD          30               JSR   PRINT
                       31   *
0326: A9 D4             32   TRACK       LDA   #"T"
0328: 20 ED FD          33               JSR   PRINT
032B: A9 BA             34               LDA   #":"
032D: 20 ED FD          35               JSR   PRINT
0330: A0 04             36               LDY   #4
0332: B1 CE             37               LDA   (IND),Y     ;TRK
0334: 20 DA FD          38               JSR   HEXOUT
0337: A9 A0             39               LDA   #$A0
0339: 20 ED FD          40               JSR   PRINT
                       41   *
033C: A9 D3             42   SECTOR      LDA   #"S"
033E: 20 ED FD          43               JSR   PRINT
0341: A9 BA             44               LDA   #":"
0343: 20 ED FD          45               JSR   PRINT
0346: A0 05             46               LDY   #5
0348: B1 CE             47               LDA   (IND),Y     ;SECT.
034A: 20 DA FD          48               JSR   HEXOUT
034D: A9 8D             49               LDA   #$8D
034F: 20 ED FD          50               JSR   PRINT
                       51   *
0352: AC 5F 03          52               LDY   YSAVE
0355: 84 48             53               STY   $48
```

```
0357: AD 60 03  54              LDA  ASAVE
035A: 85 49     55              STA  $49
035C: 4C 04 BD  56              JMP  RWTSCONT
                57   *
035F: 00        58   YSAVE     HEX  00
0360: 00        59   ASAVE     HEX  00
```

1.5. DOS-Puffer

Bei MAXFILES 3 gibt es 3 externe DOS-Puffer:

1. $ 9600 - $ 9852 (wird zuerst belegt)
2. $ 9853 - $ 9AA5
3. $ 9AA6 - $ 9CF8

Im einzelnen gilt am Beispiel des erstbenutzten Puffers:

$ 9600 - $ 96FF : Daten-Puffer (256 Bytes)
$ 9700 - $ 97FF : TSL-Puffer (256 Bytes)
$ 9800 - $ 9852 : Datenname u.a.

Außerdem unterhält DOS diverse interne Puffer, z.B.:

$ B3BB - B4BA: VTOC-Puffer (256 Bytes)
$ B4BB - B5BA : Catalog-Sektor-Puffer (256 Bytes)

2. Fehlermeldungen

Man muß zwei Arten von Fehlermeldungen unterscheiden, nämlich ersten Fehlermeldungen auf der Ebene der RWTS sowie zweitens Fehlermeldungen auf der Ebene des Applesoft-Programms, die sich ihrerseits in reine Applesoft-Interpreter- und zweitens DOS-Interpreter-Fehlermeldungen aufteilen lassen.

2.1. RWTS-Fehlermeldungen

Nach der Rückkehr von der RWTS (s. S. 109 ff.) wird das Carry-Flag (Übertrag-Flag) gesetzt (BCS ERROR), falls ein Fehler auftrat. Die Fehler-Nummer wird dann an der entsprechenden Stelle des IOBs (= Input-Output-Blocks) abgelegt. Tritt kein RWTS-Fehler auf, dann wird zwar das Carry-Flag nicht gesetzt (BCC NO ERROR), aber skurrilerweise das letzte Byte des zuletzt eingelesenen Sektors als „Fehler-Code" abgelegt. Es gibt nur folgende 4 RWTS-Fehlermeldungen:

08 = Initialisierungsfehler
10 = Diskette schreibgeschützt
20 = falsche Volume-Nummer
40 = Drive-Fehler (I/O-Fehler)

Der letztere I/O-Fehler kann besagen: keine Diskette im Laufwerk, Slot nicht belegt, defekte Diskette, geschützte Programmdiskette usw.

2.2. Applesoft- und DOS-Fehlermeldungen

Falls das On-Error-Flag gesetzt ist, wird die Fehler-Nummer in der Speicherstelle 222 ($DE) abgelegt, ansonsten wird der Fehler angezeigt und das Programm unterbro-

chen. Der Vollständigkeit halber werden nachstehend neben den reinen DOS-Fehlermeldungen auch die Applesoft-Fehlermeldungen aufgeführt:

Applesoft:

0 = NEXT WITHOUT FOR (falsche For-Next-Schleife)

DOS:

1 = LANGUAGE NOT AVAILABLE (z.B. Integer-Basic nicht geladen)
2 = RANGE ERROR (falsche DOS-Parameter)
3 = ebenfalls RANGE ERROR
4 = WRITE PROTECTED (Schreibschutz-Klebestreifen entfernen)
5 = END OF DATA (Textfile über Endmarker hinaus eingelesen)
6 = FILE NOT FOUND (Dateiname falsch)
7 = VOLUME MISMATCH (falsche Volume-Nummer)
8 = I/O ERROR (Drive-Error, siehe oben)
9 = DISK FULL (VTOC voll)
10 = FILE LOCKED (Lock-Sternchen entfernen)
11 = SYNTAX ERROR (Befehlsname vertippt o.ä.)
12 = NO BUFFERS AVAILABLE (Maxfiles erhöhen)
13 = FILE TYPE MISMATCH (Binär- mit Textfile verwechselt o.ä.)
14 = PROGRAM TOO LARGE (Basic-Programm würde über HIMEM gehen)
15 = NOT DIRECT COMMAND (indirekter Befehl im Direktmodus)

Erneut Applesoft:

16 = SYNTAX (vertippter Applesoft-Befehl)
22 = RETURN WITHOUT GOSUB (ein RETURN zuviel)
42 = OUT OF DATA (bei DATA-Statements)
53 = ILLEGAL QUANTITY (zu große Integer-Zahl usw.)
69 = OVERFLOW (z.B. Zahl zu groß)
77 = OUT OF MEMORY (LOMEM größer als HIMEM usw.)
90 = UNDEFINED STATEMENT (GOTO-Programmzeile fehlt usw.)
107 = BAD SUBSCRIPT (DIM-Index zu groß)
120 = REDIMENSIONED ARRAY (erneutes DIM ohne vorangehendes CLEAR)
133 = DIVISION BY ZERO (Division durch Null)
163 = TYPE MISMATCH (z.B. String mit Zahl verwechselt)

176 = STRING TOO LONG (String länger als 255 Zeichen)
191 = FORMULA TOO COMPLEX (zu starke Verschachtelung)
224 = UNDEFINED FUNCTION (DEF-Statement fehlt)
254 = INPUT ERROR (welcher?)
255 = CTRL-C Programmabbruch

Die reinen DOS-Fehlernummern liegen also im Bereich 1-15.

Mit dem folgenden Testprogramm kann man die Fehler-Nummern ausprobieren, indem man illegale DOS-Befehle eingibt:

```
10 ONERR GOTO 60
20 PRINT ,,Mit Reset abbrechen"
30 INPUT ,,BEFEHL: ";X$
40 PRINT CHR$(4);X$
50 GOTO 20
60 F = PEEK (222)
70 PRINT ,,FEHLER-NR. ";F
80 GET X$ : GOTO 20
```

Die Fehler-Nummer kann also mit PEEK (222) ermittelt werden. Mit POKE 216,0 kann demgegenüber das On-Error-Flag wieder zurückgesetzt werden.

3. Vermischte Tips, Tricks und Patches

Sofern bei den nachstehenden Tips DOS-Modifikationen (= „Patches" = „Flickereien") angegeben werden, beziehen sie sich auf das normale DOS 3.3, das im Bereich $9D00-$BFFF liegt (48K DOS). Man hüte sich davor, bereits gepatchte DOS-Varianten ohne nähere Prüfung erneut zu patchen! Außerdem beachte man, daß sich die Patches teilweise gegenseitig ausschließen. Persönlich bin ich ein Gegner von DOS-Patches, da man zum Schluß nicht mehr weiß, was wann und wie gepatcht wurde. Im allgemeinen ist es empfehlenswerter, für einen bestimmten Anwendungszweck erforderliche Patches nach dem Booten an dem normalen DOS 3.3 durch Pokes im Hello-Programm vorzunehmen, anstatt daß man die Patches durch INIT auf der Diskette verewigt. DOS-Varianten wie z.B. Diversi-DOS sind eigentlich nicht umgeschriebenes, sondern stark gepatchtes DOS 3.3. Das Umschreiben hätte nämlich bedeutet, daß Systemadressen hätten geändert werden müssen. Gerade dies aber ist im Interesse der Kompatibilität mit DOS 3.3 nicht möglich. Etliche Programme poken leider wahllos im DOS herum, so daß sie nur mit DOS 3.3 funktionieren. Beispielsweise läuft der TASC-Compiler nicht mit der stark modifizierten DOS-Variante Diversi-DOS, weil er während des Compilierungsvorgangs das DOS vorübergehend an bestimmten Stellen verändert, die ihrerseits z.B. von der schwach geänderten DOS-Variante Superdos nicht berührt werden, so daß sich Superdos mit dem TASC verträgt.
Insgesamt gesehen kann Diversi-DOS als die ausgereifteste DOS 3.3-Verbesserung angesehen werden, da unter möglichst großer Wahrung der DOS 3.3-Kompatibilität die Zugriffsgeschwindigkeit bei ALLEN File-Typen maßgeblich verbessert wurde.

3.1 CAT statt CATALOG?

Das Ändern, Kürzen und Eindeutschen von Befehlsnamen und Fehlermeldungen ist einfach, da sich die entsprechenden Textstellen im RAM (und nicht wie beim Applesoft im ROM) befinden. Die Befehlsnamen stehen im Speicher ab $A884, die Fehlermeldungen ab $A971. „DISK VOLUME" steht ab $B3AF (rückwärts: EMULOV

KSID), „APPLESOFT" steht ab $AAB8 usw. Es gibt Programm wie DOS BOSS, die speziell zum Ändern dieser DOS-Textstellen bestimmt sind.

3.2. CLOSE bei ONERR

Wenn bei einem Applesoft-Programm mit ONERR GOTO das Errorflag gesetzt ist, brauchen nach einem Fehler möglicherweise noch offene Files nicht mehr vom Applesoft-Programm geschlossen werden, da dies DOS selbst tut. Das besagt jedoch gleichzeitig, daß z.B. das Einlesen oder Speichern von Textfiles nicht an der Stelle fortgesetzt werden kann, bei der die Fehlermeldung auftrat.

3.3. GET bei Execfiles

GET und INPUT sind bei EXEC nicht erlaubt. Zulässig ist dagegen die direkte Tastaturabfrage, die GET simuliert, z.B. mit

```
KEY LDA $C000
    BPL KEY
    BIT $C010
    RTS
```

Hierzu muß man ggf. ein kleines Maschinenprogramm poken, wie es in Kapitel 2.6.10.2 beim „List-Maker" vorgeführt wurde.

3.4. CHR$(13) + CHR$(4)

Das DOS-Umschaltzeichen Ctrl-D ist bekanntlich nur wirksam, wenn (implizite oder explizite) ein Ctrl-M oder Return vorausging, d.h. wenn sich der Cursor — beim laufenden Programm — sozusagen am linken Bildschirmrand befinden würde. Oft sieht man deshalb in Programmlistings „sicher ist sicher"-Zeilen in der Art

```
10  PRINT CHR$(13) CHR$(4) „CLOSE"
```

Hier ist jedoch speziell bei Textfile-WRITE-Operationen Vorsicht geboten, da u.U. auf diese Weise ungewollt ein zusätzliches Return auf der Diskette mit abgespeichert werden kann. Deshalb ein Programm jeweils auf diese Gefahr hin überprüfen. (Ausführliche Beispiele stehen im Teil I, Kapitel 2.6).

3.5. Zahlenspeicherung

Zahlen werden auf der Diskette stets wie Strings gespeichert. Eine binäre Zahlenspeicherung ist von DOS nicht vorgesehen. Bei Random-Files beachte man, daß Fließkommazahlen bis zu 15 Stellen (-1.23456789E-12) und Ganzzahlen bis zu 6 Stellen (-12345) lang sein können.

3.6. BSAVE-Sektoreinsparung bei Hires-Bildern

BSAVE hat einen Fehler; es speichert nämlich immer 1 Byte mehr als spezifiziert. Allerdings kann man dieses zusätzliche Byte mit BLOAD niemals einlesen. Hinzu kommen noch die obligatorischen 4 Bytes für Startadresse und Länge, also insgesamt 5 Bytes. Bei Hires-Bildern kommt es in der Regel auf die letzten 5 Bytes nicht an. Folglich kann man mit z.B.

BSAVE BILD, A$2000,L$1FFB (statt L$2000)

einen Sektor sparen (jetzt 33 statt 34 Sektoren)

3.7. SAVE ohne Dateiname

SAVE ohne folgenden Namen ist ein Kassettenbefehl! Da meist kein Kassettenrecorder angeschlossen ist, „hängt" der Apple, so daß man sich nur mit Reset retten kann.

3.8. Reset und DOS-Vektoren

Wenn man den RESET-Vektor so ändert, daß z.B. nach Drücken der Reset-Taste der RUN-Befehl ausgeführt wird, also das Applesoft-Programm neu gestartet wird, dann muß man auch dafür Sorge tragen, daß DOS mit PR#0 : IN#0 : CALL 1002 wieder angehängt wird, sonst werden CATALOG usw. nicht mehr als DOS-Befehle erkannt. Beispiel:

```
10  POKE 1010,0 : POKE 1011, 3 : POKE 1012, 166 : REM RESET-VEKTOR
    AUF 768
```

```
20 POKE 768, 162 : POKE 769, 223 : POKE 770, 154 : POKE 771, 76 : POKE
   772, 102 : POKE 773, 213 : REM BEI RESET RUN-BEFEHL AUSFÜH-
   REN
30 PR#0 : IN#0 : CALL 1002 : REM DOS WIEDER ANHÄNGEN
40 REM AB HIER WEITERES PROGRAMM
```

3.9. Einseitige Disketten beidseitig bespielen?

Die normalen Apple-Laufwerke sind für beidseitiges Lesen nicht bestimmt (es gibt ja nur 1 Lesekopf). Durch Anbringung einer zweiten Schreibschutzeinkerbung kann man Disketten theoretisch auch auf der Rückseite benutzen, doch gibt es Firmen, die ihre SS (= single sided) Disketten auf der Rückseite unbearbeitet lassen oder gar aufrauhen. Also wenn Sie auf die Intaktheit des Lesekopfes Wert legen, Finger weg von SS-Disketten. Aber auch DS (= double sided) Disketten sind nicht ganz unproblematisch, weil der dem Lesekopf gegenüberliegende Andruck-Filz mit der Zeit nicht mehr der sauberste ist.

3.10. BRUN und EXEC Hello-Programme

Hello-Programme sind normalerweise Applesoft- oder auch Integer-Programme. Ein binäres Hello-Programm entsteht nicht dadurch, daß man den Binärfile lädt und dann die Diskette mit INIT HELLO initialisiert. Damit DOS nach dem Booten keinen Basicfile startet, sind folgende Pokes vor dem INIT erforderlich:

```
POKE 40514,52 : REM $9E42:34 BRUN
POKE 40514,20 : REM $9E42:14 EXEC
```

Nach dem INIT lösche man das Applesoft-Hello-Programm und kopiere dann von einer anderen Diskette den Binär- oder Execfile auf die initialisierte Diskette unter dem Namen HELLO.

3.11. Bank 2: E000-Patch

Nach dem Booten pokt DOS in die LC-Speicherstelle $E000 eine Null, damit Integer-Basic als nicht mehr in der LC vorhanden angesehen und bei Bedarf frisch

geladen wird. Durch

BFD3: EA EA EA

und Initialisierung des derart modifizierten DOS wird diese Routine lahmgelegt.

3.12. RUN-Modus bei Maschinenprogrammen

Die NUR-Indirektbefehle OPEN, READ usw. hängen damit zusammen, daß DOS prüft, ob ein Basic-Programm gerade läuft, d.h. sich im RUN-Modus befindet, oder nicht. Bei Maschinenprogrammen muß man DOS austricksen, denn Binärprogramme sind selbstverständlich keine Basic-Programme. Der einfachste Weg ist das Starten eines Assemblerprogramms durch ein kurzes Hello-Programm, weil dann der RUN-Modus automatisch hergestellt und nachher beibehalten wird. Ist der Binärfile jedoch z.B. das Hello-Programm selbst (siehe oben), dann füge man folgende Assemblerbefehle ein:

```
LDA   # $80
STA   $33     Prompt
STA   $76     normalerweise High Byte der Basic-Zeile
STA   $D9     eigentliches RUN-Modus Flag
```

3.13. RUN-Modus bei READ-WRITE nach Programmabbruch

Gelegentlich will man ein Applesoft-Programm, das Textfiles einliest oder speichert, nach einem Programmabruch ab einer bestimmten Programmzeile mit GOTO manuell fortsetzen. Dies geht jedoch nicht, weil DOS dann keinen RUN-Modus erkennt. Hier hilft folgender Trick (über die Tastatur im Direkt-Modus eintippen):

POKE 51,128 : GOTO 100

wobei z.B. in Zeile 100 steht:

100 PRINT CHR$(4) „OPEN DATEI“

Durch diesen Poke-Befehl wird DOS nicht modifiziert. Statt dessen kann folgender Patch vorgenommen werden:

A021: EA EA EA oder alternativ
A65E: 18 60 (POKE 42590, 24 : POKE 42591, 96)

3.14. Mystery Parameter (Zwangs-RUN-Modus, List-Schutz)

Durch POKE 214, 128 wird jeder Basic- und Applesoft-Befehl als RUN ausgeführt, d.h. man kann danach z.B. ein Programm nicht mehr listen. Es handelt sich hierbei um eine Prüfroutine, die sowohl im Applesoft-Interpreter wie auch im DOS steht, z.B. im Bereich $A397-$A3D4. POKE 214, 0 hebt den List-Schutz wieder auf.

3.15. Catalog-Pause

Nach jeweils 20 Dateinamen wartet DOS beim Catalog auf einen beliebigen Tastendruck. Wen dies stört, kann diese Pause lahmlegen mit

```
POKE 44601, 234 : REM $EA = NOP bei $AE39
POKE 44602, 234
POKE 44603, 234
```

(kürzer: POKE 44601, 96 : REM $60 = RTS)

Der CATALOG-Befehl kann auch während der Pause mit ESC (ASCII $9B) oder einer anderen Taste abgebrochen werden durch folgenden Patch:

```
AE39:   20   B4   B6   (JSR $B6B4; APPEND-Patch-Stelle)
B6B4:   20   0C   FD   (JSR RDKEY)
        C9   9B        (CMP ESC)
        D0   03        (BNE EXIT)
        4C   D0   03   (JMP DOSWARM)
EXIT    60
```

3.16. COUT in Maschinenprogrammen

Wenn ein Assemblerprogramm nicht von einem Hello-Programm geBLOADet und mit CALL XXX gestartet, sondern über die Tastatur geBRUNt wird, sollte es stets über JMP $3D0 (Warmstart) oder noch besser über JMP $3D3 (Kaltstart) verlassen werden wegen DOS-interner Probleme mit dem Stack-Pointer, und zwar stets dann, wenn Strings über COUT (JSR $FDED) ausgegeben werden. COUT1 (JSR $FDF0) ist demgegenüber problemlos.

falsch		richtig	
LDA	#$A0	LDA	#$A0
JSR	$FDED	JSR	$FDED
RTS		JMP	$3D0

Das falsche Programm würde — mit BRUN gestartet — nach dem RTS „hängen".

3.17. INIT-Befehl entfernen

Der INIT-Befehl läßt sich lahmlegen durch Modifizierung des Wortes „INIT"

A884: 09 0E 09 EA

Damit wird gleichzeitig der Speicherraum $B600-$B6FF für eigene kurze Assemblerprogramme frei.

3.18. Booten ohne DOS

Wenn eine Controller-Karte in einem Slot steckt, versucht eine Monitorroutine ($FAA6) nach dem Einschalten des Apples oder nach Ctrl-Offener Apfel-Reset den Bootvorgang einzuleiten. Erneutes Reset verhindert dies, so daß man sich danach im DOS-losen Applesoft befindet.

Normales Booten mit PR#6 verändert übrigens nicht den Speicherbereich $0900 - $95FF, wohl aber $0300 - $03FF und $0800 - $08FF.

3.19. Freie Zero-Page-Speicherstellen

Weniger wichtiger als die Frage, welche Speicherstellen im einzelnen von DOS in der Zero-Page (Nullseite) benutzt werden, ist die Frage, welche Zero-Page-Adressen unter Autostart-Monitor (oder Apple IIe Monitor), Applesoft und DOS 3.3 (aber nicht unter Integer-Basic) insgesamt noch frei sind. Es sind dies die unbenutzten Stellen:

$06-$07	(6-7)	
$08	(8)	nur bei II Plus, nicht bei IIe frei!
$09	(9)	
$19-$1E	(25-30)	
$1F	(31)	nur bei II Plus, nicht bei IIe frei!
$CE-$CF	(206-207)	
$D7	(215)	
$E3	(227)	
$EB-$EF	(235-239)	
$FA-$FE	(250-254)	

Die Stellen werden auch während des Bootens nicht verändert.

3.20. Versteckte Bildschirm-Speicherstellen

Wenn sich z.B. der Laufwerk-Controller in Slot 6 befindet, dann werden vom DOS bestimmte, nicht-sichtbare Bildschirm-Speicherstellen teilweise benutzt, z.B. zum „Verbuchen" der zuletzt benutzten Spur. Das Löschen dieser Speicherstellen kann zum I/O-Fehler oder zur Zwangs-Rekalibrierung des Lesekopfes führen.

$047E (Drive 1, letzter Track mal 2, z.B. $ 22 nach Catalog)
$04FE (Drive 2, letzter Track mal 2)
$05F0 (Boot-Slot mal 16, z.B. $60)
$0670 (Aktiver Slot mal 16)

3.21. Freie Stellen im DOS $9D00-$BFFF

$AAB8-$AAC0 (bei II Plus/IIe überflüssig)
$B5FE-$B5FF
$B6B3-$B6FC (alter APPEND-Patch)
$B78D-$B792
$B7F9-$B7FA
$B7FF
$BA69-$BA95
$BCDF-$BCFF
$BFD9-$BFDB

Wenn man feststellen will, ob ein bestimmtes DOS gepatcht ist, dann schaue man am besten zunächst bei diesen Stellen nach. Das mit dem Apple IIe gelieferte DOS 3.3 ist seinerseits an folgenden Stellen gepatcht:

$BA69-$BA75 (APPEND-Patch)
$BA76-$BA81 (stellt 80-Zeichen-Karte bei Reset korrekt ab)
$BA84-$BA93 (Stack retten)
$B300-$B35E (POSITION-Patch)

3.22. Ein- und Ausschalten des Motors

Das Ein- und Ausschalten des Laufwerkmotors ist als Peek-Befehl zulässig. Für Slot 6 gilt:

$C0E8 X = PEEK (-16152) : aus
$C0E9 X = PEEK (-16151) : an

Durch Einschalten des Motors vor dem Diskettenzugriff kann man u.U. geringfügig Zeit gewinnen. Man beachte jedoch, daß nicht zwei Laufwerke gleichzeitig eingeschaltet sein dürfen.
Darüber hinaus kann die Warteschleife, die DOS durchläuft, damit die optimale Rotationsgeschwindigkeit erreicht wird, verkürzt werden. Normalerweise gilt 1 Sekunde, doch vertragen viele Laufwerke auch 0,5 Sekunden:

POKE 47102, 216 für ca. 1 Sekunde
POKE 47102, 236 für ca. 0,5 Sekunden

3.23. SAVE ohne VERIFY

Nach jedem SAVE, BSAVE usw. führt DOS ein automatisches VERIFY durch, das Zeit kostet. Mit POKE 46732, 76 kann man dies abstellen (und mit POKE 46732, 32 wieder herstellen).

3.24. BLOAD von über 32K langen Files

Normalerweise ist die Länge eines Binärfiles auf knapp 32K (= 32767 Bytes) begrenzt. Nach einem Patch (A964: FF) kann man Binärfiles mit mehr als 32K Länge einlesen.

3.25. Verschieben des Controller-Boot-Programms

Das sich z.B. bei Slot 6 im Bereich $C600-$C6FF befindliche Boot-Programm läßt sich in einen anderen $X600-Bereich verschieben (z.B. $1600) und damit abändern. Beispielsweise lädt

```
1600<C600.C6FFM
16F8:0
1600G
```

den Sektor 0 von Track 0 von jeder (kopiergeschützten) Diskette in den Bereich $800-$8FF. Eine etwa radikalere Methode bei geschützen Programmdisketten ist die, daß man die Laufwerkklappe kurz nach dem Bootvorgang öffnet. In diesem Stadium können viele geschützte Programme den I/O-Fehler noch nicht abfangen.

3.26. Inverse Dateinamen

Bisweilen möchte man einen Catalog durch inverse File-Namen verschönern. Angenommen, der Name des Hello-Programms „MEINE PROGRAMME 1" soll invers sein. Zu diesem Zweck verfahre man wie folgt. Zunächst tippt man

INVERSE : PRINT „MEINE PROGRAMME 1" : NORMAL

Nun tippt man SAVE und fährt dann mit dem Cursor über die durch den obigen Befehl entstandene inverse Zeile am Bildschirm.
Man beachte, daß 80-Zeichen-Karten (z.B. diejenige für den Apple IIe) inverse Catalog-Dateinamen oft als Steuerzeichen interpretieren.

3.27. Kopierschutz

Wenn man eine Diskette initialisiert, nachdem man z.B. die folgenden Pokes ausgeführt hat, dann läßt sich diese Diskette nicht mehr mit COPYA und anderen normalen Kopierprogrammen kopieren. Vielmehr müssen dann schon Nibble-Kopierer wie „Locksmith" usw. herhalten. Bei den als Beispiele aufgeführten Pokes handelt es sich um die Änderung der Erkennungs-Bytes, die der Lesekopf sucht, wenn er einen bestimmten Sektor einlesen will. (Diese Bytes gehören nicht zu den eigentlichen 256 Daten-Bytes des Sektors.)

POKE -18031, 223 (normalerweise 222)
POKE -17234, 223 (normalerweise 222)

3.28. „Loch"-Kopierschutzverfahren

Das Wettrennen zwischen Kopierschützern und Programmknackern geht weiter. Nachdem die Firma Omega zunächst ihren Locksmith 4.0 bzw. 4.1 einige Zeit aus dem Verkehr gezogen hatte — man munkelt, daß die Software-Industrie zur „Beschleunigung dieser Entscheidung finanzielle Hilfen" angeboten hat -, erschien Anfang 1984 Locksmith 5.0, das so ziemlich alles „knackt", was vor diesem Zeitpunkt erschienen ist. Um zu zeigen, welche skurrilen Blüten das Kopierschützen treibt, sei auf die völlig unseriöse und z.B. von Locksmith 5.0 nicht „knackbare" Lochmethode hingewiesen, deren Prinzip — stark vereinfacht — kurz geschildert werden soll:

1. Man zerstört auf einer mit traditionellen, sprich „primitiven" Schutzverfahren (siehe 3.27) geschützten Diskette mit einer feinen Nadel mehrere Sektoren.

2. Das binäre Hello-Programm enthält eine — zweckmäßigerweise verschlüsselte — spezielle RWTS-Routine, die diese beschädigten Sektoren zunächst beschreibt und

dann wieder einliest. Wenn die Prüfsumme nicht stimmt, ist es die Originaldiskette, ansonsten eine Schwarzkopie.

Das Schutzverfahren beruht also darauf, daß bei einer physisch beschädigten Diskette die RWTS beim Beschreiben eines Sektors nicht merkt, daß die Bits „durch die Löcher rutschen" und damit keine Fehlermeldung hervorruft. Umgekehrt merkt DOS jedoch sehr wohl beim Einlesen, daß die Sektor-Daten nicht mehr vollständig sind. Aus der Sicht des unseriösen Schützers besteht das Problem darin, die Löcher, Kratzer o.ä. — raffinierte Techniker empfehlen Laserstrahlen — auf den von dem eigentlichen Programm unbenutzten Spuren anzubringen, denn sonst würde das Programm selbstverständlich selbst nicht mehr laufen. Aus der Sicht des seriösen Programmkäufers besteht das Problem darin, stets einen Ersatz-Lesekopf bereitzuhalten, da sich der Lesekopf natürlich stärker abnutzt, wenn mit einer „Lochzange" traktierte Disketten gelesen werden müssen. Schließlich stellt sich aus der Sicht des unseriösen Knackers das Problem, wie er die Löcher auf der Schwarzkopie an denselben Stellen anbringt wie auf dem Original.
Das Lochverfahren wird inzwischen von mehreren Software-Firmen angewandt. Sicherheitshalber sollte man sich deshalb beim Software-Kauf die Zusicherung geben lassen, daß die Programmdiskette physisch nicht beschädigt ist.

3.29. RWTS bei Diversi-DOS usw. schneller?

Viele denken, daß bei DOS-Varianten wie Superdos usw. die sog. RWTS schneller sei. Dies ist ein Denkfehler! DOS-Varianten verbessern im wesentlichen nur den sog. File-Manager, der bei DOS 3.3 zu langsam ist. Beispielweise ist bei Diversi-DOS die RWTS überhaupt nicht geändert. Lediglich die Zero-Page-Adresse $48 (Monitor Processor Status Register) wird teils anders benutzt, um Konflikte mit dem Dezimalmodus auszuschließen.

3.30. Freie Sektoren auf der Diskette

Das folgende kleine Programm zeigt die Anzahl der freien Sektoren auf einer Diskette als Dezimalzahl an. Vor Aufruf dieses Programm muß auf die Diskette mit CATALOG o.ä. zugegriffen worden sein.

```
:ASM
                    1               ORG  $300
                    2     *
                    3     * FREIE SEKTOREN AUF DISKETTE
                    4     *
0300: A9 00         5               LDA  #$00
0302: 8D 2B 03      6               STA  H1
0305: 8D 2C 03      7               STA  H2
0308: A0 C8         8               LDY  #$C8
030A: 18            9     L1        CLC
030B: B9 F2 B3      10              LDA  $B3F2,Y      ;VTOC
030E: F0 0E         11    L2        BEQ  L4
0310: 0A            12              ASL
0311: 90 FB         13              BCC  L2
0313: EE 2B 03      14              INC  H1
0316: D0 03         15              BNE  L3
0318: EE 2C 03      16              INC  H2
031B: 18            17    L3        CLC
031C: 90 F0         18              BCC  L2
031E: 88            19    L4        DEY
031F: D0 E9         20              BNE  L1
0321: AE 2B 03      21              LDX  H1
0324: AD 2C 03      22              LDA  H2
0327: 20 24 ED      23              JSR  $ED24        ;LINPRT
032A: 60            24              RTS
032B: 00            25    H1        HEX  00
032C: 00            26    H2        HEX  00

--End assembly--

45 bytes

Errors: 0
```

Anmerkung

$B3BB – $B4BA = VTOC-Puffer
$B3F3 – $B4BA = eigentliches VTOC
(vgl. S. 167, Zeilen 217 - 266)

4. GETLN, RDKEY und COUT: Input-Output-Vektoren

GETLN = GETLINE = JSR $FD6A = entspricht INPUT in Basic
RDKEY = READKEY = JSR $FD0C = entspricht GET in Basic
COUT = CHR OUT = JSR $FDED = entspricht PRINT CHR$(X) in Basic

Assembler-Programmierer können anstelle der Basic-Befehle auf Textfiles mit den analogen Monitor-Befehlen zugreifen, z.B.:

```
10 PRINT CH$(4) ,,OPEN TEST" : PRINT CHR $(4) ,,READ TEST"
20 CALL 768 : PRINT CHR$(4) ,,CLOSE"
```

wobei bei 768 = $0300 steht: 20 6A FD 60 = JSR $FD6A RTS

Im einzelnen gilt:

GETLN legt den eingelesenen Textfile-String im Eingabe-Puffer ab $0200 (mit Bit 7 on) einschließlich Return ab, z.B. Feld ,,aaaaa":

```
$0200: E1 E1 E1 E1 E1 8D
```

Das X-Register enthält nach GETLN die Länge des Strings. Im Gegensatz zum Applesoft-INPUT kann der String bis zu 255 Zeichen lang sein. GETLN ist außerdem schneller als INPUT. (Ein GETLN-Beispiel ist in Teil I, Kapitel 2.6.7.2 gelistet.)

RDKEY lädt den Akkumulator mit dem eingelesenen Zeichen. RDKEY ist zwar schneller als GET, jedoch immer noch relativ langsam.

RDKEY verändert das Y-Register (wegen FD0C: LDY $24) und GETLN verändert zusätzlich das X-Register, so daß man die Index-Register selbst ,,retten" muß.

COUT speichert das sich im Akkumulator befindliche Zeichen auf der Diskette. COUT ist schneller als PRINT ,,STRING" bzw. PRINT CHR$(X). COUT verändert nicht die A-, X- und Y-Register, also auch nicht den Akkumulator, so daß LDA # $A0 JSR COUT JSR COUT möglich ist. Mit COUT können auch die normalen DOS-Befehle wie CATALOG usw. an den DOS-Interpreter geschickt werden. Man beachte dabei, daß davor 8D 84 (Return + Ctrl-D) und danach 8D übertragen werden müssen.

Mit dem folgenden nützlichen Mini-Programm kann man über RDKEY jeden beliebigen sequentiellen Textfile einlesen und am Bildschirm anzeigen:

```
10 ONERR GOTO 30
20 OPEN CHR$(4) ,,OPEN TEXT" : PRINT CHR$(4) ,,READ TEXT" : CALL 768
30 END
```

Das dazugehörige Assemblerprogramm sieht so aus:

```
0300:   20   0C   FD   JSR   $FD0C   RDKEY
0303:   20   F0   FD   JSR   $FDF0   COUT1
0306:   4C   00   03   JMP   $0300
```

4.1. Fusion: Runtime + Tasc.Obj

Das Programm ,,FUSION: RUNTIME + TASC.OBJ", das einen TASC-compilierten Object-Code namens ALT mit der Runtime-Library zu einem Gesamtfile namens NEU vereint, zeigt als praktisches Beispiel die Anwendung von DOS-Befehlen in Assemblerprogrammen. Dieses Programm wird mit BRUN gestartet und setzt 48K DOS voraus.

```
:ASM
                1               ORG  $300
                2    *
                3    * FUSION:RUNTIME+TASC.OBJ
                4    * ------------------------
                5    *
                6    * COMPILIERT MIT ORG $0806
                7    * DAMIT RUNTIME ORG  $17B3
                8    *
                9    *
               10    DOSCOLD  EQU  $3D3
```

```
                        11   HOME     EQU  $FC58
                        12   GET      EQU  $FD35
                        13   HEXBYTE  EQU  $FDDA
                        14   PRINT    EQU  $FDED
                        15   RETURN   EQU  $FD8E
                        16   *
0300: 20 58 FC          17            JSR  HOME
0303: A2 00             18            LDX  #$0
0305: BD 6F 03          19   MENU     LDA  HINWEIS,X
0308: 20 ED FD          20            JSR  PRINT
030B: E8                21            INX
030C: E0 19             22            CPX  #$19
030E: D0 F5             23            BNE  MENU
0310: 20 35 FD          24            JSR  GET
0313: 20 58 FC          25            JSR  HOME
0316: A2 00             26            LDX  #$0
0318: BD 88 03          27   LOAD     LDA  FILES,X
031B: CD B6 03          28            CMP  MARKE
031E: F0 07             29            BEQ  LAENGE
0320: 20 ED FD          30            JSR  PRINT
0323: E8                31            INX
0324: 4C 18 03          32            JMP  LOAD
0327: D8                33   LAENGE   CLD
0328: 18                34            CLC
0329: AD 60 AA          35            LDA  $AA60      ;LEN-LOW
032C: 6D CB 03          36            ADC  RUNLOW
032F: 8D CA 03          37            STA  LOW
0332: AD 61 AA          38            LDA  $AA61      ;LEN-HIGH
0335: 6D CC 03          39            ADC  RUNHIGH
0338: 8D C9 03          40            STA  HIGH
                        41   *
                        42   * JMP VOR RUNTIME
                        43   *
033B: A9 4C             44            LDA  #$4C       ;JMP
033D: 8D 03 08          45            STA  $803       ;$17B3
0340: A9 B3             46            LDA  #$B3
0342: 8D 04 08          47            STA  $804
0345: A9 17             48            LDA  #$17
0347: 8D 05 08          49            STA  $805
034A: A2 00             50            LDX  #$0
034C: BD B7 03          51   SAVE     LDA  NEU,X
034F: 20 ED FD          52            JSR  PRINT
0352: E8                53            INX
0353: E0 12             54            CPX  #$12
0355: D0 F5             55            BNE  SAVE
0357: AD C9 03          56            LDA  HIGH
035A: 20 DA FD          57            JSR  HEXBYTE
035D: AD CA 03          58            LDA  LOW
0360: 20 DA FD          59            JSR  HEXBYTE
0363: 20 8E FD          60            JSR  RETURN
0366: AD 88 03          61            LDA  FILES      ;CRTL-D
0369: 20 ED FD          62            JSR  PRINT
036C: 4C D3 03          63            JMP  DOSCOLD
036F: D2 D5 CE          64   HINWEIS  ASC  "RUNTIME + ALT "
```

```
0372: D4 C9 CD C5 A0 AB A0 C1
037A: CC D4 A0
037D: C9 CE A0  65              ASC  "IN DRIVE 1 "
0380: C4 D2 C9 D6 C5 A0 B1 A0
0388: 84        66    FILES     HEX  84          ;CRTL-D
0389: 8D 8D     67              HEX  8D8D
038B: CD CF CE  68              ASC  "MON,I,O,C"
038E: AC C9 AC CF AC C3
0394: 8D 84     69              HEX  8D84
0396: C2 CC CF  70              ASC  "BLOADRUNTIME"
0399: C1 C4 D2 D5 CE D4 C9 CD
03A1: C5
03A2: AC C1 A4  71              ASC  ",A$806,D1"
03A5: B8 B0 B6 AC C4 B1
03AB: 8D 84     72              HEX  8D84
03AD: C2 CC CF  73              ASC  "BLOADALT"
03B0: C1 C4 C1 CC D4
03B5: 8D        74              HEX  8D
03B6: FF        75    MARKE     HEX  FF
                76    *
03B7: 84        77    NEU       HEX  84
03B8: C2 D3 C1  78              ASC  "BSAVENEU"
03BB: D6 C5 CE C5 D5
03C0: AC C1 A4  79              ASC  ",A$803,L$"
03C3: B8 B0 B3 AC CC A4
                80    *
                81    *NACH L$ HIGHBYTE FIRST
                82    *
03C9: 00        83    HIGH      HEX  00
03CA: 00        84    LOW       HEX  00
03CB: B2        85    RUNLOW    HEX  B2
03CC: 0F        86    RUNHIGH   HEX  0F
```

Man beachte, daß man beim TASC auf ,,MEMORY USAGE DEFAULT" mit ,,N" und dann auf ,,ADDRESS FOR LIBRARY" mit ,,2054" antworten muß.

4.2. CPM-Refiner für Wordstar-Dateien

Das Programm ,,CPM-REFINER", das der ASCII-Code-Verfeinerung von mit dem CPMXFER-Programm von Microsoft auf DOS 3.3 roh-umgewandelten Wordstar-Dateien usw. dient, zeigt die Anwendung von COUT bei assemblermäßig gespeicherten Textfiles. Dieses Programm wird aus dem übergeordneten Applesoft-Programm heraus gestartet.

```
100  REM *** CPM-REFINER ***
110  PRINT : PRINT  CHR$ (4)"BLOAD CPM-REFINER.OBJ,A$1000":
     ONERR  GOTO 220
120  CLEAR : HIMEM: 4096: TEXT : HOME : INVERSE : PRINT "*
     U.STIEHL 1/84"
130  PRINT "CPMXFER-REFINER": NORMAL
140  PRINT : PRINT "DISK EINLEGEN"
150  PRINT : PRINT "START J/N ";
160  GET X$: ON X$ <  > "J" AND X$ <  > "j" AND X$ <  > "N" AND
     X$ <  > "n" GOTO 160: IF X$ = "N" OR X$ = "n" THEN  HOME :
     HIMEM: 38400: POKE 216,0: END
170  PRINT : HOME : PRINT  CHR$ (4)"CATALOG": INVERSE : PRINT
     "ROHDATEI:";: NORMAL : INPUT " ";R$
180  ON R$ = "" GOTO 120: HOME : PRINT R$;" => C";R$
190  PRINT  CHR$ (4)"UNLOCK"R$: PRINT  CHR$
     (4)"BLOAD"R$",A$1100"
200  PRINT  CHR$ (4)"OPEN C"R$: PRINT  CHR$ (4)"WRITE C"R$: CALL
     4096: PRINT : PRINT  CHR$ (4)"CLOSE": PRINT  CHR$
     (4)"CATALOG"
210 X =  PEEK (49168): PRINT "ERLEDIGT ";: GET X$: GOTO 120
220  HOME : PRINT "ERROR # "; PEEK (222): GET X$: GOTO 120
```

```
:ASM
                        1               ORG   $1000
                        2    *
                        3    * CPM-REFINER.OBJ/23.11.83/STIEHL
                        4    *
                        5    IND        EQU   $CE
                        6    FILE       EQU   $1104       ;BEG/LEN
                        7    PRINT      EQU   $FDED
                        8    *
1000: A9 04             9               LDA   #<FILE
1002: 85 CE             10              STA   IND
1004: A9 11             11              LDA   #>FILE
1006: 85 CF             12              STA   IND+1
1008: A9 00             13              LDA   #0
100A: 8D FF 95          14              STA   $95FF       ;ENDMARK
100D: A9 01             15              LDA   #1
100F: 8D 00 01          16              STA   $100
1012: 8D 01 01          17              STA   $101
                        18   *
1015: A0 00             19   LOOP1      LDY   #0
1017: B1 CE             20              LDA   (IND),Y
1019: C9 00             21              CMP   #$00        ;ENDMARK
101B: F0 2B             22              BEQ   ENDE
101D: C9 1A             23              CMP   #$1A        ;CPM-Z
101F: F0 27             24              BEQ   ENDE
                        25   *
                        26   * CTRL-D MIT BIT 7 OFF
                        27   *
1021: C9 84             28              CMP   #$84        ;CTRL-D
```

```
1023: D0 24      29            BNE  SKIP
1025: A9 04      30            LDA  #$04        ;NO-DOS
1027: D0 02      31            BNE  PRINTER2
                 32   *
                 33   * BIT 7 SETZEN
                 34   *
1029: 09 80      35   PRINTER1 ORA  #$80        ;BIT 7
102B: AE 00 01   36   PRINTER2 LDX  $100
102E: 8E 01 01   37            STX  $101
1031: 8D 00 01   38            STA  $100
1034: EC 00 01   39            CPX  $100
1037: D0 04      40            BNE  PRINTER3
1039: E0 A0      41            CPX  #$A0
103B: F0 03      42            BEQ  INCREM
103D: 20 ED FD   43   PRINTER3 JSR  PRINT
                 44   *
1040: E6 CE      45   INCREM   INC  IND         ;INCREM.
1042: D0 D1      46            BNE  LOOP1
1044: E6 CF      47            INC  IND+1
1046: D0 CD      48            BNE  LOOP1
1048: 60         49   ENDE     RTS
                 50   *
                 51   * SKIP
                 52   *
1049: C9 1E      53   SKIP     CMP  #$1E        ;DIVIS^
104B: F0 F3      54            BEQ  INCREM
104D: C9 0A      55            CMP  #$0A        ;LF
104F: F0 EF      56            BEQ  INCREM
1051: C9 8A      57            CMP  #$8A        ;LF?
1053: F0 EB      58            BEQ  INCREM
                 59   *
                 60   * CONVERT
                 61   *
1055: C9 A0      62            CMP  #$A0
1057: F0 32      63            BEQ  CONV4
1059: C9 20      64            CMP  #$20
105B: F0 20      65            BEQ  CONV3
105D: C9 1F      66            CMP  #$1F
105F: F0 0E      67            BEQ  CONV2
1061: C9 8D      68            CMP  #$8D
1063: F0 06      69            BEQ  CONV1
1065: C9 0D      70            CMP  #$0D        ;HARD-CR
1067: F0 C0      71            BEQ  PRINTER1
1069: D0 BE      72            BNE  PRINTER1
                 73   * 8D SOWIE FERNER 20 ODER A0
106B: A9 A0      74   CONV1    LDA  #$A0
106D: D0 BC      75            BNE  PRINTER2
                 76   * 1F
106F: A0 01      77   CONV2    LDY  #1
1071: B1 CE      78            LDA  (IND),Y
1073: C9 8D      79            CMP  #$8D
1075: D0 C9      80            BNE  INCREM
1077: A9 0A      81            LDA  #$0A        ;NOP
1079: 91 CE      82            STA  (IND),Y
```

```
107B: D0 C3     83                BNE   INCREM
                84    * 20
107D: A0 01     85    CONV3       LDY   #1
107F: B1 CE     86                LDA   (IND),Y
1081: C9 20     87                CMP   #$20
1083: F0 BB     88                BEQ   INCREM
1085: C9 A0     89                CMP   #$A0
1087: F0 B7     90                BEQ   INCREM
1089: D0 E0     91                BNE   CONV1
                92    * A0
108B: A0 01     93    CONV4       LDY   #1
108D: B1 CE     94                LDA   (IND),Y
108F: C9 20     95                CMP   #$20
1091: F0 AD     96                BEQ   INCREM
1093: C9 A0     97                CMP   #$A0
1095: F0 A9     98                BEQ   INCREM
1097: D0 D2     99                BNE   CONV1

--End assembly--

153 bytes

Errors: 0
```

4.3. DOS-Output-Vektor-Änderung

Betrachtet man sich die Speicherstelle von COUT

```
FDED:  6C   36   00   JMP   ($0036)   = JMP (CSWL)
FDF0:  C9   A0        CMP   # $A0     = COUT1
```

dann sieht man, daß COUT über die Zero-Page-Vektor-Adresse bei $0036-$0037 zur eigentlichen Character-Output-Routine springt. Diese wäre ohne DOS, ohne Drucker und ohne weitere Peripheriegeräte $FDF0, d.h. die Bildschirmausgabe.

Betrachtet man sich ferner die Speicherstelle von RDKEY

```
FD0C:  A4   24        LDY  $ 24
       usw.
FD18:  6C   38   00   JMP  ($0038)   = JMP KSWL)
FD1B:  A0   06        LDY  # $06     (bei Apple IIe!) = KEYIN
FD1B:  E6   4E        INC  $4E       (bei Apple II Plus!) = KEYIN
```

dann sieht man, daß RDKEY nach einer kurzen Cursor-Routine über die Zero-Page-Vektor-Adresse bei $0038-$0039 zur eigentlichen Keyboard-Input-Routine springt. Diese wäre ohne DOS und ohne andere Peripheriegeräte $FD1B, also die Tastatureingabe.

Mit DOS werden diese Vektoren auf Routinen im DOS abgeändert, so daß DOS prüfen kann, ob Ctrl-D + DOS-Befehl eingegeben wurde oder ob gerade Daten von einem Textfile gelesen oder auf einen Textfile gespeichert werden. Wenn dem nicht so ist, springt DOS zu der eigentlichen Input- oder Output-Routine, deren Adressen im DOS wie folgt abgelegt sind:

AA53: Low Byte + High Byte der echten Output-Routine
AA55: Low Byte + High Byte der echten Input-Routine

Ein Anwenderprogramm kann das gleiche tun wie DOS, nämlich sich durch Vektoränderung bei den Eingabe-Ausgabe-Routinen zwischenschalten. Dies soll am Beispielprogramm „DOS-Output-Vektor-Änderung" demonstriert werden.
Hier dient die Vektoränderung dem Zweck, Ctrl-Zeichen in druckbare Zeichen umzuwandeln und für Programmlistings den linken Rand sowie den Blattvorschub einzustellen. Das Beispielprogramm wird mit BRUN gestartet. Die Vektoren können durch PR#0 : IN#0 : CALL 1002 wieder normalisiert werden.

```
:ASM
                        1               ORG  $0300
                        2   *===================================
                        3   *
                        4   * DOS-OUTPUT-VEKTOR-AENDERUNG
                        5   *
                        6   * ZWECK:
                        7   *
                        8   * A) UMWANDLUNG VON CTRL-ZEICHEN
                        9   *    IN DRUCKBARE ZEICHEN
                       10   *
                       11   * B) LINKER RAND BEI LISTINGS
                       12   *
                       13   * C) FORMFEED BEI LISTINGS
                       14   *
                       15   *===================================
                       16   *
                       17   * VOR BRUN DRUCKER EINSCHALTEN,
                       18   * FALLS OUTPUT-ADRESSE $C102
                       19   *
                       20   * VEKTOR NORMALISIEREN MIT PR#0,
                       21   * BEI $C307 MIT RESET
                       22   *
0300: 4C 09 03         23         JMP  BEGINN
                       24   *
                       25   *== HIER PARAMETER POKEN!!! ===*
                       26   *
                       27   *--------OUTPUT-ADRESSE----------
                       28   *
                       29   * FUER BILDSCHIRM: $FDF0  SLOT:0
                       30   * FUER DRUCKER   : $C102  SLOT:1
                       31   * FUER 80 Z-KARTE: $C307  SLOT:3
                       32   *
0303: 20 F0 FD         33   DRUCK    JSR  $FDF0      ;SCREEN
```

```
0306: 60          34            RTS
                  35    *--------LINKER RAND:0-255-------
                  36    *
                  37    * 0 = KEIN RAND DURCH PROGRAMM
                  38    *
0307: 03          39    BREITE    DFB  03          ;0-255
                  40    *
                  41    *--------ZEILEN/SEITE:0-255------
                  42    *
                  43    * 0 = KEIN FORMFEED DURCH PROGRAM
                  44    *
0308: 3C          45    TIEFE     DFB  60          ;0-255
                  46    *
                  47    *--------VEKTOR INITIALISIEREN--
0309: A9 1B       48    BEGINN    LDA  #<VEKTOR
030B: 8D 53 AA    49              STA  $AA53       ;DOSCSWL
030E: A9 03       50              LDA  #>VEKTOR
0310: 8D 54 AA    51              STA  $AA54       ;DOSCSWH
0313: A9 00       52              LDA  #0
0315: 8D 4B 03    53              STA  FFCOUNT
0318: 4C D0 03    54              JMP  $3D0        ;DOSWARM
                  55    *--------PRINT-ROUTINE----------
031B: C9 8D       56    VEKTOR    CMP  #$8D        ;RETURN
031D: D0 2D       57              BNE  CTRL
031F: 20 03 03    58              JSR  DRUCK
                  59    *--------TIEFE------------------
0322: EE 4B 03    60              INC  FFCOUNT
0325: AD 08 03    61              LDA  TIEFE
0328: F0 0F       62              BEQ  RAND0       ;KEIN FF
032A: CD 4B 03    63              CMP  FFCOUNT
032D: B0 0A       64              BCS  RAND0
032F: A9 8C       65              LDA  #$8C        ;FF
0331: 20 03 03    66              JSR  DRUCK
0334: A9 00       67              LDA  #0
0336: 8D 4B 03    68              STA  FFCOUNT
                  69    *--------BREITE------------------
0339: 8A          70    RAND0     TXA
033A: 48          71              PHA
033B: AE 07 03    72              LDX  BREITE
033E: F0 08       73              BEQ  RAND2       ;KEIN R.
0340: A9 A0       74    RAND1     LDA  #$A0
0342: 20 03 03    75              JSR  DRUCK
0345: CA          76              DEX
0346: D0 F8       77              BNE  RAND1
0348: 68          78    RAND2     PLA
0349: AA          79              TAX
034A: 60          80              RTS
                  81    *--------FORMFEED-COUNTER-------
034B: 00          82    FFCOUNT   HEX  00
                  83    *--------CTRL=>NORMAL ZEICHEN---
034C: C9 20       84    CTRL      CMP  #$20
034E: 90 13       85              BCC  CTRL1
0350: C9 40       86              CMP  #$40
0352: 90 15       87              BCC  CTRL2
0354: C9 60       88              CMP  #$60
0356: 90 17       89              BCC  CTRL3
0358: C9 80       90              CMP  #$80
035A: 90 19       91              BCC  CTRL4
035C: C9 A0       92              CMP  #$A0
035E: 90 1B       93              BCC  CTRL5
0360: 4C 03 03    94              JMP  DRUCK
0363: 18          95    CTRL1     CLC              ;$00-$1F
0364: 69 C0       96              ADC  #$C0
0366: 4C 03 03    97              JMP  DRUCK
0369: 18          98    CTRL2     CLC              ;$20-$3F
036A: 69 80       99              ADC  #$80
036C: 4C 03 03    100             JMP  DRUCK
036F: 18          101   CTRL3     CLC              ;$40-$5F
0370: 69 80       102             ADC  #$80
0372: 4C 03 03    103             JMP  DRUCK
0375: 18          104   CTRL4     CLC              ;$60-$7F
0376: 69 40       105             ADC  #$40
0378: 4C 03 03    106             JMP  DRUCK
037B: 18          107   CTRL5     CLC              ;$80-$9F
037C: 69 40       108             ADC  #$40
037E: 4C 03 03    109             JMP  DRUCK
```

5. DOS-Vektoren ab $03D0

Für einige wichtige Systemadressen im DOS wird nach dem Booten ab $03D0 leicht zugänglich eine Vektor-Tabelle eingerichtet:

03D0:	4C BF 9D	= JMP $9DBF	= DOS-Warmstart-Adresse (entspricht Basic-END)
03D3:	4C 84 9D	= JMP $9D84	= DOS-Kaltstart-Adresse (entspricht FP)
03D6:	4C FD AA	= JMP $AAFD	= File-Manager-Adresse
03D9:	4C B5 B7	= JMP $B7B5	= RWTS-Adresse
03DC:	AD OF 9D	= LDA $9D0F	= A = High Byte der File-Manager-Blocks
03DF:	AC 0E 9D	= LDY $9D0E	= Y = Low Byte der File-Manager-Blocks
03E2:	60	= RTS	
03E3:	AD C2 AA	= LDA $AAC2	= A = High Byte des RWTS-Blocks
03E6:	AD C1 AA	= LDY $AAC1	= Y = Low Byte des RWTS-Blocks
03E9:	60	= RTS	
03EA:	4C 51 A8	= JMP $A851	= Reconnect DOS-Input-Output (CALL 1002)
03ED:	EA EA	= NOP NOP	= Apple IIe: XFER-Jump-Adresse
03EF:	4C 59 FA	= JMP $FA59	= Break-Adresse (Monitor)
03F2:	BF 9D	= $9DBF	= Reset-Adresse
03F4:	38	= HEX 38	= Reset-Test-Byte (LDA $3F3 EOR # $A5 STA $3F4)
03F5:	4C 58 FF	= JMP $FF58	= Ampersand-Adresse (Applesoft)
03F8:	4C 65 FF	= JMP $FF65	= Ctrl-Y-Adresse (Monitor)
03FB:	4C 65 FF	= JMP $FF65	= Nicht-maskierbarer Interrupt (Monitor)
03FE:	65 FF	= $FF65	= Maskierbarer Interrupt (Monitor)

Die Werte sind so angegeben, wie man sie nach dem Booten einer normalen DOS-Diskette vorfindet. Wenn z.B. DOS in die Language Card geschoben wurde, müssen von dem DOS-Mover-Programm die Vektoren entsprechend geändert werden.

Beispiel für DOS-Mover (C. Bongers):

```
03D0-   4C AE BF    JMP   $BFAE
03D3-   4C B4 BF    JMP   $BFB4
03D6-   4C BA BF    JMP   $BFBA
03D9-   4C C3 BF    JMP   $BFC3
03DC-   A9 BF       LDA   #$BF
03DE-   A0 E6       LDY   #$E6
03E0-   EA          NOP
03E1-   EA          NOP
03E2-   60          RTS
03E3-   A9 BF       LDA   #$BF
03E5-   A0 D5       LDY   #$D5
03E7-   EA          NOP
03E8-   EA          NOP
03E9-   60          RTS
03EA-   4C CC BF    JMP   $BFCC
03ED-   EA          NOP
03EE-   EA          NOP
03EF-   4C 59 FA    JMP   $FA59
03F2-   AE BF 1A    LDX   $1ABF
03F5-   4C 58 FF    JMP   $FF58
03F8-   4C 65 FF    JMP   $FF65
03FB-   4C 65 FF    JMP   $FF65
03FE-   65 FF       ADC   $FF
```

6. RWTS (Read-Write-Track-Sector)

DOS läßt sich in drei Teile gliedern:

1. Der sog. Command Interpreter dient der Analyse der Befehle. Hier wird z.B. das Ctrl-D abgefangen und etwa bei dem Befehl „BLOAD FILE,A 10000" die Zahl in hexadezimal umgewandelt usw.

2. Der sog. File Manager überwacht das Einlesen und Speichern von Diskettensektoren. Er kopiert z.B. den in den DOS-Puffer eingeladenen Sektor in den RAM-Speicher usw.

3. Die sog. RWTS dient der eigentlichen Datenübertragung zwischen Diskette und RAM über den DOS-Controller.

In diesem Buch wird auf den File Manager nicht näher eingegangen, weil der assemblermäßige Zugriff auf diesen Teil des DOS kaum Vorteile bringt. Um dies näher zu belegen, wurde ein 122 Sektoren langer Binärfile insgesamt 100 mal auf verschiedene Weise von einer Apple IIe 64K Karte, die als RAM Disk benutzt wurde, eingelesen. Dies entspricht einer Datenmenge von ca. 3 Megabytes.

Test 1a:

```
10 FOR X = 1 TO 100 : PRINT CHR$(4) „UNLOCK FILE" : NEXT : REM 5 SEK.
```

Test 1b:

```
10 FOR X = 1 TO 100 : CALL 36864 : NEXT : REM 55 SEK.
```

Anmerkung: Dieser Text 1b wurde mit dem Programm „FILE.LESER" durchgeführt (gelistet ab S. 133), nachdem dort mit
9088: 60

90A1: 60
die Lösch- und Print-Routinen inaktiviert worden waren. Der File-Leser ist eines der schnellsten RWTS-Einleseprogramme für Dateien aller Art und beliebiger Länge.

Test 2:
10 FOR X = 1 TO 100 : PRINT CHR$(4) ,,BLOAD FILE" : NEXT : REM 900 SEK.

Beim Test 1a wird mit dem UNLOCK-Befehl der VTOC-Sektor eingelesen, der erste Catalog-Sektor eingelesen und wieder zurückgeschrieben sowie der TSL-Sektor eingelesen, alles übrigens teils mehrfach, d.h. insgesamt ca. 7 RWTS-Durchläufe.

Beim Test 1b werden nur die 121 Datensektoren per RWTS eingelesen. Da jedoch eine RAM-Disk verwendet wurde, wird die RWTS vom RAM-Disk-Driver abgefangen, so daß das Übertragen eines Sektors stets die gleiche Zeit in Anspruch nimmt. Damit können hardwaremäßige RWTS-Besonderheiten wie Einschalten des Motors, optimale Rotationsgeschwindigkeit, Skewing usw. ignoriert werden.

Beim Test 2 wird exakt dieselbe Anzahl an Sektoren wie beim Test 1a + 1b zusammen eingelesen. Einzige Ausnahme: Test 1b speichert wegen des UNLOCK den Catalog-Sektor zurück.

Test 1a + 1b dauern zusammen knapp 60 Sekunden, Test 2 dagegen ca. 900 Sekunden. Da die Anzahl der RWTS-Operationen fast dieselbe ist (bei Test 2 sogar etwas geringer) und da der Command Interpreter für den UNLOCK-Befehl (einschließlich der zusätzlichen RWTS-Zugriffe!) nur 5 Sekunden benötigt, was bei der Befehlsinterpretierung des BLOAD-Befehls kaum mehr sein dürfte, benötigt der Test 2 somit:

ca. 5 Sekunden für den Command Interpreter
ca. 55 Sekunden für die RWTS
ca. 840 Sekunden für den File Manager

Daraus erhellt, daß der File Manager mit Abstand der langsamste Teil des DOS darstellt. Mit anderen Worten: Wenn man den File Manager umgeht und auf die Diskette direkt per RWTS zugreift, ist das Programm bis zu 15 mal schneller!

Die RWTS liegt beim 48K DOS im Bereich $B600-BFFF bzw. beim in die Language Card geschobenen DOS meist im Bereich $F600-FFFF. Der eigentliche RWTS-Entry liegt jedoch bei $BD00 (bzw. $FD00). Die RWTS läßt sich praktisch nur aus einem Assemblerprogramm heraus benutzten, doch sind Kenntnisse der internen RWTS-

Vorgänge nicht erforderlich. Man beachte, daß die RWTS nur für das Einlesen/Beschreiben GANZER 256-Byte-Sektoren bestimmt ist. Theoretisch kann man auf noch niedrigerer Ebene auf die Diskette zugreifen, doch liest man dann keine Bytes, sondern nur codierte Nibbles. Die RWTS ist extrem zeitkritisch, d.h. die Maschinenbefehle sind so geschrieben (und teils mit NOP 's aufgefüllt), daß eine ganz genaue Anzahl von Prozessor-Takten eingehalten wird. Aus diesem Grunde kann z.B. die mit 3,5 Megahertz getaktete 6502-Accelerator-Karte (von Titan und anderen Firmen) nicht den Zugriff auf physische Disketten (wohl aber auf RAM-Karten) beschleunigen.

Die Anwendung der RWTS ist im Prinzip kinderleicht, da man lediglich vor dem Aufruf dieser Routine einige Werte in einer ca. 20 Bytes langen Parameterliste eintragen muß. Diese Parameterliste gliedert sich in zwei Teile, die im Speicher nicht unbedingt aufeinander folgen müssen: IOB und DCT.

RWTS-Parameter-Tabelle

IOB = Input-Output-Block

Byte	00:	01:	Konstante, stets 01
Byte	01:	60:	Jetziger Slot (10-70); muß festgelegt werden
Byte	02:	01:	Jetziges Drive (01-02); muß festgelegt werden
Byte	03:	00:	jetzige Volume-Nummer (00-FE); muß festgelegt werden
Byte	04:	11:	jetziger Track (00-22); muß festgelegt werden
Byte	05:	0F:	jetziger Sektor (00-0F); muß festgelegt werden
Byte	06:	LL:	Low Byte der DCT; muß festgelegt werden
Byte	07:	HH:	High Byte der DCT; muß festgelegt werden
Byte	08:	LL:	Low Byte des 256-Byte-Puffers; muß festgelegt werden
Byte	09:	HH:	High Byte des 256-Byte-Puffers; muß festgelegt werden
Byte	0A:	00:	Konstante, stets 00
Byte	0B:	00:	Konstante; stets 00
Byte	0C:	01:	Befehl (00 = Seek, 01 = Read, 02 = Write, 04 = Init); muß festgelegt werden
Byte	0D:	00:	Fehler-Code NACH der RWTS, daher nichts festlegen!
Byte	0E:	00:	vorherige Volume-Nummer; muß festgelegt werden
Byte	0F:	60:	vorheriger Slot; muß festgelegt werden
Byte	10:	01:	vorheriges Drive; muß festgelegt werden

DCT = Device Characteristics Table (Controller-Konstanten)

Byte 11: 00: Konstante, stets 00
Byte 12: 01: Konstante, stets 01
Byte 13: EF: Konstante, stets EF
Byte 14: D8: Konstante, stets D8

Da sich innerhalb der RWTS selbst ein DOS-interner IOB befindet, nämlich im Bereich $B7E8-$B7F8, gibt es grundsätzlich 2 Möglichkeiten: entweder man benutzt einen eigenen IOB oder den bereits vorliegenden IOB. In der Regel wählt man die erstere Alternative, da man dann auf die indirekte Adressierung verzichten kann.

Anmerkungen zum IOB:

Jetziger Slot: muß in der Form S0 spezifiziert werden, wobei S = 1-7, also 10 für Slot 1, 60 für Slot 6 usw.

Jetzige Volume-Nummer: wenn man 00 angibt, wird die tatsächliche Volume-Nummer ignoriert.

Puffer: ist 256 Bytes lang und kann sich fast überall im freien RAM-Speicher befinden, z.B. $0200-$02FF, $D000-$D0FF der Language Card (falls read-write-enabled) usw.
Befehle: Als Befehle kommen meist nur in Frage

01 = Read = Lese Disk-Sektor in RAM-Puffer
02 = Write = Schreibe RAM-Puffer auf Disk-Sektor

Mit 00 = Seek kann man die Positioniergeschwindigkeit des Lesekopfs testen. Wen dies interessiert, kann bei dem Programm 6.1.2. „RWTS mit Warteschleife" durch 0812: 00 einen vollständigen Seek-Test durchführen, da dann alle Sektoren „gesucht" werden.
Wenn man 04 = Init eingibt, dann wird die GANZE Diskette initialisiert, also nicht etwa nur der spezifizierte Sektor. Die Werte von Spur und Sektor brauchen für diesen Zweck nicht initialisiert zu werden, doch müssen an dieser Stelle legale Werte stehen.

Vorherige Volume-, Slot-, Drive-Werte: diese müssen stets initialisiert werden, wenn man einen eigenen IOB benutzt, wobei die vorherigen Werte dem DOS-internen IOB zu entnehmen sind. Dies geschieht so:

```
JSR   $03E3          ;lädt A mit High und Y mit Low Byte des DOS-internen
                     IOB
STY   $CE            ;frei auf Zero-Page
STA   $CF            ;frei auf Zero-Page
LDY   # $01          ;Byte 01
LDA   ($CE),Y        ;jetziger Slot wird
STA   VORSLOT        ;Vorslot im eigenen IOB (Byte 0F)
STA   JETZTSLOT      ;Byte 01, diese Zeile nur, wenn Jetztslot = Vorslot sein soll!
LDY   # $02          ;Byte 02
LDA   ($CE),Y        ;jetziger Drive wird
STA   VORDRIVE       ;Vordrive im eigenen IOB (Byte 10)
STA   JETZTDRIVE     ;Byte 02, diese Zeile nur, wenn Jetztdrive = Vordrive sein
                     soll!
LDY   # $0E          ;Byte 0E
LDA   ($CE),Y        ;Vorvolume wird
STA   VORVOLUME      ;Vorvolume im eigenen IOB (Byte 0E)
LDA   # $00          ;diese und die nächte Zeile nur, wenn Volume ignorierbar!
STA   VOLUME         ;Volumes (Byte 03): 00 = jedes Volume!
RTS
```

RWTS-Aufruf

Die RWTS wird am besten über die Page 3 DOS-Vektor-Tabelle, d.h. über JSR $03D9 aufgerufen. Zuvor muß der Akkumulator mit dem High Byte und das Y-Register mit dem Low Byte der IOB-Adresse geladen werden:

```
LDA   # >IOB         ;High Byte des IOB
LDY   # <IOB         ;Low Byte des IOB
JSR   $03D9          ;RWTS = meist $BD00
LDA   # $00          ;00 in
STA   $48            ;P-Register poken
BCS   FEHLER
RTS
```

Fehler-Routine

Wenn ein RWTS-Fehler auftrat, wird das Carry-Flag gesetzt. Man kann dann mit BCS in eine eigene Fehler-Routine springen, z.B.:

```
FEHLER  LDA   DOSERROR  ;Byte 0D
        JSR   $FDDA     ;Hex-Code anzeigen
        JMP   $03D0     ;DOS-Warmstart, entspricht Basic-END
```

Man beachte auch, daß man die Speicherstelle $0048 nach der RWTS auf Null setzen muß, da sonst der Monitor u.U. in den Dezimalmodus gelangen kann mit der Folge, daß z.B. die Bildschirmadressen nicht mehr richtig berechnet werden (sog. $48-Falle).

Es gibt also drei wichtige Vektor-Adressen, die man sich merken muß:

JSR $03E3 = GETIOB = Wo ist DOS-interner IOB?
JSR $03D9 = RWTS = Aufruf der eigentlichen RWTS
JMP $03D0 = WARMSTART = etwa Basic-END entsprechend

6.1. Testprogramme für RWTS

In diesem Kapitel (6.1.1. — 6.1.3) werden ein RWTS-Musterprogramm sowie zwei RWTS-Testprogramme als Assemblerlistings vorgestellt:

1. RWTS-Muster

Dieses vereinfachte, leichtverständliche, ca. 100 Bytes lange RWTS-Programm dient zum Einlesen von jeweils einem einzigen, beliebigen Sektor der Diskette. Die Pufferadresse steht — der Einfachheit halber mit High Byte first — in

0305: 10 00 = $1000-$10FF

und kann durch Pokes verändert werden. Z.B. 0305: 20 00 würde Puffer $2000-$20FF bedeutet. Man beachte, daß das Low Byte des Puffers keine Null sein muß, d.h. 0305: 12 34 = ab $1234 wäre zulässig.
Das RWTS-Muster nimmt stets die zuletzt durch CATALOG o.ä. angesprochenen Slot und Drive als Parameter (in der Regel S6,D1).
Anstelle des Basic-Programms kann man die gewünschte Spur mit 0303: XX und den gewünschten Sektor mit 0304: YY poken und dann die Routine mit 300G vom Monitor aus starten. Beispiel:

```
BLOAD RWTS-MUSTER
CALL -151       ;Monitor
```

```
303: 11 00      ;Spur $11, Sektor $00
300G            ;Start der RWTS
1000L           ;Listen des eingelesenen Sektors
```

Warnung!

RWTS-„Neulinge" können mit diesem Programm ihre ersten RWTS-Erfahrungen sammeln. Grundsätzlich gilt sowohl für Neulinge wie für Erfahrene: Verwenden Sie zum Austesten eines eigenen RWTS-Programms grundsätzlich zunächst eine „Scratch"-Diskette oder eine Kopie der eigentliche Diskette, da immer mit Programmierfehlern und daher mit einer möglichen Datenzerstörung gerechnet werden muß. Denken Sie immer daran, daß die RWTS in nur 1 Sekunde fast 30 Sektoren überschreiben kann!

2. RWTS-Test mit Warteschleife

Dieser Test zeigt, daß zwischen dem vorangehenden und dem nachfolgenden Aufruf der RWTS bei physischen Laufwerken (im Gegensatz zu RAM-Disks) maximal 2200 Prozessor-Takte vergehen dürfen, wenn die maximale Übertragungsrate von ca. 7400 Bytes/Sekunde nicht dramatisch sinken soll. 2200 Takte entsprechen

1000000 Takte : 1 Sekunde = 2200 Takte : ? Sekunden

0,0022 Sekunden, d.h. 2,2 Tausendstel Sekunde.

3. RWTS-Test mit Skewing

Dieses Beispiel zeigt, daß das Einlesen in aufsteigenden Spuren (00-22 statt 22-00) und Sektoren (00-0F statt 0F-00) selbst ohne Warteschleife wie beim vorangehenden Test zu einer dramatischen Verringerung der Übertragungsrate führt. Auch dieser Test gilt selbstverständlich wiederum nicht für RAM-Disks.

Unter Skewing („Abschrägung") versteht man das Verhältnis zwischen physischer und logischer Anordnung der Sektoren auf einer Spur. Nehmen wir an, die 16 Sektoren wären auf der Diskette physisch aufsteigend

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

angeordnet und würden auch in dieser Folge eingelesen, dann würde gelten

physische Anordnung = logische Anordnung.

Wenn die RWTS so schnell wäre, daß mit einer einzigen Rotation oder Umdrehung der Diskette eine ganze Spur eingelesen werden könnte, dann wäre diese Anordnung der Sektoren optimal. Aufgrund der Hardware-Gegebenheiten ist jedoch exakt in dem Moment, wo gerade der vorangehende physische Sektor eingelesen wurde, der Anfang des nachfolgenden physischen Sektors bereits am Lesekopf vorbeirotiert. Ja, es dauert in der Regel sogar meist mehr als 1 Umdrehung, bis der Lesekopf den physisch nachfolgenden Sektor einlesen kann. Daher ist es sinnvoll, die Sektoren einer Spur physisch verteilt anzuordnen, z.B. nach dem Descending Skew 2 Verfahren (absteigend jeder zweite Sektor):

```
00  07  0E  06  0D  05  0C  04  0B  03  0A  02  09  01  08  0F
    07      06      05      04      03      02      01
```

Die Sektoren muß man sich kreisförmig angeordnet denken. Der Einfachheit halber sind die Sektoren 01-07 in der zweiten Zeile erneut aufgeführt, da bei ihnen der absteigende Skew-Faktor 2 besonders deutlich wird.
Anstelle daß man die Sektoren physisch auf der Diskette z.B. mit Descending Skew 2 anordnet, besteht auch die Möglichkeit, daß man die physisch normale Anordnung (siehe obiges Beispiel) beläßt und statt dessen mit einer logischen Skew-Tabelle arbeitet. Dieses letztere Verfahren wird von DOS benutzt.

Praktische Schlußfolgerungen aus den Test-Programmen 6.1.2 und 6.1.3:

a) Wenn möglich, lese man eine Spur oder eine Datei „auf einen Schlag" ein, anstatt daß man zwischendurch nach jedem eingelesenen Sektor z.B. Move-Routinen, Datenkonvertierungen usw. vornimmt.

b) Wenn man Dateien einliest, liest man aufgrund der TSL automatisch die Sektoren absteigend von 0F-00. Beim Einlesen reiner Spuren sollte man dasselbe Verfahren praktizieren.

c) Für reinen RWTS-Diskettenzugriff ist der durch das Initialisieren entstandene Descending Skew 2 am besten geeignet. Für gemischte Diskettenzugriffsroutinen (reine RWTS und ferner LOAD, SAVE, GETLN, RDKEY, COUT, PRINT, INPUT, GET usw.) muß man von Fall zu Fall entscheiden, ob es sich lohnt, mit einer auf dem Software-Markt käuflichen Utility wie z.B. „Bag of Tricks" (Quality Software) die Skew-Werte zu ändern. In der Regel ist hiervon abzuraten, da z.B. derart modifizierte Disketten nicht mit COPYA kopiert werden können. Genauer gesagt entsteht zwar eine korrekte Kopie, doch mit dem üblichen und nicht mit dem modifizierten Skew-Wert. Hinzu kommt, daß das INIT-Modul von „Bag of Tricks" nach

meinen Erfahrungen gelegentlich Datendisketten zerstört. Ganz abzuraten ist von dem Verfahren, die DOS-interne Skew-Tabelle ($BFB8-$BFC7) vor dem Initialisieren abzuändern.

6.1.1. RWTS-Muster

```
:ASM
                    1   *         Rwts-Muster
                    2   *         ===========
                    3   *
                    4   * Leichtverständliches Muster
                    5   * für die RWTS-Routine
                    6   *
                    7   * Hier aus Sicherheitsgründen
                    8   * auf Read beschränkt!
                    9   *
                   10   * Man kann jeweils einen Sektor
                   11   * einlesen, der sich dann bei
                   12   * $1000-$10FF befindet.
                   13   *
                   14   * Zugehöriges Applesoft-Programm
                   15   * ------------------------------
                   16   *
                   17   * 10 INPUT "TRACK  (0-34):";T
                   18   * 20 INPUT "SEKTOR (0-15):";S
                   19   * 30 POKE 768+3,T
                   20   * 40 POKE 768+4,S
                   21   * 50 CALL 768
                   22   * 60 GOTO 10
                   23   *
                   24   *--------------------------------
                   25   *
                   26            ORG  768
                   27   *
0300: 4C 1C 03     28            JMP  START
                   29   *
0303: 00           30   SPUR     DFB  0          ;0-34
0304: 00           31   SEKTOR   DFB  0          ;0-15
                   32   *
0305: 10 00        33   PUFFER   HEX  1000       ;$1000
                   34   *
                   35   *--------------------------------
                   36   *
                   37   * RWTS-Datenblock: IOB + DCT
                   38   * --------------------------
                   39   *
                   40   * Input Output Block
                   41   *
0307: 01           42   IOB      HEX  01         ;IMMER
                   43   *
                   44   * Slot, Drive, Volume jetzt
                   45   *
0308: 60           46   SLOT     HEX  60         ;10-70
0309: 01           47   DRIVE    HEX  01         ;01-02
030A: 00           48   VOLUME   HEX  00
```

```
                        49  *
                        50  * Spur und Sektor jetzt
                        51  *
030B: 00                52  TRACK    HEX  00
030C: 00                53  SECTOR   HEX  00
                        54  *
                        55  * DCT-Adresse direkt nach IOB
                        56  *
030D: 18                57  DCTLOW   DFB  #<DCT
030E: 03                58  DCTHIGH  DFB  #>DCT
                        59  *
                        60  * Puffer-Adresse, hier $1000
                        61  *
030F: 00                62  BUFLOW   HEX  00
0310: 10                63  BUFHIGH  HEX  10
                        64  *
                        65  * Hier immer 2 mal 00
                        66  *
0311: 00 00             67  COUNT    HEX  0000       ;IMMER
                        68  *
                        69  * Command = Befehl
                        70  *
                        71  * 00 = SEEK  = Lesekopf auf
                        72  *              Sektor ausrichten
                        73  * 01 = READ  = Sector einlesen
                        74  *              in Puffer
                        75  * 02 = WRITE = Sector beschreiben
                        76  *              mit Pufferinhalt
                        77  * 04 = INIT  = ganze Diskette
                        78  *              formatieren
                        79  *
0313: 01                80  COMMAND  HEX  01         ;01-04
                        81  *
                        82  * Hier wird nach der RWTS der
                        83  * mögliche Fehler-Code abgelegt
                        84  *
0314: 00                85  DOSERROR HEX  00
                        86  *
                        87  * Hier müssen vor (!) der RWTS
                        88  * die vorherigen Volume-, Slot-
                        89  * und Drive-Werte eingesetzt
                        90  * werden. (Nach der RWTS wird
                        91  * effektives Volume automatisch
                        92  * hier von der RWTS gepokt.)
                        93  *
0315: 00                94  VORVOL   HEX  00
0316: 60                95  VORSLOT  HEX  60
0317: 01                96  VORDRIVE HEX  01
                        97  *
                        98  * Device Characteristics Table
                        99  * enthält stets diese 4 Bytes
                       100  *
0318: 00 01 EF         101  DCT      HEX  0001EFD8   ;IMMER
031B: D8
                       102  *
                       103  *-------------------------------
                       104  *
                       105  * Aktiver Slot + Drive ermitteln
```

```
                        106  *
031C: 20 E3 03          107  START    JSR  $03E3       ;RWTSWO?
031F: 84 CE             108           STY  $CE         ;LOWIOB
0321: 85 CF             109           STA  $CE+1       ;HIGHIOB
0323: A0 01             110           LDY  #1
0325: B1 CE             111           LDA  ($CE),Y
0327: 8D 08 03          112           STA  SLOT
032A: C8                113           INY
032B: B1 CE             114           LDA  ($CE),Y
032D: 8D 09 03          115           STA  DRIVE
                        116  *
0330: AD 03 03          117           LDA  SPUR
0333: 8D 0B 03          118           STA  TRACK
0336: AD 04 03          119           LDA  SEKTOR
0339: 8D 0C 03          120           STA  SECTOR
                        121  *
                        122  * Puffer festlegen, hier $1000
                        123  *
033C: AD 05 03          124           LDA  PUFFER
033F: 8D 10 03          125           STA  BUFHIGH
0342: AD 06 03          126           LDA  PUFFER+1
0345: 8D 0F 03          127           STA  BUFLOW
                        128  *
                        129  *-------------------------------
                        130  *
                        131  * Eigentliche RWTS
                        132  * ----------------
                        133  *
0348: A9 03             134           LDA  #>IOB
034A: A0 07             135           LDY  #<IOB
034C: 20 D9 03          136           JSR  $03D9       ;RWTS
034F: B0 05             137           BCS  FEHLER
                        138  *
                        139  * Processor Status Register
                        140  * auf Null setzen und Exit
                        141  *
0351: A9 00             142           LDA  #0
0353: 85 48             143           STA  $48         ;P-REG
0355: 60                144           RTS              ;EXIT
                        145  *
                        146  * Fehler-Nummer anzeigen und Exit
                        147  *
0356: AD 14 03          148  FEHLER   LDA  DOSERROR
0359: 20 DA FD          149           JSR  $FDDA       ;HEXZAHL
035C: A9 00             150           LDA  #0
035E: 85 48             151           STA  $48         ;P-REG
0360: 4C D0 03          152           JMP  $03D0       ;DOSWARM


--End assembly--

99 bytes

Errors: 0
```

6.1.2. RWTS-Test mit Warteschleife

```
:ASM
                        1               ORG  $803
                        2    *
                        3    *  RWTS-TEST MIT WARTESCHLEIFE
                        4    *
                        5    *  FAUSTREGEL
                        6    *  ==========
                        7    *
                        8    *  DIE RWTS DARF DURCH BIS ZU
                        9    *  2200 TAKTE-LANGE ASSEMBLER-
                       10    *  ROUTINEN UNTERBROCHEN WERDEN
                       11    *  OHNE DASS SICH LESETEMPO
                       12    *  VERRINGERT.
                       13    *
0803: 4C 1B 08         14              JMP  STARTER
                       15    RWTS      EQU  $03D9
                       16    HOME      EQU  $FC58
                       17    DOSCOLD   EQU  $3D3
                       18    *INPUT-OUTPUT-CONTROL-BLOCK*****
0806: 01               19    IOB       HEX  01          ;STETS
0807: 60               20    SLOT      HEX  60          ;SLOT 6
0808: 01               21    DRIVE     HEX  01          ;DRIVE1
0809: 00               22    VOLUME    HEX  00
080A: 00               23    TRACK     HEX  00
080B: 00               24    SECTOR    HEX  00
080C: 17               25    DCTLOW    DFB  #<DCT
080D: 08               26    DCTHIGH   DFB  #>DCT
080E: 00               27    BUFLOW    HEX  00
080F: 10               28    BUFHIGH   HEX  10          ;$1000
0810: 00               29              HEX  00          ;UNUSED
0811: 00               30              HEX  00          ;COUNT:0
0812: 01               31    COMMAND   HEX  01          ;READ
0813: 00               32    DOSERROR  HEX  00
0814: 00               33    EFFVOL    HEX  00
0815: 60               34    VORSLOT   HEX  60
0816: 01               35    VORDRIVE  HEX  01
                       36    *DEVICE-CHARACTERISTICS-TABLE***
0817: 00 01 EF         37    DCT       HEX  0001EFD8    ;STETS
081A: D8
                       38    *
                       39    * 3 MAL TOTALES DISKLESEN
                       40    *
081B: 20 58 FC         41    STARTER   JSR  HOME
081E: 20 2A 08         42              JSR  LESERO
0821: 20 2A 08         43              JSR  LESERO
0824: 20 2A 08         44              JSR  LESERO
0827: 4C D3 03         45              JMP  DOSCOLD
082A: A9 22            46    LESERO    LDA  #$22         ;22-00
082C: 8D 0A 08         47              STA  TRACK
082F: A9 0F            48              LDA  #$0F         ;0F-00
0831: 8D 0B 08         49              STA  SECTOR
0834: 4C 47 08         50              JMP  LESER2
```

```
0837: CE 0B 08  51  LESER1   DEC  SECTOR
083A: 10 0B     52           BPL  LESER2
083C: A9 0F     53           LDA  #$0F
083E: 8D 0B 08  54           STA  SECTOR
0841: CE 0A 08  55           DEC  TRACK
0844: 10 01     56           BPL  LESER2
0846: 60        57           RTS              ;ENDE
0847: 18        58  LESER2   CLC
0848: AD 0B 08  59           LDA  SECTOR
084B: 69 10     60           ADC  #$10
084D: 8D 0F 08  61           STA  BUFHIGH
0850: 20 5F 08  62           JSR  WAITER
0853: A9 08     63           LDA  #>IOB
0855: A0 06     64           LDY  #<IOB
0857: 20 D9 03  65           JSR  RWTS
085A: 90 DB     66           BCC  LESER1      ;OKAY
085C: 4C D3 03  67           JMP  DOSCOLD
                68  *
                69  * WAIT-ROUTINE ****
                70  *
                71  * (X * 200) + 6 TAKTE
                72  *
                73  * BEI INSGESAMT 11*200+6=2206
                74  * TAKTEN WAIT-ROUTINE LIEGT NOCH
                75  * DIE OPTIMALE GESCHWINDIGKEIT
                76  * VON CA. 58 SEKUNDEN VOR
                77  * D.H. 7400 BYTES/SEKUNDE
                78  *
085F: 20 87 08  79  WAITER   JSR  WAITER1    ;6 T.
0862: 20 87 08  80           JSR  WAITER1
0865: 20 87 08  81           JSR  WAITER1
0868: 20 87 08  82           JSR  WAITER1
086B: 20 87 08  83           JSR  WAITER1
086E: 20 87 08  84           JSR  WAITER1
0871: 20 87 08  85           JSR  WAITER1
0874: 20 87 08  86           JSR  WAITER1
0877: 20 87 08  87           JSR  WAITER1
087A: 20 87 08  88           JSR  WAITER1
087D: 20 87 08  89           JSR  WAITER1    ;11.
                90  *
                91  * BEI 12*200+6=2406 TAKTEN SACKT
                92  * DIE GESCHWINDIGKEIT BEREITS
                93  * AUF 96 SEKUNDEN AB
                94  * D.H. 4470 BYTES/SEKUNDE
                95  *
0880: 20 87 08  96           JSR  WAITER1    ;12.
                97  *
                98  * BEI 13*200+6=2606 TAKTEN IST
                99  * TIEFPUNKT DER GESCHWINDIGKEIT
               100  * ERREICHT MIT 310 (!!!) SEKUNDEN
               101  * D.H. 1410 BYTES/SEKUNDE
               102  *
0883: 20 87 08  103          JSR  WAITER1    ;13.
               104  *
```

```
0886: 60          105             RTS                 ;6 T.
                  106  *
                  107  * 5 * (30+6) + 4*2 + 6 = 194 T.
                  108  *
0887: 20 9B 08    109  WAITER1    JSR  WAITER2       ;6 T.
088A: 20 9B 08    110             JSR  WAITER2
088D: 20 9B 08    111             JSR  WAITER2
0890: 20 9B 08    112             JSR  WAITER2
0893: 20 9B 08    113             JSR  WAITER2
0896: EA          114             NOP                 ;2 T.
0897: EA          115             NOP
0898: EA          116             NOP
0899: EA          117             NOP
089A: 60          118             RTS                 ;6 T.
                  119  * 30 TAKTE
089B: AD FF FF    120  WAITER2    LDA  $FFFF          ;4 T.
089E: AD FF FF    121             LDA  $FFFF
08A1: AD FF FF    122             LDA  $FFFF
08A4: AD FF FF    123             LDA  $FFFF
08A7: AD FF FF    124             LDA  $FFFF
08AA: AD FF FF    125             LDA  $FFFF
08AD: 60          126             RTS                 ;6 T.

--End assembly--
```

6.1.3. RWTS-Test mit Skewing

```
:ASM
                        1               ORG   $803
                        2     *
                        3     * RWTS-TEST MIT SKEWING
                        4     * ASCENDING: T.00-22 UND S.00-0F
                        5     *
0803: 4C 1B 08          6               JMP   STARTER
                        7     RWTS      EQU   $03D9
                        8     HOME      EQU   $FC58
                        9     DOSCOLD   EQU   $3D3
                       10     *INPUT-OUTPUT-CONTROL-BLOCK*****
0806: 01                11     IOB       HEX   01            ;STETS
0807: 60                12     SLOT      HEX   60            ;SLOT 6
0808: 01                13     DRIVE     HEX   01            ;DRIVE1
0809: 00                14     VOLUME    HEX   00
080A: 00                15     TRACK     HEX   00
080B: 00                16     SECTOR    HEX   00
080C: 17                17     DCTLOW    DFB   #<DCT
080D: 08                18     DCTHIGH   DFB   #>DCT
080E: 00                19     BUFLOW    HEX   00
080F: 10                20     BUFHIGH   HEX   10            ;$1000
0810: 00                21               HEX   00            ;UNUSED
0811: 00                22               HEX   00            ;COUNT:0
0812: 01                23     COMMAND   HEX   01            ;READ
0813: 00                24     DOSERROR  HEX   00
0814: 00                25     EFFVOL    HEX   00
0815: 60                26     VORSLOT   HEX   60
0816: 01                27     VORDRIVE  HEX   01
                        28     *DEVICE-CHARACTERISTICS-TABLE***
0817: 00 01 EF          29     DCT       HEX   0001EFD8      ;STETS
081A: D8
                        30     *
                        31     * 3 MAL TOTALES DISKLESEN
                        32     *
081B: 20 58 FC          33     STARTER   JSR   HOME
081E: 20 2A 08          34               JSR   LESERO
0821: 20 2A 08          35               JSR   LESERO
0824: 20 2A 08          36               JSR   LESERO
0827: 4C D3 03          37               JMP   DOSCOLD
082A: A9 00             38     LESERO    LDA   #$00           ;00-22
082C: 8D 0A 08          39               STA   TRACK
082F: A9 00             40               LDA   #$00           ;00-0F
0831: 8D 0B 08          41               STA   SECTOR
0834: 4C 51 08          42               JMP   LESER2
0837: EE 0B 08          43     LESER1    INC   SECTOR
083A: AD 0B 08          44               LDA   SECTOR
083D: C9 10             45               CMP   #$10
083F: 90 10             46               BCC   LESER2
0841: A9 00             47               LDA   #$00
0843: 8D 0B 08          48               STA   SECTOR
0846: EE 0A 08          49               INC   TRACK
```

```
0849: AD 0A 08  50            LDA  TRACK
084C: C9 23     51            CMP  #$23
084E: 90 01     52            BCC  LESER2
0850: 60        53            RTS              ;ENDE
0851: 18        54  LESER2    CLC
0852: AD 0B 08  55            LDA  SECTOR
0855: 69 10     56            ADC  #$10
0857: 8D 0F 08  57            STA  BUFHIGH
085A: A9 08     58            LDA  #>IOB
085C: A0 06     59            LDY  #<IOB
085E: 20 D9 03  60            JSR  RWTS
0861: 90 D4     61            BCC  LESER1      ;OKAY
0863: 4C D3 03  62            JMP  DOSCOLD
```

6.2. Die Programme auf der Begleitdiskette

Die Begleitdiskette zu diesem Buch enthält die nachfolgend charakterisierten Programme. Bei Maschinenprogrammen ist der Quelltext im Big-Mac/Merlin-Textfile-Format mit aufgenommen worden. In der Regel wird normales 48K-DOS mit Maxfiles 3 erwartet. Als „normales DOS" gilt auch Diversi-DOS 2C. Wie mann die Begleitdiskette, die selbst kein DOS 3.3 enthält, mit Hilfe des Diversi-DOS-Files ASMDIV in eine Diversi-DOS-Bootdiskette verwandelt, wird weiter unten gezeigt.
Soweit nicht anders vermerkt, laufen die Programme auf allen Apple-II-Typen (+/e/c/enhanced e).

DISK VOLUME 254

A 002 STIEHL
T 004 BLOAD.FINDER
T 003 LIST.MAKER
B 002 TRACK.SECTOR.TRACER
T 004 T.TRACK.SECTOR.TRACER
B 002 FREIE.SEKTOREN
T 003 T.FREIE.SEKTOREN
B 002 FUSION.TASC
T 006 T.FUSION.TASC
A 004 CPM.REFINER
B 002 CPM.REFINER.OBJ
T 006 T.CPM.REFINER.OBJ
B 002 DOS.OUTPUT.VEKTOR
T 008 T.DOS.OUTPUT.VEKTOR
B 002 RWTS.MUSTER
T 011 T.RWTS.MUSTER
B 002 RWTS.TEST.WARTESCHLEIFE
T 009 T.RWTS.TEST.WARTESCHLEIFE
B 002 RWTS.TEST.SKEWING
T 005 T.RWTS.TEST.SKEWING
A 003 DISK.LESER
B 002 DISK.LESER.OBJ
T 007 T.DISK.LESER.OBJ
A 004 FILE.LESER
B 003 FILE.LESER.OBJ
T 013 T.FILE.LESER.OBJ

B 004 KOPIERPROGRAMM.2DRIVE
T 018 T.KOPIERPROGRAMM.2DRIVE
B 005 KOPIERPROGRAMM.1DRIVE
T 020 T.KOPIERPROGRAMM.1DRIVE
B 004 DISK.COMPARER
T 016 T.DISK.COMPARER
B 003 BAD.SECTOR.ROUTINE
T 010 T.BAD.SECTOR.ROUTINE
A 003 DOS.KOPIE
B 002 DOS.KOPIE.OBJ
T 008 T.DOS.KOPIE.OBJ
A 002 DOSLOS
B 004 DOSLOS.OBJ
T 017 T.DOSLOS.OBJ
B 005 RAMDISK64
T 040 T.RAMDISK64
B 007 RAMDISK64.MOVER
T 032 T.RAMDISK64.MOVER
A 024 HELLO
B 114 ASMDIV
A 002 DIVERSI.START

BLOAD.FINDER (s.S. 66)
Nach dem Laden einer Binärdatei mit BLOAD kann deren Startadresse und Länge mit
EXEC BLOAD.FINDER
ermittelt werden.

LIST.MAKER (s.S. 67)
Der LIST.MAKER, der mit
EXEC LIST.MAKER
gestartet wird, verwandelt ein sich bereits im Speicher befindliches Applesoft-Programm in einen Textfile.

TRACK.SECTOR.TRACER (s.S. 76)
T.TRACK.SECTOR.TRACER
Mit diesem Programm, das mit
BRUN TRACK.SECTOR.TRACER
gestartet wird und das ein normales 48K-DOS voraussetzt, läßt sich verfolgen, auf welche Sektoren bei CATALOG und anderen DOS-Befehlen im einzelnen zugegrif-

fen wird. Dieses Programm ist nur zum Experimentieren und nicht als eigenständiges Hilfsprogramm gedacht. Wenn man den Tracer nicht mehr benötigt, sollte man neu booten. Der Tracer funktioniert nicht, wenn eine RAM-Disk installiert ist.

FREIE.SEKTOREN (s.S. 93)
T.FREIE.SEKTOREN
Dieses Programm, das ein normales 48K-DOS voraussetzt, wird mit
BLOAD FREIE.SEKTOREN
installiert (nicht mit BRUN starten!). Sodann kann nach einem Diskettenzugriff, z.B. nach CATALOG, mit
CALL 768
die Anzahl der freien Sektoren auf der angesprochenen Diskette ermittelt und angezeigt werden. Diese Utility ist in der Regel auch auf RAM-Disks anwendbar, doch versagt sie bei modifizierten DOS-Disketten mit mehr als 35 Spuren.

FUSION.TASC (s.S. 95)
T.FUSION.TASC
Diese Utility, die normales 48K-DOS voraussetzt, vereinigt die TASC-RUNTIME und einen TASC-Objektcode namens ALT zu einer Gesamtdatei namens NEU, die später mit
BRUN NEU
gestartet werden kann. ALT muß mit 2054 als „Address for Library" compiliert worden sein, und es dürfen keine Common-Definitionen und HGR-Befehle vorkommen.

CPM.REFINER (s.S. 97)
CPM.REFINER.OBJ
T.CPM.REFINER.OBJ
Diese Utility, die mit
RUN CPM.REFINER
gestartet wird, entfernt aus einer Wordstar-Datei, die mit CPMXFER von einer CP/M- auf eine DOS-Diskette als Binärfile transferiert wurde, die CP/M-typischen und zugleich DOS-untypischen Steuerzeichen.

DOS.OUTPUT.VEKTOR (s.S. 101)
T.DOS.OUTPUT.VEKTOR
Dieses Programm, das normales 48K-DOS voraussetzt und mit
BRUN DOS.OUTPUT.VEKTOR

gestartet wird, dient lediglich als Beispiel für die Steuerung und Filterung von Zeichen, die über COUT ausgegeben werden. Je nach Apple-II-Typ und Hardware-Konfiguration muß man eine eigene Output-Adresse bei
$0303: 20 LL HH
einsetzen.

RWTS.MUSTER (s.S. 113)
T.RWTS.MUSTER
Diese Routine läßt sich als Quelltext T.RWTS.MUSTER in eigene RWTS-Programme einbauen und dient zum Lesen/Schreiben eines definierten Sektors.

RWTS.TEST.WARTESCHLEIFE (s.S. 116)
T.RWTS.TEST.WARTESCHLEIFE
Dieses Testprogramm, das mit
BLOAD RWTS.TEST.WARTESCHLEIFE
CALL 2051
von Slot 6, Drive 1 gestartet wird, dient zur Ermittlung der Verzögerungszeit bei unterschiedlichen Laufwerken und/oder DOS-Varianten. Bei Bedarf kann man mit
CALL −151
0807: S0
0808: 0D
andere Slot/Drive-Parameter eintragen. Zur Feinabstimmung der Warteschleifen muß man den Quelltext T.RWTS.TEST.WARTESCHLEIFE selbst ändern.

RWTS.TEST.SKEWING (s.S. 119)
T.RWTS.TEST.SKEWING
Dieses Testprogramm, das wie RWTS.TEST.WARTESCHLEIFE gestartet wird, veranschaulicht das bestmögliche Skewing.

Alle nachfolgenden Programme haben einen größeren Umfang und werden im Anschluß an die Bedienungsanleitung komplett gelistet. Sie sind allesamt als selbstständige Utilities ausgelegt, die mit RUN oder BRUN gestartet werden und keinerlei Assemblerkenntnisse voraussetzen.

DISK.LESER (s.S. 130)
DISK.LESER.OBJ
T.DISK.LESER.OBJ
Dieses Programm, das mit
RUN DISK.LESER
gestartet wird, dient zum Einlesen und Anzeigen einzelner Spuren im ASCII-Format. Damit eignet es sich besonders zum (Unter)suchen von Textstellen auf einer Diskette.

Vor dem Programmstart kann man bei Bedarf die 80-Zeichenkarte einschalten. Man beachte, daß alle Ctrl-Zeichen in inverse und somit sichtbare Zeichen umgewandelt werden; dies gilt auch für Return. Nach dem Start des DISK.LESER braucht man lediglich die jeweils gewünschte Spur (im Bereich 0-34) einzugeben. Mit der Leertaste kann man das Scrollen vorübergehend anhalten. Der DISK.LESER greift immer auf den zuletzt angesprochenen Slot/Drive zu, also im Normalfall auf den Slot, von dem er gestartet wurde. Will man nach dem Start auf einen anderen Slot, z.B. S4, zugreifen, so verlasse man das Menü über „E" = Ende, erteile den Befehl CATALOG, S4 und gehe wieder in das Programm mit RUN 120 zurück.

FILE.LESER (s.S. 133)
FILE.LESER.OBJ
T.FILE.LESER.OBJ

Dieses Programm, das 48K-DOS mit Maxfiles 3 voraussetzt (ggf. vorher FP!) und mit

RUN FILE.LESER

gestartet wird, dient zum sehr schnellen Einlesen von sequentiellen Dateien beliebiger Art und Größe. Es verwendet den sog. UNLOCK-Trick, der in den zwei vorangehenden Auflagen dieses Buches anhand weiterer Beispielprogramme näher beschrieben wurde. Nach dem Start des FILE.LESER braucht man lediglich den Namen der Datei einzugeben, die dann eingelesen und im ASCII-Format angezeigt wird. Mit Ausnahme von Return werden alle Ctrl-Zeichen in inverse Großbuchstaben umgewandelt, so daß sich der FILE.LESER besonders für sequentielle Textfiles eignet. Wegen des UNLOCK-Tricks darf die Diskette, auf die zugegriffen wird, nicht schreibgeschützt sein. Wie der DISK.LESER greift auch der FILE.LESER auf das Start-Laufwerk zu. Wenn Sie nach dem Start von einem anderen Slot, z.B. S4, lesen wollen, so verlassen sie über „N" vorübergehend das Menü, sprechen Sie mit CATALOG, S4 das neue Laufwerk an und geben Sie dann RUN ein. Das Maschinenprogramm FILE.LESER.OBJ wird nämlich nach RUN nicht ein zweites Mal geladen. Der FILE.LESER ist nur für 40-Zeichendarstellung gedacht.

KOPIERPROGRAMM.2DRIVE (s.S. 138)
T.KOPIERPROGRAMM.2DRIVE

Dieses 35-Spur-Kopierprogramm, das mit

BRUN KOPIERPROGRAMM.2DRIVE

gestartet wird, setzt erstens eine Language-Card sowie zweitens zwei Laufwerke (an demselben Slot angeschlossen) voraus. Ferner muß 48K-DOS verwendet werden, weil die Language-Card als Datenpuffer benötigt wird. Nach dem Start muß man die

Originaldiskette in Drive 1 und die Duplikatdiskette in Drive 2 einlegen und dann „J" für „Start J/N" tippen. Dies ist alles. Es werden nur 3 Durchgänge bzw. Kopierschübe benötigt. Ein Neustart ist mit
CALL 3000
möglich. Wenn man zuvor mit
POKE 3003, 0
die Initialisierung der Duplikatdiskette unterbindet, was selbstverständlich eine bereits früher formatierte Diskette voraussetzt, so reduziert sich die Kopierzeit auf ca. 45s.

KOPIERPROGRAMM.1DRIVE (s.S. 145)
T.KOPIERPROGRAMM.1DRIVE
Dieses Programm, das mit
BRUN KOPIERPROGRAMM.1DRIVE
gestartet wird und 48K-DOS benötigt, ist für 1-Drive-Besitzer gedacht, doch wird auch hier eine Language-Card vorausgesetzt. Beachten Sie genau die Hinweise am Bildschirm, da Sie abwechselnd Original- und Duplikatdiskette einlegen müssen. Apple-IIe/c-Besitzer schalten bitte vor dem Start die 80-Zeichenkarte aus.

DISK.COMPARER (s.S. 154)
T.DISK.COMPARER
Dieses Programm vergleicht den Inhalt von zwei Disketten und setzt zwei Laufwerke (an demselben Slot angeschlossen) voraus. Starten Sie das Programm mit
BRUN DISK.COMPARER
von Drive 1 und legen Sie dann in die Laufwerke 1 und 2 die zu vergleichenden Disketten ein. Mit „W" für „Weiter" beginnt der spurweise Vergleich. Abweichende Sektoren werden angezeigt. Danach können Sie mit „N" bei „Weiter J/N" die Suche abbrechen und im Monitor die abweichenden Sektoren genauer untersuchen:
$1000-$1FFF enthält Spur von Drive 1
$2000-$2FFF enthält Spur von Drive 2
$1000-$10FF = 0. Sektor, $1100-$11FF = 1. Sektor usw.

BAD.SECTOR.ROUTINE (s.S. 160)
T.BAD.SECTOR.ROUTINE
Dieses Programm, das mit
BRUN BAD.SECTOR.ROUTINE
gestartet wird, ermittelt schadhafte Sektoren auf einer Diskette. Legen Sie nach dem Start die defekte Diskette in das Start-Laufwerk und beginnen Sie die Suche mit „W"

für „Weiter". Im Gegensatz zu allen anderen hier beschriebenen Programmen wird nach einen I/O-Fehler die Bad-Sector-Suche nicht abgebrochen. Durch Drücken der ESC-Taste können Sie jedoch das Programm vorzeitig beenden.

DOS.KOPIE (s.S. 164)
DOS.KOPIE.OBJ
T.DOS.KOPIE.OBJ

DOS 3.3 befindet sich auf den Spuren 0-2. Mit DOS.KOPIE können Sie diese Spuren neu überschreiben, z.B. mit Diversi-DOS o.ä. Ferner gestattet das Programm die Änderung des Hello-Namens. Starten Sie DOS.KOPIE mit

RUN DOS.KOPIE

Legen Sie dann die Originaldiskette mit dem gewünschten DOS in das Start-Laufwerk und tippen Sie nach „Original einlegen" auf die Return-Taste. Danach wird das Original-DOS eingelesen. Nun wird nach dem „Neuen Hello-Namen" gefragt. Geben Sie den neuen Namen ein oder tippen Sie lediglich auf die Return-Taste, falls der angezeigte alte Hello-Name nicht verändert werden soll. Nun legen Sie die Duplikatdiskette ein und tippen nach „Duplikat einlegen" auf die Return-Taste. Danach wird das Original-DOS (mit geändertem oder nicht-geändertem Hello-Namen) auf die Duplikatdiskette geschrieben.

Sie können DOS.KOPIE auch benutzen, wenn Sie bei einer Ihnen unbekannten Diskette den Hello-Namen erfahren wollen. Nach „Neuen Hello-Namen" drücken Sie Ctrl-Reset, um das Zurückschreiben des Original-DOS zu verhindern.

DOSLOS (s.S. 167)
DOSLOS.OBJ
T.DOSLOS.OBJ

Von den drei DOS-Spuren 0-2 kann man die Spuren 1-2 (dies sind immerhin 8192 weitere Bytes), nicht jedoch die Spur 0, als Datenspuren gewinnen. Zu diesem Zweck muß allerdings auch die VTOC geändert werden. Das Programm DOSLOS dient zum Formatieren von solchen sog. nicht-bootfähigen Datendisketten. Neben der Korrektur der VTOC wird außerdem auf dem Bootsektor (Spur 0, Sektor 0) eine Meldung installiert. Starten Sie das Programm mit

RUN DOSLOS

und legen Sie dann in das Start-Laufwerk die zu formatierende Diskette ein. Nach der Bildschirmmeldung brauchen Sie nur auf die Return-Taste zu tippen. Der Vorgang kann mehrmals wiederholt werden. Aus Sicherheitsgründen können Sie jedoch nicht im voraus auf Return tippen, sondern immer nur, wenn die Meldung erscheint.

RAMDISK64 (s.S. 174)
T.RAMDISK64
RAMDISK64.MOVER
T.RAMDISK64.MOVER

Dieses Programm stellt einen neuen, sehr schnellen RAM-Disk-Driver für die Apple-IIe-64K-Karte dar, der durch ein RAM-Disk-Kopierprogramm ergänzt wird. RAMDISK64 läuft unter jedem 48K-DOS, weil sich der Driver komplett auf der 64K-Karte befindet und außerhalb vom DOS im Bereich $03C0-$03CF der Sprung zur Karte installiert wird. Außerdem wird die 64K-Karte maximal als RAM-Disk ausgenutzt; lediglich der Bildschirmspeicher $0400-$07FF bleibt ausgespart. Der Driver wird mit

BRUN RAMDISK64

installiert. Auf die Frage „RAMDISK64 mit Init J/N" antworten Sie mit „J". Danach können Sie die RAM-Disk mit

S3, D1 oder
S3, D2

ansprechen. Sollte durch versehentliches Booten (mit PR#6 oder Ctrl-Offner Apfel-Reset) der Driver abgehängt worden sein, so können Sie nach erneutem

BRUN RAMDISK64

auf „Init J/N" mit „N" antworten. Dann wird nur der Driver angeschlossen, und der alte RAM-Disk-Inhalt ist wieder verfügbar.
RAMDISK64 läuft nicht auf dem Apple IIc, weil der IIc Teile der 64K-Karte als Modem-Puffer benutzt.

Mit dem Spezialprogramm RAMDISK64.MOVER besteht die Möglichkeit, den kompletten Inhalt der RAM-Disk auf einer physischen Diskette zu sichern. Starten Sie dieses Programm, das Maxfiles 3 voraussetzt (ggf. zuvor FP!), von Slot 6, Drive 1 mit

BRUN RAMDISK64.MOVER

und geben Sie „S" für „Speichern" ein. Danach wird der RAM-Disk-Inhalt als zwei Dateien (RAM1 und RAM2) auf Diskette gesichert. Der einmal gesicherte Inhalt kann mit „L" für „Laden" ebenso einfach wieder von der Diskette auf die RAM-Disk kopiert werden. Damit man keinen Fragenkatalog durchgehen muß, erwartet RAMDISK64.MOVER die Dateien RAM1/RAM2 in Slot 6, Drive 1. Mit

POKE 2054, Slot
POKE 2055, Drive

lassen sich jedoch die Parameter ändern.

HELLO
ASMDIV
DIVERSI.START
Die Begleitdiskette enthält zusätzlich die Diversi-DOS-2C-Dateien HELLO und ASMDIV. Diese stellen zunächst kein bootfähiges Diversi-DOS dar. Um die Konvertierung in bootfähiges Diversi-DOS zu ermöglichen, wurde die DOS-lose Begleitdiskette zu diesem Buch bereits entsprechend präpariert. Verfahren Sie zu diesem Zweck haargenau wie folgt:

1. Booten Sie die normale System-Master-Diskette mit dem Datum „08/25/80" mit PR#6.
Die System-Master-Diskette ist diejenige Diskette, die FID und COPYA enthält und bis vor einiger Zeit beim Erwerb eines Disk-II-Laufwerkes automatisch mitgeliefert wurde. Verwenden Sie nicht die Version mit dem Datum von 1983!
2. Jetzt legen Sie das *Original* der Begleitdiskette in Drive 1 und eine Leerdiskette in Drive 2. Starten Sie nunmehr mit
BRUN KOPIERPROGRAMM.2DRIVE
das oben beschriebene Kopierprogramm. Als 1-Drive-Besitzer verwenden Sie statt desssen KOPIERPROGRAMM.1DRIVE.
3. Nach Beendigung des Kopiervorgangs legen sie die *Kopie* der Begleitdiskette in Drive 1 und geben Sie den Befehl
RUN DIVERSI.START
ein. Nach etwa einer 1/2 Minute gelangen Sie in den Vorspann von ASMDIV. Tippen Sie auf Return und geben Sie dann im Hauptmenü die Ziffer „2" für „Put Diversi-DOS onto a Disk" ein. Nach wenigen Sekunden haben Sie eine unter Diversi-DOS bootfähige Begleitdiskette.
4. Booten Sie jetzt zu Testzwecken die Kopie der Begleitdiskette. Danach können Sie mit INIT HELLO wie üblich Disketten unter Diversi-DOS formatieren.

Für 40-Spur-Laufwerke kann Diversi-DOS 2C wie folgt gepatcht werden:
Call −151
AEB5: A0 (vorher 8C)
BEFE: 28 (vorher 23)

6.2.1. Diskettenleseprogramm

DISK.LESER

```
100  REM *** DISK.LESER ***
110  CLEAR : HIMEM: 8191: PRINT :
     PRINT  CHR$ (4)"BLOAD DISK.LESER.OBJ,A$300"
120  TEXT : HOME : INVERSE : PRINT "*** DISKLESER ***":
     PRINT "U.STIEHL * 1/1986": NORMAL
130  PRINT : PRINT : INPUT "E = ENDE * TRACK (0 - 34): ";X$:
     IF X$ = "E" THEN  TEXT : HOME : END
140  T =  INT ( VAL (X$)): ON T > 34 GOTO 120: POKE 775,T
150  HOME : INVERSE : PRINT "TRACK # ";T;
     "   ESC = ENDE  *  SPACE = STOP";: NORMAL :
     PRINT : POKE 34,2
160  CALL 768: REM TRACK AUF BILDSCHIRM LESEN
170  PRINT : PRINT "ERNEUT J/N ";
180  GET X$: IF X$ = "J" THEN 120
190  IF X$ <  > "N" THEN 180
200  PRINT : TEXT : INVERSE :
     PRINT "TRACK # ";T;"   IN MONITOR $2000 TO $2FFF  ":
     NORMAL
```

DISK.LESER.OBJ

```
BSAVE DISK.LESER.OBJ, A$0300, L190

                  1            ORG  $0300
                  2    *
                  3    * DISK.LESER.OBJ
                  4    * ==============
                  5    *
                  6    PUFFER   EQU  $2000
                  7    IND      EQU  $CE
                  8    DOSWARM  EQU  $03D0
                  9    COUT     EQU  $FDED
                  10   CROUT    EQU  $FD8E
                  11   PRBYTE   EQU  $FDDA
                  12   SETINV   EQU  $FE80
                  13   SETNORM  EQU  $FE84
                  14   *
0300: 4C 18 03    15           JMP  START
                  16   *
0303: 01          17   IOB      HEX  01
0304: 60          18   SLOT     HEX  60         ;Slot 6
0305: 01          19   DRIVE    HEX  01         ;Drive 1
```

```
0306: 00        20            HEX  00
0307: 00        21  TRACK     HEX  00          ;Track
0308: 00        22  SECTOR    HEX  00          ;Sektor
0309: 14        23            DFB  #<DCT
030A: 03        24            DFB  #>DCT
030B: 00        25  PUFFLOW   HEX  00          ;$4000
030C: 40        26  PUFFHIGH  HEX  40
030D: 00        27            HEX  00
030E: 00        28            HEX  00
030F: 01        29            HEX  01          ;Read
0310: 00        30            HEX  00
0311: 00        31            HEX  00
0312: 60        32            HEX  60          ;Slot 6
0313: 01        33            HEX  01          ;Drive 1
0314: 00        34  DCT       DFB  0
0315: 01        35            DFB  1
0316: EF        36            DFB  239
0317: D8        37            DFB  216
                38  *
                39  * Lese eine Spur von Sektor F-0
                40  * rückwärts in den Puffer von
                41  * $2000-2FFF
                42  *
0318: 20 E3 03  43  START     JSR  $03E3       ;RWTSWO?
031B: 84 CE     44            STY  IND
031D: 85 CF     45            STA  IND+1
031F: A0 01     46            LDY  #1
0321: B1 CE     47            LDA  (IND),Y
0323: 8D 04 03  48            STA  SLOT
0326: C8        49            INY
0327: B1 CE     50            LDA  (IND),Y
0329: 8D 05 03  51            STA  DRIVE
                52  *
032C: A9 10     53            LDA  #16         ;16-1
032E: 8D 08 03  54            STA  SECTOR
0331: A9 00     55            LDA  #<PUFFER
0333: 8D 0B 03  56            STA  PUFFLOW
0336: A9 1F     57            LDA  #>PUFFER-1
0338: 8D 0C 03  58            STA  PUFFHIGH    ;$1F00
033B: EE 0C 03  59  READLOOP  INC  PUFFHIGH
033E: CE 08 03  60            DEC  SECTOR
0341: AD 08 03  61            LDA  SECTOR
0344: 30 0F     62            BMI  DISPLAY
0346: A9 03     63            LDA  #>IOB
0348: A0 03     64            LDY  #<IOB
034A: 20 D9 03  65            JSR  $03D9       ;RWTS
034D: B0 03     66            BCS  ERROR
034F: 4C 3B 03  67            JMP  READLOOP
0352: 4C D0 03  68  ERROR     JMP  DOSWARM
                69  *
                70  * ASCII-Speicher anzeigen
                71  *
0355: A9 FF     72  DISPLAY   LDA  #<PUFFER-1
0357: 85 CE     73            STA  IND
```

```
0359: A9 1F     74           LDA  #>PUFFER-1 ;$-1FFF
035B: 85 CF     75           STA  IND+1
035D: A9 10     76           LDA  #$10
035F: 8D BD 03  77           STA  COUNTER
0362: A0 00     78  LOOP     LDY  #0
0364: E6 CE     79           INC  IND
0366: D0 3F     80           BNE  LOOP1
0368: A9 A0     81           LDA  #$A0
036A: 20 ED FD  82           JSR  COUT
036D: 20 8E FD  83           JSR  CROUT
0370: A5 CF     84           LDA  IND+1
0372: C9 2F     85           CMP  #$2F
0374: F0 21     86           BEQ  ENDE
0376: E6 CF     87           INC  IND+1
0378: A9 D3     88           LDA  #"S"
037A: 20 ED FD  89           JSR  COUT
037D: A9 BA     90           LDA  #":"
037F: 20 ED FD  91           JSR  COUT
0382: CE BD 03  92           DEC  COUNTER
0385: AD BD 03  93           LDA  COUNTER
0388: 20 DA FD  94           JSR  PRBYTE
038B: 20 8E FD  95           JSR  CROUT
038E: AD 00 C0  96           LDA  $C000
0391: 10 14     97           BPL  LOOP1
0393: C9 9B     98           CMP  #$9B       ;ESC
0395: D0 01     99           BNE  SPACE
0397: 60        100 ENDE     RTS
0398: C9 A0     101 SPACE    CMP  #$A0
039A: D0 0B     102          BNE  LOOP1
039C: 2C 10 C0  103          BIT  $C010
039F: AD 00 C0  104 WAIT     LDA  $C000
03A2: 10 FB     105          BPL  WAIT
03A4: 2C 10 C0  106          BIT  $C010
03A7: B1 CE     107 LOOP1    LDA  (IND),Y
03A9: 09 80     108          ORA  #$80
03AB: C9 A0     109          CMP  #$A0
03AD: B0 05     110          BCS  DRUCK
03AF: 69 40     111          ADC  #$40
03B1: 20 80 FE  112          JSR  SETINV
03B4: 20 ED FD  113 DRUCK    JSR  COUT
03B7: 20 84 FE  114          JSR  SETNORM
03BA: 4C 62 03  115          JMP  LOOP
03BD: 00        116 COUNTER  HEX  00
```

6.2.2. Dateileseprogramm

FILE.LESER

```
100  REM *** FILE.LESER ***
110  PRINT  CHR$ (21): IF  PEEK (36864) = 76 AND
     PEEK (36865) = 26 AND  PEEK (36866) = 144 THEN 140
120  PRINT : PRINT  CHR$ (4)"MAXFILES 3":
     CLEAR : HIMEM: 4096
130  PRINT : PRINT  CHR$ (4)"BLOAD FILE.LESER.OBJ"
140  TEXT : HOME : INVERSE : PRINT "* FILE-LESER  *":
     PRINT "* U.STIEHL/85 *": NORMAL : PRINT :
     PRINT "START J/N ";
150  GET X$: ON X$ = "J" GOTO 160:
     ON X$ <  > "N" GOTO 150: PRINT : END
160  HOME : INVERSE : PRINT "DISKETTE EINLEGEN": NORMAL:
     PRINT : PRINT "W = WEITER ";
170  GET X$: IF X$ <  > "W" THEN 170
180  HOME : PRINT : PRINT  CHR$ (4)"CATALOG":
     INPUT "DATEI:";X$: IF X$ = "" GOTO 140
190  HOME : INVERSE : PRINT "ESC=ENDE * SPACE=STOPP":
     NORMAL : PRINT
200  PRINT  CHR$ (4)"UNLOCK";X$
210  PR# 0: IN# 0: CALL 36864: REM $9000
220  CALL 1002: PRINT : PRINT "1=ERNEUT * 0=ENDE ";
230  GET X$: ON X$ = "1" GOTO 140: IF X$ <  > "0" THEN 230
```

FILE.LESER.OBJ

```
BSAVE FILE.LESER.OBJ, A$9000, L347

                1             ORG  36864      ;$9000
                2    *
9000: 4C 1A 90  3             JMP  INITIAL
                4    *
                5    * FILE.LESER
                6    * ==========
                7    *
                8    IND1     EQU  $CE        ;$CF
                9    IND2     EQU  $FE        ;$FF
                10   TEXT     EQU  $FB2F
                11   HOME     EQU  $FC58
                12   PRINT    EQU  $FDED
                13   HEXOUT   EQU  $FDDA
                14   *
                15   * Max. 5 TSL = 5 * 122 * 2 = 1220 = $04C4
                16   TOTALTSL EQU  $8B00      ;-$8FC4
```

```
                    17  *
                    18  * Macropuffer = 122 Sektoren
                    19  *
                    20  MACPUFF  EQU  $1000       ;-$8A00
                    21  *
                    22  * Maxfiles muß 3 sein!
                    23  *
                    24  TSLPUFF  EQU  $9700       ;Anfang
                    25  TSLPTR   EQU  $9701       ;Pointer
                    26  TSL      EQU  $970C       ;Eig.TSL
                    27  RWTS     EQU  $03D9
                    28  RWTSLOCO EQU  $03E3
                    29  DOSWARM  EQU  $03D0
                    30  *
                    31  *        IOB-Block
                    32  *
9003: 01            33  IOB      HEX  01          ;immer
9004: 60            34  SLOT     HEX  60
9005: 01            35  DRIVE    HEX  01
9006: 00            36  VOLUME   HEX  00
9007: 00            37  TRACK    HEX  00
9008: 00            38  SECTOR   HEX  00
9009: 14            39  DCTLOW   DFB  #<DCT
900A: 90            40  DCTHIGH  DFB  #>DCT
900B: 00            41  BUFLOW   HEX  00
900C: 00            42  BUFHIGH  HEX  00
900D: 00            43           HEX  00          ;unbenutzt
900E: 00            44           HEX  00          ;Count:0
900F: 01            45  COMMAND  HEX  01          ;Read
9010: 00            46  DOSERROR HEX  00
9011: 00            47  EFFVOL   HEX  00
9012: 60            48  VORSLOT  HEX  60
9013: 01            49  VORDRIVE HEX  01
                    50  *
                    51  *        DCT
                    52  *
9014: 00 01 EF      53  DCT      HEX  0001EFD8    ;immer
9017: D8
                    54  *
9018: 00            55  YSAVER   HEX  00
9019: 00            56  XSAVER   HEX  00
                    57  *
                    58  * Aktiven Slot/Drive ermitteln
                    59  *
901A: 20 E3 03      60  INITIAL  JSR  RWTSLOCO
901D: 84 CE         61           STY  IND1        ;Low-IOB
901F: 85 CF         62           STA  IND1+1      ;High-IOB
9021: A0 01         63           LDY  #1
9023: B1 CE         64           LDA  (IND1),Y
9025: 8D 04 90      65           STA  SLOT
9028: C8            66           INY
9029: B1 CE         67           LDA  (IND1),Y
902B: 8D 05 90      68           STA  DRIVE
902E: 2C 10 C0      69           BIT  $C010
```

```
                    70  *
                    71  * Total-TSL initialisieren
                    72  *
9031: A9 00         73          LDA  #<TOTALTSL
9033: 85 CE         74          STA  IND1
9035: A9 8B         75          LDA  #>TOTALTSL
9037: 85 CF         76          STA  IND1+1
9039: A9 00         77          LDA  #<TSLPUFF
903B: 8D 0B 90      78          STA  BUFLOW
903E: A9 97         79          LDA  #>TSLPUFF
9040: 8D 0C 90      80          STA  BUFHIGH
9043: 4C 5D 90      81          JMP  SSTORE1    ;erste
9046: AD 01 97      82  NEXTTSL LDA  TSLPTR
9049: F0 29         83          BEQ  SSTORE4    ;letzte
904B: 8D 07 90      84          STA  TRACK
904E: AD 02 97      85          LDA  TSLPTR+1
9051: 8D 08 90      86          STA  SECTOR
9054: A9 90         87          LDA  #>IOB
9056: A0 03         88          LDY  #<IOB
9058: 20 D9 03      89          JSR  RWTS
905B: B0 1C         90          BCS  ERROR1
905D: A2 00         91  SSTORE1 LDX  #$00
905F: A0 00         92          LDY  #$00
9061: BD 0C 97      93  SSTORE2 LDA  TSL,X
9064: 91 CE         94          STA  (IND1),Y
9066: E6 CE         95          INC  IND1
9068: D0 02         96          BNE  SSTORE3
906A: E6 CF         97          INC  IND1+1
906C: E8            98  SSTORE3 INX
906D: E0 F4         99          CPX  #244       ;122*2
906F: D0 F0        100          BNE  SSTORE2    ;max.
9071: 4C 46 90     101          JMP  NEXTTSL
9074: 91 CE        102  SSTORE4 STA  (IND1),Y   ;Endmarker 0
9076: 4C E7 90     103          JMP  BATCH0
                   104  *
                   105  * Lesefehler
                   106  *
9079: 20 2F FB     107  ERROR1  JSR  TEXT
907C: 20 58 FC     108          JSR  HOME
907F: AD 10 90     109          LDA  DOSERROR
9082: 20 DA FD     110          JSR  HEXOUT
9085: 4C D0 03     111          JMP  DOSWARM
                   112  *
                   113  * Löschen $1000-$8AFF
                   114  *
9088: A9 00        115  CLEAR0  LDA  #<MACPUFF
908A: 85 FE        116          STA  IND2
908C: A9 10        117          LDA  #>MACPUFF
908E: 85 FF        118          STA  IND2+1
9090: A9 00        119          LDA  #0
9092: A8           120          TAY
9093: 91 FE        121  CLEAR1  STA  (IND2),Y
9095: C8           122          INY
9096: D0 FB        123          BNE  CLEAR1
```

```
9098: E6 FF     124           INC  IND2+1
909A: A6 FF     125           LDX  IND2+1
909C: E0 8B     126           CPX  #>TOTALTSL
909E: D0 F3     127           BNE  CLEAR1
90A0: 60        128           RTS
                129 *
                130 * 122 Sektoren anzeigen
                131 *
90A1: A9 00     132 PRINT0    LDA  #0
90A3: 85 FE     133           STA  IND2
90A5: A9 10     134           LDA  #>MACPUFF
90A7: 85 FF     135           STA  IND2+1
90A9: A0 00     136           LDY  #0
90AB: B1 FE     137 PRINT1    LDA  (IND2),Y
90AD: 09 80     138           ORA  #$80
90AF: C9 8D     139           CMP  #$8D
90B1: F0 07     140           BEQ  PRINT2
90B3: C9 A0     141           CMP  #$A0
90B5: B0 03     142           BCS  PRINT2
90B7: 38        143           SEC
90B8: E9 80     144           SBC  #$80
90BA: 20 ED FD  145 PRINT2    JSR  PRINT
90BD: AD 00 C0  146           LDA  $C000
90C0: 10 13     147           BPL  PRINT3     ;keine Taste
90C2: 2C 10 C0  148           BIT  $C010
90C5: C9 9B     149           CMP  #$9B       ;ESC
90C7: F0 1B     150           BEQ  PRINT4
90C9: C9 A0     151           CMP  #$A0       ;Leertaste
90CB: D0 08     152           BNE  PRINT3
90CD: AD 00 C0  153 WAIT      LDA  $C000
90D0: 10 FB     154           BPL  WAIT
90D2: 2C 10 C0  155           BIT  $C010
90D5: C8        156 PRINT3    INY
90D6: D0 D3     157           BNE  PRINT1
90D8: E6 FF     158           INC  IND2+1
90DA: A5 FF     159           LDA  IND2+1
90DC: CD 0C 90  160           CMP  BUFHIGH
90DF: 90 CA     161           BCC  PRINT1
90E1: F0 C8     162           BEQ  PRINT1
90E3: 60        163           RTS             ;zurück
90E4: 68        164 PRINT4    PLA
90E5: 68        165           PLA
90E6: 60        166           RTS             ;Ende
                167 *
                168 * 122 Sektoren lesen
                169 *
90E7: A0 00     170 BATCH0    LDY  #0
90E9: 8C 18 90  171           STY  YSAVER
90EC: A9 00     172           LDA  #<TOTALTSL
90EE: 85 CE     173           STA  IND1
90F0: A9 8B     174           LDA  #>TOTALTSL
90F2: 85 CF     175           STA  IND1+1
                176 *
                177 * Äußerer Loop
                178 *
```

```
90F4: AC 18 90  179 BATCH1  LDY  YSAVER
90F7: B1 CE     180         LDA  (IND1),Y
90F9: D0 05     181         BNE  BATCH2
90FB: A9 00     182         LDA  #0
90FD: 85 48     183         STA  $48        ;P-Register
90FF: 60        184         RTS             ;Ende
                185 *
                186 * Batch lesen
                187 *
9100: A2 00     188 BATCH2  LDX  #0
9102: 8E 19 90  189         STX  XSAVER
9105: A9 00     190         LDA  #<MACPUFF
9107: 8D 0B 90  191         STA  BUFLOW
910A: A9 10     192         LDA  #>MACPUFF
910C: 8D 0C 90  193         STA  BUFHIGH
910F: CE 0C 90  194         DEC  BUFHIGH
9112: 20 88 90  195         JSR  CLEAR0
                196 *
                197 * Innerer Loop
                198 *
9115: AC 18 90  199 BATCH3  LDY  YSAVER
9118: B1 CE     200         LDA  (IND1),Y
911A: F0 35     201         BEQ  BATCH7
911C: EE 0C 90  202         INC  BUFHIGH
911F: 8D 07 90  203         STA  TRACK
9122: EE 18 90  204         INC  YSAVER
9125: AC 18 90  205         LDY  YSAVER
9128: D0 02     206         BNE  BATCH4
912A: E6 CF     207         INC  IND1+1
912C: B1 CE     208 BATCH4  LDA  (IND1),Y
912E: 8D 08 90  209         STA  SECTOR
9131: A9 90     210         LDA  #>IOB
9133: A0 03     211         LDY  #<IOB
9135: 20 D9 03  212         JSR  RWTS
9138: 90 01     213         BCC  BATCH5
                214 *
                215 * Hier ggf. RTS einfügen
                216 *
913A: EA        217         NOP             ;Fehler
913B: EE 18 90  218 BATCH5  INC  YSAVER
913E: AC 18 90  219         LDY  YSAVER
9141: D0 02     220         BNE  BATCH6
9143: E6 CF     221         INC  IND1+1
9145: AE 19 90  222 BATCH6  LDX  XSAVER
9148: E8        223         INX
9149: E8        224         INX
914A: 8E 19 90  225         STX  XSAVER
914D: E0 F4     226         CPX  #244
914F: 90 C4     227         BCC  BATCH3
9151: A9 00     228 BATCH7  LDA  #0
9153: 85 48     229         STA  $48        ;P-Register
9155: 20 A1 90  230         JSR  PRINT0
9158: 4C F4 90  231         JMP  BATCH1
```

6.2.3. Diskettenkopierprogramm für 2-Drive-Besitzer

KOPIERPROGRAMM.2DRIVE

```
BSAVE KOPIERPROGRAMM.2DRIVE, A$0BB8, L726
```

```
                  1             ORG  3000
                  2     *
                  3     * Schnelles Kopierprogramm
                  4     * für Apple IIe oder Apple
                  5     * mit 16K-Karte - DOS 3.3
                  6     * von U.Stiehl, 01.11.1983
                  7     *
                  8     * Programm nur für 2 Drives
                  9     * von Drive 1 nach Drive 2
                 10     * Wahlweise mit oder ohne
                 11     * Initialisierung der Kopie
                 12     *
0BB8: 4C D1 0B   13             JMP  START
                 14     *
                 15     * 0=ohne, 1=mit Init
                 16     *
0BBB: 01         17     INIT     HEX  01          ;3000+3
                 18     *
                 19     IND1     EQU  $CE         ;+$CF
                 20     DOSCOLD  EQU  $3D3
                 21     RWTS     EQU  $3D9
                 22     RWTSLOCO EQU  $3E3
                 23     TEXT     EQU  $FB2F
                 24     HOME     EQU  $FC58
                 25     HEXOUT   EQU  $FDDA
                 26     PRINT    EQU  $FDF0
                 27     PUFFER0  EQU  $1000
                 28     PUFFER1  EQU  $D000       ;BANK1
                 29     PUFFER2  EQU  $D000       ;BANK2
                 30     *
                 31     *        IOB-Block
                 32     *
0BBC: 01         33     IOB      HEX  01          ;IMMER
0BBD: 60         34     SLOT     HEX  60
0BBE: 01         35     DRIVE    HEX  01
0BBF: 00         36     VOLUME   HEX  00
0BC0: 00         37     TRACK    HEX  00
0BC1: 00         38     SECTOR   HEX  00
0BC2: CD         39     DCTLOW   DFB  #<DCT
0BC3: 0B         40     DCTHIGH  DFB  #>DCT
0BC4: 00         41     BUFLOW   HEX  00
0BC5: 00         42     BUFHIGH  HEX  00
0BC6: 00         43              HEX  00          ;UNUSED
0BC7: 00         44              HEX  00          ;COUNT:0
```

```
0BC8: 01          45   COMMAND  HEX  01          ;READ
0BC9: 00          46   DOSERROR HEX  00
0BCA: 00          47   VORVOL   HEX  00
0BCB: 60          48   VORSLOT  HEX  60
0BCC: 01          49   VORDRIVE HEX  01
                  50   *
                  51   *        DCT
                  52   *
0BCD: 00 01 EF    53   DCT      HEX  0001EFD8  ;IMMER
0BD0: D8
                  54   *
                  55   * Start J/N
                  56   *
0BD1: A9 15       57   START    LDA  #$15      ;Ctrl-U
0BD3: 20 F0 FD    58            JSR  PRINT
0BD6: 20 2F FB    59            JSR  TEXT
0BD9: 20 58 FC    60            JSR  HOME
0BDC: A2 00       61            LDX  #0
0BDE: BD 2D 0E    62   MENU0    LDA  MENU1,X
0BE1: F0 06       63            BEQ  KEY0
0BE3: 20 F0 FD    64            JSR  PRINT
0BE6: E8          65            INX
0BE7: D0 F5       66            BNE  MENU0
0BE9: 2C 10 C0    67   KEY0     BIT  $C010
0BEC: AD 00 C0    68   KEY1     LDA  $C000
0BEF: 10 FB       69            BPL  KEY1
0BF1: 2C 10 C0    70            BIT  $C010
0BF4: C9 CA       71            CMP  #"J"
0BF6: F0 12       72            BEQ  SLDR
0BF8: C9 EA       73            CMP  #"j"
0BFA: F0 0E       74            BEQ  SLDR
0BFC: C9 CE       75            CMP  #"N"
0BFE: F0 07       76            BEQ  NEIN
0C00: C9 EE       77            CMP  #"n"
0C02: F0 03       78            BEQ  NEIN
0C04: 4C EC 0B    79            JMP  KEY1
0C07: 4C 81 0D    80   NEIN     JMP  EXIT
                  81   *
                  82   * Slot + Drive ermitteln
                  83   *
0C0A: 20 58 FC    84   SLDR     JSR  HOME
0C0D: A9 43       85            LDA  #"C"
0C0F: 20 F0 FD    86            JSR  PRINT
0C12: 20 E3 03    87            JSR  RWTSLOC0
0C15: 84 CE       88            STY  IND1      ;LOWIOB
0C17: 85 CF       89            STA  IND1+1    ;HIGHIOB
0C19: A0 01       90            LDY  #$01
0C1B: B1 CE       91            LDA  (IND1),Y  ;VORSLOT
0C1D: 8D BD 0B    92            STA  SLOT
0C20: 8D CB 0B    93            STA  VORSLOT
0C23: A0 02       94            LDY  #$02
0C25: B1 CE       95            LDA  (IND1),Y  ;VORDRI.
0C27: 8D BE 0B    96            STA  DRIVE
0C2A: 8D CC 0B    97            STA  VORDRIVE
```

```
0C2D: 20 0F 0E  98          JSR  DRIVE1
                99  *
                100 * Volume Original ermitteln
                101 *
0C30: A0 0E     102         LDY  #$0E
0C32: B1 CE     103         LDA  (IND1),Y   ;VORVOL.
0C34: 8D CA 0B  104         STA  VORVOL
0C37: A9 00     105         LDA  #0
0C39: 8D BF 0B  106         STA  VOLUME
0C3C: A9 22     107         LDA  #34
0C3E: 8D C0 0B  108         STA  TRACK
0C41: A9 0F     109         LDA  #15
0C43: 8D C1 0B  110         STA  SECTOR
0C46: 20 9D 0D  111         JSR  PUF0
0C49: 20 01 0E  112         JSR  RWTS1
0C4C: AD CA 0B  113         LDA  VORVOL
0C4F: 8D BF 0B  114         STA  VOLUME
                115 *
                116 * Read Tracks 34-27, 26, 25-23
                117 *
0C52: 20 0F 0E  118         JSR  DRIVE1
0C55: A9 22     119         LDA  #34
0C57: 8D C0 0B  120         STA  TRACK
0C5A: A9 08     121         LDA  #8
0C5C: 8D E6 0D  122         STA  TCOUNT
0C5F: 20 9D 0D  123         JSR  PUF0
0C62: 20 CA 0D  124         JSR  BLOCK0
                125 *
0C65: A9 1A     126         LDA  #26
0C67: 8D C0 0B  127         STA  TRACK
0C6A: A9 01     128         LDA  #1
0C6C: 8D E6 0D  129         STA  TCOUNT
0C6F: 20 A8 0D  130         JSR  PUF1
0C72: 20 CA 0D  131         JSR  BLOCK0
                132 *
0C75: A9 19     133         LDA  #25
0C77: 8D C0 0B  134         STA  TRACK
0C7A: A9 03     135         LDA  #3
0C7C: 8D E6 0D  136         STA  TCOUNT
0C7F: 20 B9 0D  137         JSR  PUF2
0C82: 20 CA 0D  138         JSR  BLOCK0
                139 *
                140 * Initialisieren
                141 *
0C85: AD BB 0B  142 INITJA  LDA  INIT
0C88: F0 0B     143         BEQ  INITNEIN
0C8A: 20 1E 0E  144         JSR  DRIVE2
0C8D: A9 04     145         LDA  #$04       ;INIT
0C8F: 8D C8 0B  146         STA  COMMAND
0C92: 20 01 0E  147         JSR  RWTS1
0C95: EA        148 INITNEIN NOP
                149 *
                150 * Write Tracks 34-27, 26, 25-23
                151 *
```

```
0C96: 20 1E 0E  152          JSR  DRIVE2
0C99: A9 22     153          LDA  #34
0C9B: 8D C0 0B  154          STA  TRACK
0C9E: A9 08     155          LDA  #8
0CA0: 8D E6 0D  156          STA  TCOUNT
0CA3: 20 9D 0D  157          JSR  PUF0
0CA6: 20 CA 0D  158          JSR  BLOCK0
                159  *
0CA9: A9 1A     160          LDA  #26
0CAB: 8D C0 0B  161          STA  TRACK
0CAE: A9 01     162          LDA  #1
0CB0: 8D E6 0D  163          STA  TCOUNT
0CB3: 20 A8 0D  164          JSR  PUF1
0CB6: 20 CA 0D  165          JSR  BLOCK0
                166  *
0CB9: A9 19     167          LDA  #25
0CBB: 8D C0 0B  168          STA  TRACK
0CBE: A9 03     169          LDA  #3
0CC0: 8D E6 0D  170          STA  TCOUNT
0CC3: 20 B9 0D  171          JSR  PUF2
0CC6: 20 CA 0D  172          JSR  BLOCK0
                173  *
                174  * Read Tracks 22-15, 14, 13-11
                175  *
0CC9: 20 0F 0E  176          JSR  DRIVE1
0CCC: A9 16     177          LDA  #22
0CCE: 8D C0 0B  178          STA  TRACK
0CD1: A9 08     179          LDA  #8
0CD3: 8D E6 0D  180          STA  TCOUNT
0CD6: 20 9D 0D  181          JSR  PUF0
0CD9: 20 CA 0D  182          JSR  BLOCK0
                183  *
0CDC: A9 0E     184          LDA  #14
0CDE: 8D C0 0B  185          STA  TRACK
0CE1: A9 01     186          LDA  #1
0CE3: 8D E6 0D  187          STA  TCOUNT
0CE6: 20 A8 0D  188          JSR  PUF1
0CE9: 20 CA 0D  189          JSR  BLOCK0
                190  *
0CEC: A9 0D     191          LDA  #13
0CEE: 8D C0 0B  192          STA  TRACK
0CF1: A9 03     193          LDA  #3
0CF3: 8D E6 0D  194          STA  TCOUNT
0CF6: 20 B9 0D  195          JSR  PUF2
0CF9: 20 CA 0D  196          JSR  BLOCK0
                197  *
                198  * Write Tracks 22-15, 14, 13-11
                199  *
0CFC: 20 1E 0E  200          JSR  DRIVE2
0CFF: A9 16     201          LDA  #22
0D01: 8D C0 0B  202          STA  TRACK
0D04: A9 08     203          LDA  #8
0D06: 8D E6 0D  204          STA  TCOUNT
0D09: 20 9D 0D  205          JSR  PUF0
```

```
0D0C: 20 CA 0D  206          JSR  BLOCK0
                207  *
0D0F: A9 0E     208          LDA  #14
0D11: 8D C0 0B  209          STA  TRACK
0D14: A9 01     210          LDA  #1
0D16: 8D E6 0D  211          STA  TCOUNT
0D19: 20 A8 0D  212          JSR  PUF1
0D1C: 20 CA 0D  213          JSR  BLOCK0
                214  *
0D1F: A9 0D     215          LDA  #13
0D21: 8D C0 0B  216          STA  TRACK
0D24: A9 03     217          LDA  #3
0D26: 8D E6 0D  218          STA  TCOUNT
0D29: 20 B9 0D  219          JSR  PUF2
0D2C: 20 CA 0D  220          JSR  BLOCK0
                221  *
                222  * Read Tracks 10-3, 2-0
                223  *
0D2F: 20 0F 0E  224          JSR  DRIVE1
0D32: A9 0A     225          LDA  #10
0D34: 8D C0 0B  226          STA  TRACK
0D37: A9 08     227          LDA  #8
0D39: 8D E6 0D  228          STA  TCOUNT
0D3C: 20 9D 0D  229          JSR  PUF0
0D3F: 20 CA 0D  230          JSR  BLOCK0
                231  *
0D42: A9 02     232          LDA  #2
0D44: 8D C0 0B  233          STA  TRACK
0D47: A9 03     234          LDA  #3
0D49: 8D E6 0D  235          STA  TCOUNT
0D4C: 20 B9 0D  236          JSR  PUF2
0D4F: 20 CA 0D  237          JSR  BLOCK0
                238  *
                239  * Write Tracks 10-3, 2-0
                240  *
0D52: 20 1E 0E  241          JSR  DRIVE2
0D55: A9 0A     242          LDA  #10
0D57: 8D C0 0B  243          STA  TRACK
0D5A: A9 08     244          LDA  #8
0D5C: 8D E6 0D  245          STA  TCOUNT
0D5F: 20 9D 0D  246          JSR  PUF0
0D62: 20 CA 0D  247          JSR  BLOCK0
                248  *
0D65: A9 02     249          LDA  #2
0D67: 8D C0 0B  250          STA  TRACK
0D6A: A9 03     251          LDA  #3
0D6C: 8D E6 0D  252          STA  TCOUNT
0D6F: 20 B9 0D  253          JSR  PUF2
0D72: 20 CA 0D  254          JSR  BLOCK0
                255  *
0D75: AD 81 C0  256          LDA  $C081
0D78: AD 81 C0  257          LDA  $C081
0D7B: 20 0F 0E  258          JSR  DRIVE1
0D7E: 4C D1 0B  259          JMP  START
```

```
                    260  *
                    261  * Exit
                    262  *
0D81: AD 81 C0      263  EXIT     LDA  $C081
0D84: AD 81 C0      264          LDA  $C081      ;RDROM
0D87: 20 0F 0E      265          JSR  DRIVE1
0D8A: 20 58 FC      266          JSR  HOME
0D8D: A2 00         267          LDX  #0
0D8F: BD 77 0E      268  EXIT1    LDA  MENU2,X
0D92: F0 06         269          BEQ  EXIT2
0D94: 20 F0 FD      270          JSR  PRINT
0D97: E8            271          INX
0D98: D0 F5         272          BNE  EXIT1
0D9A: 4C D3 03      273  EXIT2    JMP  DOSCOLD
                    274  *
                    275  * PUF0: 1000-8FFF
                    276  *
0D9D: A9 00         277  PUF0     LDA  #<PUFFER0
0D9F: 8D C4 0B      278          STA  BUFLOW
0DA2: A9 10         279          LDA  #>PUFFER0
0DA4: 8D C5 0B      280          STA  BUFHIGH
0DA7: 60            281          RTS
                    282  *
                    283  * PUF1: D000-DFFF - Bank 1
                    284  *
0DA8: A9 00         285  PUF1     LDA  #<PUFFER1
0DAA: 8D C4 0B      286          STA  BUFLOW
0DAD: A9 D0         287          LDA  #>PUFFER1
0DAF: 8D C5 0B      288          STA  BUFHIGH
0DB2: AD 8B C0      289          LDA  $C08B
0DB5: AD 8B C0      290          LDA  $C08B      ;RD/WR
0DB8: 60            291          RTS
                    292  *
                    293  * PUF2: D000-FFFF - Bank 2
                    294  *
0DB9: A9 00         295  PUF2     LDA  #<PUFFER2
0DBB: 8D C4 0B      296          STA  BUFLOW
0DBE: A9 D0         297          LDA  #>PUFFER2
0DC0: 8D C5 0B      298          STA  BUFHIGH
0DC3: AD 83 C0      299          LDA  $C083
0DC6: AD 83 C0      300          LDA  $C083      ;RD/WR
0DC9: 60            301          RTS
                    302  *
                    303  * Block-Transfer
                    304  *
0DCA: A9 0F         305  BLOCK0   LDA  #$0F
0DCC: 8D C1 0B      306          STA  SECTOR
0DCF: 20 01 0E      307  BLOCK1   JSR  RWTS1
0DD2: EE C5 0B      308          INC  BUFHIGH
0DD5: CE C1 0B      309          DEC  SECTOR
0DD8: AD C1 0B      310          LDA  SECTOR
0DDB: 10 F2         311          BPL  BLOCK1
0DDD: CE C0 0B      312          DEC  TRACK
0DE0: CE E6 0D      313          DEC  TCOUNT
```

```
0DE3: D0 E5      314           BNE  BLOCK0
0DE5: 60         315           RTS
0DE6: 00         316  TCOUNT   HEX  00
                 317  *
                 318  * Lesefehler
                 319  *
0DE7: 68         320  ERROR    PLA
0DE8: 68         321           PLA
0DE9: AD 81 C0   322           LDA  $C081
0DEC: AD 81 C0   323           LDA  $C081       ;RDROM
0DEF: 20 2F FB   324           JSR  TEXT
0DF2: 20 58 FC   325           JSR  HOME
0DF5: AD C9 0B   326           LDA  DOSERROR
0DF8: 20 DA FD   327           JSR  HEXOUT
0DFB: 20 0F 0E   328           JSR  DRIVE1
0DFE: 4C D3 03   329           JMP  DOSCOLD
                 330  *
                 331  * RWTS
                 332  *
0E01: A0 BC      333  RWTS1    LDY  #<IOB
0E03: A9 0B      334           LDA  #>IOB
0E05: 20 D9 03   335           JSR  RWTS
0E08: B0 DD      336           BCS  ERROR
0E0A: A9 00      337           LDA  #0
0E0C: 85 48      338           STA  $48
0E0E: 60         339           RTS
                 340  *
                 341  * Drive 1 Read
                 342  *
0E0F: AD BE 0B   343  DRIVE1   LDA  DRIVE
0E12: 8D CC 0B   344           STA  VORDRIVE
0E15: A9 01      345           LDA  #1
0E17: 8D BE 0B   346           STA  DRIVE
0E1A: 8D C8 0B   347           STA  COMMAND     ;READ
0E1D: 60         348           RTS
                 349  *
                 350  * Drive 2 Write
                 351  *
0E1E: AD BE 0B   352  DRIVE2   LDA  DRIVE
0E21: 8D CC 0B   353           STA  VORDRIVE
0E24: A9 02      354           LDA  #2
0E26: 8D BE 0B   355           STA  DRIVE
0E29: 8D C8 0B   356           STA  COMMAND     ;WRITE
0E2C: 60         357           RTS
0E2D: 2A 2A 20   358  MENU1    INV  "** COPY.IIE **"
0E30: 03 0F 10 19 2E 09 09 05
0E38: 20 2A 2A
0E3B: 8D         359           HEX  8D
0E3C: 03 31 39   360           INV  "C1984 U.STIEHL"
0E3F: 38 34 20 15 2E 13 14 09
0E47: 05 08 0C
0E4A: 8D 8D      361           HEX  8D8D
0E4C: 0F 12 09   362           INV  "ORIGINAL  -> 1"
0E4F: 07 09 0E 01 0C 20 20 2D
```

```
0E57: 3E 20 31
0E5A: 8D        363           HEX  8D
0E5B: 04 15 10  364           INV  "DUPLIKAT  -> 2"
0E5E: 0C 09 0B 01 14 20 20 2D
0E66: 3E 20 32
0E69: 8D 8D     365           HEX  8D8D
0E6B: 13 14 01  366           INV  "START J/N"
0E6E: 12 14 20 0A 2F 0E
0E74: A0        367           ASC  " "
0E75: 7F        368           FLS  "?"
0E76: 00        369           HEX  00
0E77: CE C5 D5  370  MENU2    ASC  "NEUSTART MIT "
0E7A: D3 D4 C1 D2 D4 A0 CD C9
0E82: D4 A0
0E84: C3 C1 CC  371           ASC  "CALL 3000"
0E87: CC A0 B3 B0 B0 B0
0E8D: 00        372           HEX  00
```

6.2.4. Diskettenkopierprogramm für 1-Drive-Besitzer

KOPIERPROGRAMM.1DRIVE

```
BSAVE KOPIERPROGRAMM.1DRIVE, A$0BB8, L864

              1             ORG  3000
              2    *
              3    * Schnelles Kopierprogramm
              4    * für Apple IIe oder Apple
              5    * mit 16K-Karte - DOS 3.3
              6    * von U.Stiehl, 01.11.1983
              7    *
              8    * 1-Drive-Version
              9    *
0BB8: 4C D1 0B  10            JMP  START
             11    *
             12    * 0 = ohne, 1 = mit Init
             13    *
0BBB: 01     14    INIT     HEX  01          ;3000+3
             15    *
             16    IND1     EQU  $CE         ;+$CF
             17    DOSCOLD  EQU  $3D3
             18    RWTS     EQU  $3D9
             19    RWTSLOCO EQU  $3E3
             20    TEXT     EQU  $FB2F
             21    HOME     EQU  $FC58
```

```
                    22  HEXOUT   EQU  $FDDA
                    23  PRINT    EQU  $FDF0
                    24  PUFFER0  EQU  $1000
                    25  PUFFER1  EQU  $D000      ;BANK1
                    26  PUFFER2  EQU  $D000      ;BANK2
                    27  *
                    28  *        IOB-BLOCK
                    29  *
0BBC: 01            30  IOB      HEX  01         ;IMMER
0BBD: 60            31  SLOT     HEX  60
0BBE: 01            32  DRIVE    HEX  01
0BBF: 00            33  VOLUME   HEX  00
0BC0: 00            34  TRACK    HEX  00
0BC1: 00            35  SECTOR   HEX  00
0BC2: CD            36  DCTLOW   DFB  #<DCT
0BC3: 0B            37  DCTHIGH  DFB  #>DCT
0BC4: 00            38  BUFLOW   HEX  00
0BC5: 00            39  BUFHIGH  HEX  00
0BC6: 00            40           HEX  00         ;UNUSED
0BC7: 00            41           HEX  00         ;COUNT:0
0BC8: 01            42  COMMAND  HEX  01         ;READ
0BC9: 00            43  DOSERROR HEX  00
0BCA: 00            44  VORVOL   HEX  00
0BCB: 60            45  VORSLOT  HEX  60
0BCC: 01            46  VORDRIVE HEX  01
                    47  *
                    48  *        DCT
                    49  *
0BCD: 00 01 EF      50  DCT      HEX  0001EFD8   ;IMMER
0BD0: D8
                    51  *
                    52  * Start J/N
                    53  *
0BD1: 20 2F FB      54  START    JSR  TEXT
0BD4: 20 58 FC      55           JSR  HOME
0BD7: A2 00         56           LDX  #0
0BD9: BD 39 0E      57  MENU0    LDA  MENU1,X
0BDC: F0 06         58           BEQ  KEY0
0BDE: 20 F0 FD      59           JSR  PRINT
0BE1: E8            60           INX
0BE2: D0 F5         61           BNE  MENU0
0BE4: 2C 10 C0      62  KEY0     BIT  $C010
0BE7: AD 00 C0      63  KEY1     LDA  $C000
0BEA: 10 FB         64           BPL  KEY1
0BEC: 2C 10 C0      65           BIT  $C010
0BEF: C9 CA         66           CMP  #"J"
0BF1: F0 12         67           BEQ  SLDR
0BF3: C9 EA         68           CMP  #"j"
0BF5: F0 0E         69           BEQ  SLDR
0BF7: C9 CE         70           CMP  #"N"
0BF9: F0 07         71           BEQ  NEIN
0BFB: C9 EE         72           CMP  #"n"
0BFD: F0 03         73           BEQ  NEIN
0BFF: 4C E7 0B      74           JMP  KEY1
```

```
0C02: 4C 8E 0D  75   NEIN     JMP  EXIT1A
                76   *
                77   * Slot + Drive ermitteln
                78   *
0C05: 20 58 FC  79   SLDR     JSR  HOME
0C08: 20 E3 03  80            JSR  RWTSLOCO
0C0B: 84 CE     81            STY  IND1        ;LOWIOB
0C0D: 85 CF     82            STA  IND1+1      ;HIGHIOB
0C0F: A0 01     83            LDY  #$01
0C11: B1 CE     84            LDA  (IND1),Y    ;VORSLOT
0C13: 8D BD 0B  85            STA  SLOT
0C16: 8D CB 0B  86            STA  VORSLOT
0C19: A0 02     87            LDY  #$02
0C1B: B1 CE     88            LDA  (IND1),Y    ;VORDRI.
0C1D: 8D BE 0B  89            STA  DRIVE
0C20: 8D CC 0B  90            STA  VORDRIVE
0C23: 20 13 0E  91            JSR  DRIVE1
                92   *
                93   * Volume Original ermitteln
                94   *
0C26: A0 0E     95            LDY  #$0E
0C28: B1 CE     96            LDA  (IND1),Y    ;VORVOL.
0C2A: 8D CA 0B  97            STA  VORVOL
0C2D: A9 00     98            LDA  #0
0C2F: 8D BF 0B  99            STA  VOLUME
0C32: A9 22     100           LDA  #34
0C34: 8D C0 0B  101           STA  TRACK
0C37: A9 0F     102           LDA  #15
0C39: 8D C1 0B  103           STA  SECTOR
0C3C: 20 A1 0D  104           JSR  PUF0
0C3F: 20 05 0E  105           JSR  RWTS1
0C42: AD CA 0B  106           LDA  VORVOL
0C45: 8D BF 0B  107           STA  VOLUME
                108  *
                109  * Read Tracks 34-27, 26, 25-23
                110  *
0C48: A9 22     111           LDA  #34
0C4A: 8D C0 0B  112           STA  TRACK
0C4D: A9 08     113           LDA  #8
0C4F: 8D EA 0D  114           STA  TCOUNT
0C52: 20 A1 0D  115           JSR  PUF0
0C55: 20 CE 0D  116           JSR  BLOCK0
                117  *
0C58: A9 1A     118           LDA  #26
0C5A: 8D C0 0B  119           STA  TRACK
0C5D: A9 01     120           LDA  #1
0C5F: 8D EA 0D  121           STA  TCOUNT
0C62: 20 AC 0D  122           JSR  PUF1
0C65: 20 CE 0D  123           JSR  BLOCK0
                124  *
0C68: A9 19     125           LDA  #25
0C6A: 8D C0 0B  126           STA  TRACK
0C6D: A9 03     127           LDA  #3
0C6F: 8D EA 0D  128           STA  TCOUNT
```

```
0C72: 20 BD 0D  129          JSR  PUF2
0C75: 20 CE 0D  130          JSR  BLOCK0
0C78: AD 81 C0  131          LDA  $C081
0C7B: AD 81 C0  132          LDA  $C081
                133  *
                134  * Initialisieren
                135  *
0C7E: 20 25 0E  136  INITJA  JSR  DRIVE2
0C81: AD BB 0B  137          LDA  INIT
0C84: F0 0D     138          BEQ  INITNEIN
0C86: A9 04     139          LDA  #$04       ;INIT
0C88: 8D C8 0B  140          STA  COMMAND
0C8B: 20 05 0E  141          JSR  RWTS1
0C8E: A9 02     142          LDA  #2         ;WRITE
0C90: 8D C8 0B  143          STA  COMMAND
0C93: EA        144  INITNEIN NOP
                145  *
                146  * Write Tracks 34-27, 26, 25-23
                147  *
0C94: A9 22     148          LDA  #34
0C96: 8D C0 0B  149          STA  TRACK
0C99: A9 08     150          LDA  #8
0C9B: 8D EA 0D  151          STA  TCOUNT
0C9E: 20 A1 0D  152          JSR  PUF0
0CA1: 20 CE 0D  153          JSR  BLOCK0
                154  *
0CA4: A9 1A     155          LDA  #26
0CA6: 8D C0 0B  156          STA  TRACK
0CA9: A9 01     157          LDA  #1
0CAB: 8D EA 0D  158          STA  TCOUNT
0CAE: 20 AC 0D  159          JSR  PUF1
0CB1: 20 CE 0D  160          JSR  BLOCK0
                161  *
0CB4: A9 19     162          LDA  #25
0CB6: 8D C0 0B  163          STA  TRACK
0CB9: A9 03     164          LDA  #3
0CBB: 8D EA 0D  165          STA  TCOUNT
0CBE: 20 BD 0D  166          JSR  PUF2
0CC1: 20 CE 0D  167          JSR  BLOCK0
0CC4: AD 81 C0  168          LDA  $C081
0CC7: AD 81 C0  169          LDA  $C081
                170  *
                171  * Read Tracks 22-15, 14, 13-11
                172  *
0CCA: 20 13 0E  173          JSR  DRIVE1
0CCD: A9 16     174          LDA  #22
0CCF: 8D C0 0B  175          STA  TRACK
0CD2: A9 08     176          LDA  #8
0CD4: 8D EA 0D  177          STA  TCOUNT
0CD7: 20 A1 0D  178          JSR  PUF0
0CDA: 20 CE 0D  179          JSR  BLOCK0
                180  *
0CDD: A9 0E     181          LDA  #14
0CDF: 8D C0 0B  182          STA  TRACK
```

```
0CE2: A9 01     183           LDA  #1
0CE4: 8D EA 0D  184           STA  TCOUNT
0CE7: 20 AC 0D  185           JSR  PUF1
0CEA: 20 CE 0D  186           JSR  BLOCK0
                187  *
0CED: A9 0D     188           LDA  #13
0CEF: 8D C0 0B  189           STA  TRACK
0CF2: A9 03     190           LDA  #3
0CF4: 8D EA 0D  191           STA  TCOUNT
0CF7: 20 BD 0D  192           JSR  PUF2
0CFA: 20 CE 0D  193           JSR  BLOCK0
0CFD: AD 81 C0  194           LDA  $C081
0D00: AD 81 C0  195           LDA  $C081
                196  *
                197  * Write Tracks 22-15, 14, 13-11
                198  *
0D03: 20 25 0E  199           JSR  DRIVE2
0D06: A9 16     200           LDA  #22
0D08: 8D C0 0B  201           STA  TRACK
0D0B: A9 08     202           LDA  #8
0D0D: 8D EA 0D  203           STA  TCOUNT
0D10: 20 A1 0D  204           JSR  PUF0
0D13: 20 CE 0D  205           JSR  BLOCK0
                206  *
0D16: A9 0E     207           LDA  #14
0D18: 8D C0 0B  208           STA  TRACK
0D1B: A9 01     209           LDA  #1
0D1D: 8D EA 0D  210           STA  TCOUNT
0D20: 20 AC 0D  211           JSR  PUF1
0D23: 20 CE 0D  212           JSR  BLOCK0
                213  *
0D26: A9 0D     214           LDA  #13
0D28: 8D C0 0B  215           STA  TRACK
0D2B: A9 03     216           LDA  #3
0D2D: 8D EA 0D  217           STA  TCOUNT
0D30: 20 BD 0D  218           JSR  PUF2
0D33: 20 CE 0D  219           JSR  BLOCK0
0D36: AD 81 C0  220           LDA  $C081
0D39: AD 81 C0  221           LDA  $C081
                222  *
                223  * Read Tracks 10-3, 2-0
                224  *
0D3C: 20 13 0E  225           JSR  DRIVE1
0D3F: A9 0A     226           LDA  #10
0D41: 8D C0 0B  227           STA  TRACK
0D44: A9 08     228           LDA  #8
0D46: 8D EA 0D  229           STA  TCOUNT
0D49: 20 A1 0D  230           JSR  PUF0
0D4C: 20 CE 0D  231           JSR  BLOCK0
                232  *
0D4F: A9 02     233           LDA  #2
0D51: 8D C0 0B  234           STA  TRACK
0D54: A9 03     235           LDA  #3
0D56: 8D EA 0D  236           STA  TCOUNT
```

```
0D59: 20 BD 0D  237          JSR  PUF2
0D5C: 20 CE 0D  238          JSR  BLOCK0
0D5F: AD 81 C0  239          LDA  $C081
0D62: AD 81 C0  240          LDA  $C081
                241  *
                242  * Write Tracks 10-3, 2-0
                243  *
0D65: 20 25 0E  244          JSR  DRIVE2
0D68: A9 0A     245          LDA  #10
0D6A: 8D C0 0B  246          STA  TRACK
0D6D: A9 08     247          LDA  #8
0D6F: 8D EA 0D  248          STA  TCOUNT
0D72: 20 A1 0D  249          JSR  PUF0
0D75: 20 CE 0D  250          JSR  BLOCK0
                251  *
0D78: A9 02     252          LDA  #2
0D7A: 8D C0 0B  253          STA  TRACK
0D7D: A9 03     254          LDA  #3
0D7F: 8D EA 0D  255          STA  TCOUNT
0D82: 20 BD 0D  256          JSR  PUF2
0D85: 20 CE 0D  257          JSR  BLOCK0
                258  *
                259  * Exit
                260  *
0D88: AD 81 C0  261  EXIT    LDA  $C081
0D8B: AD 81 C0  262          LDA  $C081      ;RDROM
0D8E: 20 58 FC  263  EXIT1A  JSR  HOME
0D91: A2 00     264          LDX  #0
0D93: BD 84 0E  265  EXIT1   LDA  MENU2,X
0D96: F0 06     266          BEQ  EXIT2
0D98: 20 F0 FD  267          JSR  PRINT
0D9B: E8        268          INX
0D9C: D0 F5     269          BNE  EXIT1
0D9E: 4C D3 03  270  EXIT2   JMP  DOSCOLD
                271  *
                272  * PUF0: 1000-8FFF
                273  *
0DA1: A9 00     274  PUF0    LDA  #<PUFFER0
0DA3: 8D C4 0B  275          STA  BUFLOW
0DA6: A9 10     276          LDA  #>PUFFER0
0DA8: 8D C5 0B  277          STA  BUFHIGH
0DAB: 60        278          RTS
                279  *
                280  * PUF1: D000-DFFF - Bank 1
                281  *
0DAC: A9 00     282  PUF1    LDA  #<PUFFER1
0DAE: 8D C4 0B  283          STA  BUFLOW
0DB1: A9 D0     284          LDA  #>PUFFER1
0DB3: 8D C5 0B  285          STA  BUFHIGH
0DB6: AD 8B C0  286          LDA  $C08B
0DB9: AD 8B C0  287          LDA  $C08B      ;RD/WR
0DBC: 60        288          RTS
                289  *
                290  * PUF2: D000-FFFF - Bank 2
                291  *
```

```
0DBD: A9 00     292  PUF2     LDA  #<PUFFER2
0DBF: 8D C4 0B  293          STA  BUFLOW
0DC2: A9 D0     294          LDA  #>PUFFER2
0DC4: 8D C5 0B  295          STA  BUFHIGH
0DC7: AD 83 C0  296          LDA  $C083
0DCA: AD 83 C0  297          LDA  $C083      ;RD/WR
0DCD: 60        298          RTS
                299  *
                300  * Block-Transfer
                301  *
0DCE: A9 0F     302  BLOCK0   LDA  #$0F
0DD0: 8D C1 0B  303          STA  SECTOR
0DD3: 20 05 0E  304  BLOCK1   JSR  RWTS1
0DD6: EE C5 0B  305          INC  BUFHIGH
0DD9: CE C1 0B  306          DEC  SECTOR
0DDC: AD C1 0B  307          LDA  SECTOR
0DDF: 10 F2     308          BPL  BLOCK1
0DE1: CE C0 0B  309          DEC  TRACK
0DE4: CE EA 0D  310          DEC  TCOUNT
0DE7: D0 E5     311          BNE  BLOCK0
0DE9: 60        312          RTS
0DEA: 00        313  TCOUNT   HEX  00
                314  *
                315  * Lesefehler
                316  *
0DEB: 68        317  ERROR    PLA
0DEC: 68        318          PLA
0DED: AD 81 C0  319          LDA  $C081
0DF0: AD 81 C0  320          LDA  $C081      ;RDROM
0DF3: 20 2F FB  321          JSR  TEXT
0DF6: 20 58 FC  322          JSR  HOME
0DF9: AD C9 0B  323          LDA  DOSERROR
0DFC: 20 DA FD  324          JSR  HEXOUT
0DFF: 20 13 0E  325          JSR  DRIVE1
0E02: 4C D3 03  326          JMP  DOSCOLD
                327  *
                328  * RWTS
                329  *
0E05: A0 BC     330  RWTS1    LDY  #<IOB
0E07: A9 0B     331          LDA  #>IOB
0E09: 20 D9 03  332          JSR  RWTS
0E0C: B0 DD     333          BCS  ERROR
0E0E: A9 00     334          LDA  #0
0E10: 85 48     335          STA  $48
0E12: 60        336          RTS
                337  *
                338  * Drive 1 Read
                339  *
0E13: AD BE 0B  340  DRIVE1   LDA  DRIVE
0E16: 8D CC 0B  341          STA  VORDRIVE
0E19: A9 01     342          LDA  #1
0E1B: 8D BE 0B  343          STA  DRIVE
0E1E: 8D C8 0B  344          STA  COMMAND    ;READ
0E21: 20 9B 0E  345          JSR  BOT1
```

```
0E24: 60          346           RTS
                  347  *
                  348  * Drive 2 Write
                  349  *
0E25: AD BE 0B    350  DRIVE2    LDA  DRIVE
0E28: 8D CC 0B    351           STA  VORDRIVE
0E2B: A9 01       352           LDA  #1
0E2D: 8D BE 0B    353           STA  DRIVE
0E30: A9 02       354           LDA  #2
0E32: 8D C8 0B    355           STA  COMMAND    ;WRITE
0E35: 20 CB 0E    356           JSR  BOT2
0E38: 60          357           RTS
                  358  *
0E39: 0B 0F 10    359  MENU1     INV  "KOPIERPROGRAMM"
0E3C: 09 05 12 10 12 0F 07 12
0E44: 01 0D 0D
0E47: 8D          360           HEX  8D
0E48: 03 31 39    361           INV  "C1984 U.STIEHL"
0E4B: 38 34 20 15 2E 13 14 09
0E53: 05 08 0C
0E56: 8D 8D       362           HEX  8D8D
0E58: 0F 12 09    363           INV  "ORIGINAL IN D1"
0E5B: 07 09 0E 01 0C 20 09 0E
0E63: 20 04 31
0E66: 8D          364           HEX  8D
0E67: 04 15 10    365           INV  "DUPLIKAT IN D1"
0E6A: 0C 09 0B 01 14 20 09 0E
0E72: 20 04 31
0E75: 8D 8D       366           HEX  8D8D
0E77: 13 14 01    367           INV  "START  J/N"
0E7A: 12 14 20 20 0A 2F 0E
0E81: A0          368           ASC  " "
0E82: 7F          369           FLS  "?"
0E83: 00          370           HEX  00
0E84: CE C5 D5    371  MENU2     ASC  "NEUSTART MIT "
0E87: D3 D4 C1 D2 D4 A0 CD C9
0E8F: D4 A0
0E91: C3 C1 CC    372           ASC  "CALL 3000"
0E94: CC A0 B3 B0 B0 B0
0E9A: 00          373           HEX  00
                  374  *
0E9B: A2 00       375  BOT1     LDX  #0
0E9D: BD AB 0E    376  BOT1C    LDA  BOT1A,X
0EA0: F0 06       377           BEQ  BOT1B
0EA2: 20 F0 FD    378           JSR  PRINT
0EA5: E8          379           INX
0EA6: D0 F5       380           BNE  BOT1C
0EA8: 4C F8 0E    381  BOT1B    JMP  BOT3
0EAB: 0F 12 09    382  BOT1A    INV  "ORIGINAL"
0EAE: 07 09 0E 01 0C
0EB3: A0 C5 C9    383           ASC  " EINLEGEN"
0EB6: CE CC C5 C7 C5 CE
0EBC: 8D 8D       384           HEX  8D8D
0EBE: D7 A0 BD    385           ASC  "W = WEITER "
```

```
0EC1: A0 D7 C5 C9 D4 C5 D2 A0
0EC9: 60        386           FLS  " "
0ECA: 00        387           HEX  00
                388  *
0ECB: A2 00     389  BOT2      LDX  #0
0ECD: BD DB 0E  390  BOT2C     LDA  BOT2A,X
0ED0: F0 06     391           BEQ  BOT2B
0ED2: 20 F0 FD  392           JSR  PRINT
0ED5: E8        393           INX
0ED6: D0 F5     394           BNE  BOT2C
0ED8: 4C F8 0E  395  BOT2B     JMP  BOT3
0EDB: 0B 0F 10  396  BOT2A     INV  "KOPIE"
0EDE: 09 05
0EE0: A0 C5 C9  397           ASC  " EINLEGEN"
0EE3: CE CC C5 C7 C5 CE
0EE9: 8D 8D     398           HEX  8D8D
0EEB: D7 A0 BD  399           ASC  "W = WEITER "
0EEE: A0 D7 C5 C9 D4 C5 D2 A0
0EF6: 60        400           FLS  " "
0EF7: 00        401           HEX  00
                402  *
0EF8: 2C 10 C0  403  BOT3      BIT  $C010
0EFB: AD 00 C0  404  BOT3A     LDA  $C000
0EFE: 10 FB     405           BPL  BOT3A
0F00: C9 D7     406           CMP  #"W"
0F02: F0 06     407           BEQ  BOT3B
0F04: C9 F7     408           CMP  #"w"
0F06: F0 02     409           BEQ  BOT3B
0F08: D0 EE     410           BNE  BOT3
0F0A: 2C 10 C0  411  BOT3B     BIT  $C010
0F0D: 20 58 FC  412           JSR  HOME
0F10: AD 17 0F  413           LDA  COPY
0F13: 8D 00 04  414           STA  $400
0F16: 60        415           RTS
0F17: 43        416  COPY      FLS  "C"
```

6.2.5. Diskettenvergleichsprogramm

DISK.COMPARER

```
BSAVE DISK.COMPARER, A$0803, L670

                  1             ORG  $803
                  2    *
                  3    * Diskcomparer
                  4    * ------------
                  5    *
                  6    * Es wird gelesen von Track 0-34
                  7    * und von Sektor 15-0 (rückwärts)
                  8    *
0803: 4C 20 08    9             JMP  READANF1
0806: 60          10   SLOT6    HEX  60          ;6
0807: 01          11   DRIVE1   HEX  01
0808: 02          12   DRIVE2   HEX  02
0809: 00          13   SPUR     HEX  00          ;0-34
080A: 00          14   FLAG     HEX  00
                  15   RWTS     EQU  $3D9
                  16   DOSCOLD  EQU  $3D3
                  17   HOME     EQU  $FC58
                  18   HEXOUT   EQU  $FDDA
                  19   PRINT    EQU  $FDED
                  20   PRINT1   EQU  $FDF0
                  21   INVERSE  EQU  $FE80
                  22   NORMAL   EQU  $FE84
                  23   IND1     EQU  $CE         ;+$CF
                  24   BUFBOT1  EQU  $1000
                  25   BUFTOP1  EQU  $2000
                  26   IND2     EQU  $FE         ;+$FF
                  27   BUFBOT2  EQU  $2000
                  28   BUFTOP2  EQU  $3000
                  29   *
                  30   * Input-Output-Control-Block
                  31   *
080B: 01          32   IOB      HEX  01          ;STETS
080C: 60          33   SLOT     HEX  60          ;SLOT 6
080D: 00          34   DRIVE    HEX  00
080E: 00          35   VOLUME   HEX  00
080F: 00          36   TRACK    HEX  00
0810: 00          37   SECTOR   HEX  00
0811: 1C          38   DCTLOW   DFB  #<DCT
0812: 08          39   DCTHIGH  DFB  #>DCT
0813: 00          40   BUFLOW   HEX  00          ;IMMER
0814: 00          41   BUFHIGH  HEX  00
0815: 00          42             HEX  00          ;UNUSED
0816: 00          43             HEX  00          ;COUNT00
0817: 01          44   COMMAND  HEX  01          ;READ
```

```
0818: 00          45   DOSERROR HEX  00
0819: 00          46   EFFVOL   HEX  00
081A: 60          47   VORSLOT  HEX  60
081B: 00          48   VORDRIVE HEX  00
                  49   *
                  50   * Device-Characteristics-Table
                  51   *
081C: 00 01 EF    52   DCT      HEX  0001EFD8
081F: D8
                  53   *
                  54   * Total-Leser
                  55   *
0820: 20 58 FC    56   READANF1 JSR  HOME
0823: 20 80 FE    57            JSR  INVERSE
0826: A2 00       58            LDX  #0
0828: BD E4 09    59   MENU1    LDA  STRING1,X
082B: F0 07       60            BEQ  NORM1
082D: 20 F0 FD    61            JSR  PRINT1
0830: E8          62            INX
0831: 4C 28 08    63            JMP  MENU1
0834: 20 84 FE    64   NORM1    JSR  NORMAL
0837: A2 00       65            LDX  #0
0839: BD 02 0A    66   MENU2    LDA  STRING2,X
083C: F0 07       67            BEQ  WEITER1
083E: 20 F0 FD    68            JSR  PRINT1
0841: E8          69            INX
0842: 4C 39 08    70            JMP  MENU2
0845: 20 80 FE    71   WEITER1  JSR  INVERSE
0848: A9 60       72            LDA  #$60       ;CURSOR
084A: 20 F0 FD    73            JSR  PRINT1
084D: 20 84 FE    74            JSR  NORMAL
0850: A9 8D       75            LDA  #$8D
0852: 20 F0 FD    76            JSR  PRINT1
0855: 2C 10 C0    77            BIT  $C010
0858: AD 00 C0    78   KEY0     LDA  $C000
085B: 10 FB       79            BPL  KEY0
085D: 2C 10 C0    80            BIT  $C010
0860: C9 D7       81            CMP  #"W"
0862: D0 F4       82            BNE  KEY0
                  83   *
                  84   * Löschen
                  85   *
0864: A9 00       86            LDA  #0
0866: 85 CE       87            STA  IND1
0868: A9 0F       88            LDA  #>BUFBOT1-1
086A: 85 CF       89            STA  IND1+1
086C: A0 00       90            LDY  #0
086E: 98          91            TYA
086F: 91 CE       92   LOESCH1  STA  (IND1),Y
0871: E6 CE       93            INC  IND1
0873: D0 FA       94            BNE  LOESCH1
0875: E6 CF       95            INC  IND1+1
0877: A6 CF       96            LDX  IND1+1
0879: E0 30       97            CPX  #>BUFTOP2+1
```

```
087B: D0 F2     98           BNE  LOESCH1
                99  *
                100 * Überschrift
                101 *
087D: 20 58 FC  102          JSR  HOME
0880: A2 00     103          LDX  #0
0882: BD 50 0A  104 MENU3    LDA  STRING3,X
0885: F0 07     105          BEQ  READANF2
0887: 20 ED FD  106          JSR  PRINT
088A: E8        107          INX
088B: 4C 82 08  108          JMP  MENU3
088E: AD 06 08  109 READANF2 LDA  SLOT6
0891: 8D 0C 08  110          STA  SLOT
0894: 8D 1A 08  111          STA  VORSLOT
0897: A9 FF     112          LDA  #$FF       ;FF+1=0
0899: 8D 09 08  113          STA  SPUR
089C: AD 07 08  114          LDA  DRIVE1
089F: 8D 1B 08  115          STA  VORDRIVE
08A2: 8D 0D 08  116          STA  DRIVE
                117 *
                118 * Nächster Track
                119 *
08A5: 20 DF 09  120 NEXTSPUR JSR  PREGIST
08A8: EE 09 08  121          INC  SPUR
08AB: AD 09 08  122          LDA  SPUR
08AE: 8D 0F 08  123          STA  TRACK
08B1: C9 23     124          CMP  #35        ;0-34
08B3: D0 03     125          BNE  READDR1
08B5: 4C CA 09  126          JMP  FINIS1
                127 *
                128 * Drive 1
                129 *
08B8: AD 0D 08  130 READDR1  LDA  DRIVE
08BB: 8D 1B 08  131          STA  VORDRIVE
08BE: AD 07 08  132          LDA  DRIVE1
08C1: 8D 0D 08  133          STA  DRIVE
08C4: A9 10     134          LDA  #16        ;15+1
08C6: 8D 10 08  135          STA  SECTOR
08C9: A9 20     136          LDA  #>BUFTOP1
08CB: 8D 14 08  137          STA  BUFHIGH
08CE: CE 14 08  138 READDR1A DEC  BUFHIGH
08D1: CE 10 08  139          DEC  SECTOR
08D4: AD 10 08  140          LDA  SECTOR
08D7: C9 00     141          CMP  #0
08D9: 10 03     142          BPL  READDR1B
08DB: 4C EA 08  143          JMP  READDR2
08DE: A9 08     144 READDR1B LDA  #>IOB
08E0: A0 0B     145          LDY  #<IOB
08E2: 20 D9 03  146          JSR  RWTS
08E5: 90 E7     147          BCC  READDR1A   ;OKAY
08E7: 4C D0 09  148          JMP  ERROR1
                149 *
                150 * Drive 2
                151 *
```

```
08EA: AD 0D 08  152 READDR2  LDA  DRIVE
08ED: 8D 1B 08  153          STA  VORDRIVE
08F0: AD 08 08  154          LDA  DRIVE2
08F3: 8D 0D 08  155          STA  DRIVE
08F6: A9 10     156          LDA  #16        ;15+1
08F8: 8D 10 08  157          STA  SECTOR
08FB: A9 30     158          LDA  #>BUFTOP2
08FD: 8D 14 08  159          STA  BUFHIGH
0900: CE 14 08  160 READDR2A DEC  BUFHIGH
0903: CE 10 08  161          DEC  SECTOR
0906: AD 10 08  162          LDA  SECTOR
0909: C9 00     163          CMP  #0
090B: 10 03     164          BPL  READDR2B
090D: 4C 1C 09  165          JMP  COMPARE1
0910: A9 08     166 READDR2B LDA  #>IOB
0912: A0 0B     167          LDY  #<IOB
0914: 20 D9 03  168          JSR  RWTS
0917: 90 E7     169          BCC  READDR2A   ;OKAY
0919: 4C D0 09  170          JMP  ERROR1
                171 *
                172 * Vergleich beider Spuren
                173 *
091C: 20 DF 09  174 COMPARE1 JSR  PREGIST
091F: A9 8D     175          LDA  #$8D
0921: 20 ED FD  176          JSR  PRINT
0924: A9 D4     177          LDA  #"T"
0926: 20 ED FD  178          JSR  PRINT
0929: A9 A0     179          LDA  #$A0
092B: 20 ED FD  180          JSR  PRINT
092E: AD 09 08  181          LDA  SPUR
0931: 20 DA FD  182          JSR  HEXOUT
0934: A9 A0     183          LDA  #$A0
0936: 20 ED FD  184          JSR  PRINT
0939: A9 BA     185          LDA  #":"
093B: 20 ED FD  186          JSR  PRINT
093E: A9 00     187          LDA  #0
0940: 85 CE     188          STA  IND1
0942: 85 FE     189          STA  IND2
0944: A9 10     190          LDA  #>BUFBOT1
0946: 85 CF     191          STA  IND1+1
0948: A9 20     192          LDA  #>BUFBOT2
094A: 85 FF     193          STA  IND2+1
094C: A9 00     194          LDA  #0
094E: 8D 0A 08  195          STA  FLAG       ;0=OKAY
0951: A0 00     196 COMPARE2 LDY  #0
0953: B1 CE     197          LDA  (IND1),Y
0955: D1 FE     198          CMP  (IND2),Y
0957: F0 1B     199          BEQ  COMPARE3
                200 *
                201 * Flag 1 = nicht identisch
                202 *
0959: A9 01     203          LDA  #1
095B: 8D 0A 08  204          STA  FLAG
095E: A9 A0     205          LDA  #$A0
```

```
0960: 20 ED FD  206           JSR  PRINT
0963: 38        207           SEC
0964: A5 CF     208           LDA  IND1+1
0966: E9 10     209           SBC  #>BUFBOT1
0968: 20 DA FD  210           JSR  HEXOUT
096B: A9 00     211           LDA  #0
096D: 85 FE     212           STA  IND2
096F: 85 CE     213           STA  IND1
0971: 4C 7A 09  214           JMP  COMPARE4
0974: E6 FE     215  COMPARE3 INC  IND2
0976: E6 CE     216           INC  IND1
0978: D0 D7     217           BNE  COMPARE2
097A: E6 FF     218  COMPARE4 INC  IND2+1
097C: E6 CF     219           INC  IND1+1
097E: A5 CF     220           LDA  IND1+1
0980: C9 20     221           CMP  #>BUFTOP1
0982: 90 CD     222           BCC  COMPARE2
0984: AD 0A 08  223           LDA  FLAG
0987: D0 12     224           BNE  FLAG1
0989: A9 A0     225           LDA  #$A0       ;OK
098B: 20 ED FD  226           JSR  PRINT
098E: A9 CF     227           LDA  #"O"
0990: 20 ED FD  228           JSR  PRINT
0993: A9 CB     229           LDA  #"K"
0995: 20 ED FD  230           JSR  PRINT
0998: 4C A5 08  231  COMPARE5 JMP  NEXTSPUR
                232  *
                233  * nicht identisch
                234  *
099B: A2 00     235  FLAG1    LDX  #0
099D: BD 6D 0A  236  MENU4    LDA  STRING4,X
09A0: F0 07     237           BEQ  KEY1
09A2: 20 F0 FD  238           JSR  PRINT1
09A5: E8        239           INX
09A6: 4C 9D 09  240           JMP  MENU4
09A9: 2C 10 C0  241  KEY1     BIT  $C010
09AC: AD 00 C0  242  KEY2     LDA  $C000
09AF: 10 FB     243           BPL  KEY2
09B1: 2C 10 C0  244           BIT  $C010
09B4: C9 CA     245           CMP  #"J"
09B6: F0 E0     246           BEQ  COMPARE5
09B8: C9 CE     247           CMP  #"N"
09BA: D0 F0     248           BNE  KEY2
09BC: A2 00     249           LDX  #0
09BE: BD 7A 0A  250  MENU5    LDA  STRING5,X
09C1: F0 07     251           BEQ  FINIS1
09C3: 20 F0 FD  252           JSR  PRINT1
09C6: E8        253           INX
09C7: 4C BE 09  254           JMP  MENU5
09CA: 20 DF 09  255  FINIS1   JSR  PREGIST
09CD: 4C D3 03  256           JMP  DOSCOLD
```

```
                    257  *
                    258  * DOS-Fehler
                    259  *
09D0: 20 58 FC      260  ERROR1   JSR  HOME
09D3: 20 80 FE      261          JSR  INVERSE
09D6: AD 18 08      262          LDA  DOSERROR
09D9: 20 DA FD      263          JSR  HEXOUT
09DC: 20 84 FE      264          JSR  NORMAL
                    265  *
                    266  * P-Status-Register
                    267  *
09DF: A9 00         268  PREGIST  LDA  #0
09E1: 85 48         269          STA  $48
09E3: 60            270          RTS
                    271  *
                    272  * Menüs
                    273  *
09E4: C4 C9 D3      274  STRING1  ASC  "DISC-COMPARER"
09E7: C3 AD C3 CF CD D0 C1 D2
09EF: C5 D2
09F1: 8D            275          HEX  8D
09F2: D5 AE D3      276          ASC  "U.STIEHL/1984"
09F5: D4 C9 C5 C8 CC AF B1 B9
09FD: B8 B4
09FF: 8D 8D 00      277          HEX  8D8D00
0A02: C1 D5 C6      278  STRING2  ASC  "AUF IDENTITAET"
0A05: A0 C9 C4 C5 CE D4 C9 D4
0A0D: C1 C5 D4
0A10: A0 DA D5      279          ASC  " ZU PRUEFENDE"
0A13: A0 D0 D2 D5 C5 C6 C5 CE
0A1B: C4 C5
0A1D: A0 C4 C9      280          ASC  " DISKETTEN"
0A20: D3 CB C5 D4 D4 C5 CE
0A27: 8D            281          HEX  8D
0A28: C9 CE A0      282          ASC  "IN DRIVE 1 + 2"
0A2B: C4 D2 C9 D6 C5 A0 B1 A0
0A33: AB A0 B2
0A36: A0 C5 C9      283          ASC  " EINLEGEN!!!"
0A39: CE CC C5 C7 C5 CE A1 A1
0A41: A1
0A42: 8D 8D         284          HEX  8D8D
0A44: D7 A0 BD      285          ASC  "W = WEITER "
0A47: A0 D7 C5 C9 D4 C5 D2 A0
0A4F: 00            286          HEX  00
0A50: 8D            287  STRING3  HEX  8D
0A51: D4 D2 C1      288          ASC  "TRACK: "
0A54: C3 CB BA A0
0A58: C1 C2 D7      289          ASC  "ABWEICHENDE"
0A5B: C5 C9 C3 C8 C5 CE C4 C5
0A63: A0 D3 C5      290          ASC  " SEKTOREN"
0A66: CB D4 CF D2 C5 CE
0A6C: 00            291          HEX  00
0A6D: 8D            292  STRING4  HEX  8D
0A6E: D7 C5 C9      293          ASC  "WEITER J/N "
```

```
0A71: D4 C5 D2 A0 CA AF CE A0
0A79: 00        294           HEX  00
0A7A: 8D        295  STRING5  HEX  8D
0A7B: D4 D2 C1  296           ASC  "TRACKS "
0A7E: C3 CB D3 A0
0A82: C9 CD A0  297           ASC  "IM MONITOR "
0A85: CD CF CE C9 D4 CF D2 A0
0A8D: C1 C2 A0  298           ASC  "AB $1000 "
0A90: A4 B1 B0 B0 B0 A0
0A96: D5 CE C4  299           ASC  "UND $2000"
0A99: A0 A4 B2 B0 B0 B0
0A9F: 8D 00     300           HEX  8D00
```

6.2.6. Bad-Sector-Routine

BAD.SECTOR.ROUTINE

```
BSAVE BAD.SECTOR.ROUTINE, A$0803, L370

                1             ORG  $803
                2    *
                3    * Bad-Sector-Routine
                4    * ------------------
                5    *
0803: 4C 1D 08  6             JMP  START1
0806: 00        7    SPUR     HEX  00          ;$00-$0F
0807: 00        8    SEKTOR   HEX  00          ;$00-$22
                9    RWTS     EQU  $03D9
                10   HOME     EQU  $FC58
                11   HEXOUT   EQU  $FDDA
                12   PRINT    EQU  $FDED
                13   INVERSE  EQU  $FE80
                14   NORMAL   EQU  $FE84
                15   *
                16   * IOB
                17   *
0808: 01        18   IOB      HEX  01          ;STETS
0809: 60        19   SLOT     HEX  60          ;SLOT 6
080A: 01        20   DRIVE    HEX  01
080B: 00        21   VOLUME   HEX  00
080C: 00        22   TRACK    HEX  00
080D: 00        23   SECTOR   HEX  00
080E: 19        24   DCTLOW   DFB  #<DCT
080F: 08        25   DCTHIGH  DFB  #>DCT
```

```
0810: 00          26   BUFLOW   HEX  00
0811: 00          27   BUFHIGH  HEX  00          ;AB 1000
0812: 00          28            HEX  00          ;UNUSED
0813: 00          29            HEX  00          ;COUNT:0
0814: 01          30   COMMAND  HEX  01          ;READ
0815: 00          31   DOSERROR HEX  00
0816: 00          32   VORVOL   HEX  00
0817: 60          33   VORSLOT  HEX  60
0818: 01          34   VORDRIVE HEX  01
                  35   *
                  36   * DCT
                  37   *
0819: 00 01 EF    38   DCT      HEX  0001EFD8
081C: D8
                  39   *
                  40   * Total-Leser
                  41   *
081D: 20 58 FC    42   START1   JSR  HOME
0820: A2 00       43            LDX  #0
0822: BD 47 09    44   MESS1    LDA  MESSAGE1,X
0825: F0 06       45            BEQ  KEY
0827: 20 ED FD    46            JSR  PRINT
082A: E8          47            INX
082B: D0 F5       48            BNE  MESS1
082D: 2C 10 C0    49   KEY      BIT  $C010
0830: AD 00 C0    50   KEY1     LDA  $C000
0833: C9 D7       51            CMP  #"W"
0835: D0 F9       52            BNE  KEY1
0837: 2C 10 C0    53            BIT  $C010
083A: 20 58 FC    54            JSR  HOME
083D: A2 00       55            LDX  #0
083F: BD 69 09    56   MESS2    LDA  MESSAGE2,X
0842: F0 06       57            BEQ  START2
0844: 20 ED FD    58            JSR  PRINT
0847: E8          59            INX
0848: D0 F5       60            BNE  MESS2
                  61   *
                  62   * Vorherige Slot + Drive + Volume
                  63   *
084A: 20 E3 03    64   START2   JSR  $3E3         ;IOB-WO?
084D: 84 CE       65            STY  $CE
084F: 85 CF       66            STA  $CF
0851: A0 0E       67            LDY  #$0E
0853: B1 CE       68            LDA  ($CE),Y
0855: 8D 16 08    69            STA  VORVOL
0858: A9 00       70            LDA  #0
085A: 8D 0B 08    71            STA  VOLUME
085D: C8          72            INY
085E: B1 CE       73            LDA  ($CE),Y
0860: 8D 17 08    74            STA  VORSLOT
0863: 8D 09 08    75            STA  SLOT
0866: C8          76            INY
0867: B1 CE       77            LDA  ($CE),Y
0869: 8D 18 08    78            STA  VORDRIVE
```

```
086C: 8D 0A 08  79              STA  DRIVE
                80  *
086F: A9 00     81              LDA  #0          ;AB 0
0871: 8D 06 08  82              STA  SPUR
0874: A9 00     83              LDA  #0          ;AB 0
0876: 8D 07 08  84              STA  SEKTOR
0879: 20 7F 08  85              JSR  SPURWORT
087C: 4C E1 08  86              JMP  LESER2
                87  *
087F: 20 80 FE  88  SPURWORT    JSR  INVERSE
0882: A9 8D     89              LDA  #$8D
0884: 20 ED FD  90              JSR  PRINT
0887: A9 D3     91              LDA  #"S"
0889: 20 ED FD  92              JSR  PRINT
088C: A9 D0     93              LDA  #"P"
088E: 20 ED FD  94              JSR  PRINT
0891: A9 D5     95              LDA  #"U"
0893: 20 ED FD  96              JSR  PRINT
0896: A9 D2     97              LDA  #"R"
0898: 20 ED FD  98              JSR  PRINT
089B: A9 BA     99              LDA  #":"
089D: 20 ED FD  100             JSR  PRINT
08A0: AD 06 08  101             LDA  SPUR
08A3: 20 DA FD  102             JSR  HEXOUT
08A6: 20 84 FE  103             JSR  NORMAL
08A9: A9 A0     104             LDA  #$A0
08AB: 20 ED FD  105             JSR  PRINT
08AE: 60        106             RTS
08AF: EE 07 08  107 LESER1      INC  SEKTOR      ;00->0F
08B2: AD 07 08  108             LDA  SEKTOR
08B5: C9 10     109             CMP  #16
08B7: 90 28     110             BCC  LESER2      ;<=15
08B9: A9 00     111             LDA  #0
08BB: 8D 07 08  112             STA  SEKTOR
08BE: EE 06 08  113             INC  SPUR        ;00->22
08C1: A9 8D     114             LDA  #$8D
08C3: 20 ED FD  115             JSR  PRINT
08C6: AD 00 C0  116             LDA  $C000
08C9: C9 9B     117             CMP  #$9B        ;ESC
08CB: F0 07     118             BEQ  EXIT
08CD: AD 06 08  119             LDA  SPUR
08D0: C9 23     120             CMP  #35         ;<35
08D2: 90 0A     121             BCC  LESER1A
08D4: A9 00     122 EXIT        LDA  #0
08D6: 85 48     123             STA  $48         ;P-REG
08D8: 2C 10 C0  124             BIT  $C010
08DB: 4C D0 03  125             JMP  $3D0        ;DOSWRM
08DE: 20 7F 08  126 LESER1A     JSR  SPURWORT
08E1: AD 07 08  127 LESER2      LDA  SEKTOR
08E4: 20 DA FD  128             JSR  HEXOUT
08E7: A9 A0     129             LDA  #$A0
08E9: 20 ED FD  130             JSR  PRINT
08EC: AD 06 08  131 LESER3      LDA  SPUR
08EF: 8D 0C 08  132             STA  TRACK
```

```
08F2: AD 07 08  133           LDA  SEKTOR
08F5: 8D 0D 08  134           STA  SECTOR
08F8: 18        135           CLC
08F9: 69 10     136           ADC  #$10        ;+$1000
08FB: 8D 11 08  137           STA  BUFHIGH
08FE: A9 00     138           LDA  #0
0900: 8D 74 09  139           STA  FLAG
0903: A9 08     140           LDA  #>IOB
0905: A0 08     141           LDY  #<IOB
0907: 20 D9 03  142           JSR  RWTS
090A: 90 06     143           BCC  LESER5      ;OKAY
090C: 20 80 FE  144           JSR  INVERSE
090F: EE 74 09  145           INC  FLAG
0912: 20 42 09  146  LESER5   JSR  LESER6
0915: A9 A8     147           LDA  #"("
0917: 20 ED FD  148           JSR  PRINT
091A: AD 74 09  149           LDA  FLAG
091D: F0 09     150           BEQ  OKAY
091F: AD 15 08  151           LDA  DOSERROR
0922: 20 DA FD  152           JSR  HEXOUT
0925: 4C 32 09  153           JMP  LESER5A
0928: A9 CF     154  OKAY     LDA  #"O"
092A: 20 ED FD  155           JSR  PRINT
092D: A9 CB     156           LDA  #"K"
092F: 20 ED FD  157           JSR  PRINT
0932: A9 A9     158  LESER5A  LDA  #")"
0934: 20 ED FD  159           JSR  PRINT
0937: 20 84 FE  160           JSR  NORMAL
093A: A9 A0     161           LDA  #$A0
093C: 20 ED FD  162           JSR  PRINT
093F: 4C AF 08  163           JMP  LESER1
0942: A9 00     164  LESER6   LDA  #0
0944: 85 48     165           STA  $48         ;P-REG
0946: 60        166           RTS
0947: 02 01 04  167  MESSAGE1 INV  "BAD-SECTOR"
094A: 2D 13 05 03 14 0F 12
0951: 8D        168           HEX  8D
0952: 15 2E 13  169           INV  "U.STIEHL84"
0955: 14 09 05 08 0C 38 34
095C: 8D 8D     170           HEX  8D8D
095E: 17 3D 17  171           INV  "W=WEITER"
0961: 05 09 14 05 12
0966: A0        172           ASC  " "
0967: 60        173           FLS  " "
0968: 00        174           HEX  00
0969: 05 13 03  175  MESSAGE2 INV  "ESC=EXIT"
096C: 3D 05 18 09 14
0971: 8D 8D     176           HEX  8D8D
0973: 00        177           HEX  00
0974: 00        178  FLAG     HEX  00
```

6.2.7. Kopie der DOS-Spuren

DOS.KOPIE

```
100  PRINT  CHR$ (4)"BLOAD DOS.KOPIE.OBJ,A$0300"
110  TEXT : HOME : INVERSE : PRINT "DOS-KOPIE":
     NORMAL : PRINT
120  INPUT "ORIGINAL EINLEGEN ";X$
130  CALL 768
140  X$ = "": FOR X = 1 TO 30:X$ = X$ +
     CHR$ ( PEEK (14709 - 1 + X) - 128): NEXT
150  VTAB 8: HTAB 1: PRINT "NEUER HELLO-NAME":
     VTAB 10: HTAB 1: PRINT X$: VTAB 10: HTAB 1:
     INPUT "";Y$
160  IF Y$ = "" THEN  PRINT X$: GOTO 200
170  VTAB 10: HTAB 1: CALL  - 868: PRINT : PRINT Y$
180  IF  LEN (Y$) < 30 THEN Y-8d- = Y$ + " ": GOTO 180
190  FOR X = 1 TO 30: POKE 14709 - 1 + X,
     ASC ( MID$ (Y$,X,1)) + 128: NEXT
200  PRINT : PRINT : INPUT "DUPLIKAT EINLEGEN ";X$
210  CALL 771
```

DOS.KOPIE.OBJ

```
BSAVE DOS.KOPIE.OBJ, A$0300, L204

                1            ORG  $300
                2    *
                3    * DOS.KOPIE.OBJ
                4    *
                5    IND      EQU  $CE
                6    DOSWARM  EQU  $03D0
                7    RWTS     EQU  $03D9
                8    GETIOB   EQU  $03E3
                9    HEXOUT   EQU  $FDDA
               10    *
               11    *-------------------------------
               12    *
0300: 4C 39 03 13            JMP  READORIG   ;768
0303: 4C 4F 03 14            JMP  WRITEDUP   ;771
               15    *
               16    *-------------------------------
               17    *
               18    * IOB
               19    *
0306: 01       20    IOB      HEX  01         ;STETS
0307: 60       21    SLOT     HEX  60         ;SLOT 6
```

```
0308: 01          22  DRIVE    HEX  01
0309: 00          23  VOLUME   HEX  00
030A: 00          24  TRACK    HEX  00
030B: 00          25  SECTOR   HEX  00
030C: 17          26  DCTLOW   DFB  #<DCT
030D: 03          27  DCTHIGH  DFB  #>DCT
030E: 00          28  BUFLOW   HEX  00
030F: 00          29  BUFHIGH  HEX  00          ;STETS 0
0310: 00          30           HEX  00          ;UNUSED
0311: 00          31           HEX  00          ;COUNT:0
0312: 01          32  COMMAND  HEX  01          ;READ
0313: 00          33  DOSERROR HEX  00
0314: 00          34  EFFVOL   HEX  00
0315: 60          35  VORSLOT  HEX  60
0316: 01          36  VORDRIVE HEX  01
                  37  *
                  38  * DCT
                  39  *
0317: 00 01 EF    40  DCT      HEX  0001EFD8   ;STETS
031A: D8
                  41  *
                  42  * Slot/Drive/Volume
                  43  *
031B: 20 E3 03    44  PARAMS   JSR  GETIOB     ;IOB WO?
031E: 84 CE       45           STY  IND
0320: 85 CF       46           STA  IND+1
0322: A0 01       47           LDY  #1
0324: B1 CE       48           LDA  (IND),Y    ;IOB
0326: 8D 07 03    49           STA  SLOT
0329: 8D 15 03    50           STA  VORSLOT
032C: C8          51           INY
032D: B1 CE       52           LDA  (IND),Y
032F: 8D 16 03    53           STA  VORDRIVE
0332: C8          54           INY
0333: B1 CE       55           LDA  (IND),Y
0335: 8D 14 03    56           STA  EFFVOL
0338: 60          57           RTS
                  58  *
0339: 20 1B 03    59  READORIG JSR  PARAMS
033C: A9 01       60           LDA  #$01       ;Drive 1
033E: 8D 08 03    61           STA  DRIVE
0341: A9 00       62           LDA  #$00
0343: 8D 09 03    63           STA  VOLUME
0346: A9 01       64           LDA  #$01
0348: 8D 12 03    65           STA  COMMAND
034B: 20 68 03    66           JSR  RDWR
034E: 60          67           RTS
                  68  *
034F: 20 1B 03    69  WRITEDUP JSR  PARAMS
0352: A9 01       70           LDA  #$01       ;Drive 1
0354: 8D 08 03    71           STA  DRIVE
0357: A9 00       72           LDA  #$00
0359: 8D 09 03    73           STA  VOLUME
```

```
035C: 8D 14 03  74            STA  EFFVOL
035F: A9 02     75            LDA  #$02
0361: 8D 12 03  76            STA  COMMAND
0364: 20 68 03  77            JSR  RDWR
0367: 60        78            RTS
                79  *
0368: A9 20     80  RDWR      LDA  #$20       ;2000
036A: 8D 0F 03  81            STA  BUFHIGH
036D: A9 00     82            LDA  #$00       ;T 00
036F: 8D 0A 03  83            STA  TRACK
0372: 20 90 03  84            JSR  TRACKER1
0375: A9 30     85            LDA  #$30       ;3000
0377: 8D 0F 03  86            STA  BUFHIGH
037A: A9 01     87            LDA  #$01       ;T 01
037C: 8D 0A 03  88            STA  TRACK
037F: 20 90 03  89            JSR  TRACKER1
0382: A9 40     90            LDA  #$40       ;4000
0384: 8D 0F 03  91            STA  BUFHIGH
0387: A9 02     92            LDA  #$02       ;T 02
0389: 8D 0A 03  93            STA  TRACK
038C: 20 90 03  94            JSR  TRACKER1
038F: 60        95            RTS
                96  *
                97  * Erst BUFHIGH + Track bestimmen
                98  *
0390: A9 00     99  TRACKER1  LDA  #$00
0392: 8D 0E 03  100           STA  BUFLOW
0395: 18        101           CLC
0396: AD 0F 03  102           LDA  BUFHIGH
0399: 69 0F     103           ADC  #$0F
039B: 8D 0F 03  104           STA  BUFHIGH
039E: A9 0F     105           LDA  #$0F
03A0: 8D 0B 03  106           STA  SECTOR
03A3: 20 B3 03  107 TRACKER2  JSR  RWTSLOOP
03A6: CE 0F 03  108           DEC  BUFHIGH
03A9: CE 0B 03  109           DEC  SECTOR
03AC: 10 F5     110           BPL  TRACKER2
03AE: A9 00     111 READEND   LDA  #0
03B0: 85 48     112           STA  $48        ;P-REG
03B2: 60        113           RTS
03B3: A9 03     114 RWTSLOOP  LDA  #>IOB
03B5: A0 06     115           LDY  #<IOB
03B7: 20 D9 03  116           JSR  RWTS
03BA: B0 01     117           BCS  READERR
03BC: 60        118           RTS
03BD: 68        119 READERR   PLA
03BE: 68        120           PLA
03BF: AD 13 03  121           LDA  DOSERROR
03C2: 20 DA FD  122           JSR  HEXOUT
03C5: A9 00     123           LDA  #0
03C7: 85 48     124           STA  $48
03C9: 4C D0 03  125           JMP  DOSWARM
```

6.2.8. Datendiskette ohne DOS

DOSLOS

```
10  PRINT  CHR$ (4)"BLOAD DOSLOS.OBJ":
    PRINT  CHR$ (7):X =  PEEK (49168)
20  HOME : VTAB 10: INVERSE : PRINT "LEERDISK EINLEGEN":
    PRINT "DANN RETURN INIT?";: NORMAL : PRINT " ";
30  INPUT "";X$: PRINT  CHR$ (4)"INIT X":
    PRINT  CHR-8d- (4)"DELETE X":
    CALL 4102:X =  PEEK (49168): GOTO 20
```

DOSLOS.OBJ

```
BSAVE DOSLOS.OBJ, A$1006, L762
```

```
                1             ORG  $1006      ;4102
                2   *
1006: 4C 1E 10  3             JMP  INITIAL
                4   *
                5   * DOSLOS.OBJ
                6   * ==========
                7   *
                8   * Dieses Unterprogramm verwandelt
                9   * eine mit DOSLOS initialisierte
                10  * Diskette in eine Datendiskette
                11  * ohne DOS mit 2 Spuren mehr
                12  * Speicherraum und einer ent-
                13  * sprechenden Warnmeldung auf
                14  * Spur 0, Sektor 0
                15  *
                16  CONTROL  EQU  $C088
                17  SETNORM  EQU  $FE84
                18  INIT     EQU  $FB2F
                19  SETKBD   EQU  $FE89
                20  SETVID   EQU  $FE93
                21  HOME     EQU  $FC58
                22  COUT1    EQU  $FDF0
                23  RDKEY    EQU  $FD0C
                24  HEXOUT   EQU  $FDDA
                25  BOOTSLOT EQU  $3E        :3E-3F
                26  IND1     EQU  $CE        ;CE-CF
                27  RWTSLOCO EQU  $3E3
                28  RWTS     EQU  $3D9
                29  *
                30  * IOB
                31  *
```

```
1009: 01        32   IOB      HEX  01         ;STETS
100A: 60        33   SLOT     HEX  60         ;SLOT 6
100B: 01        34   DRIVE    HEX  01
100C: 00        35   VOLUME   HEX  00
100D: 00        36   TRACK    HEX  00
100E: 00        37   SECTOR   HEX  00
100F: 1A        38   DCTLOW   DFB  #<DCT
1010: 10        39   DCTHIGH  DFB  #>DCT
1011: 00        40   BUFLOW   HEX  00
1012: 00        41   BUFHIGH  HEX  00
1013: 00        42            HEX  00         ;UNUSED
1014: 00        43            HEX  00         ;COUNT:0
1015: 02        44   COMMAND  HEX  02         ;WRITE
1016: 00        45   DOSERROR HEX  00
1017: 00        46   EFFVOL   HEX  00
1018: 60        47   VORSLOT  HEX  60
1019: 01        48   VORDRIVE HEX  01
                49   *
                50   * DCT
                51   *
101A: 00 01 EF  52   DCT      HEX  0001EFD8   ;STETS
101D: D8
                53   *
                54   * Boot- und VTOC-Sektor kopieren
                55   *
101E: 20 E3 03  56   INITIAL  JSR  RWTSLOCO
1021: 84 CE     57            STY  IND1       ;LOWIOB
1023: 85 CF     58            STA  IND1+1     ;HIGHIOB
1025: A0 01     59            LDY  #1
1027: B1 CE     60            LDA  (IND1),Y
1029: 8D 0A 10  61            STA  SLOT
102C: 8D 18 10  62            STA  VORSLOT
102F: C8        63            INY
1030: B1 CE     64            LDA  (IND1),Y
1032: 8D 0B 10  65            STA  DRIVE
1035: 8D 19 10  66            STA  VORDRIVE
                67   *
                68   * Write TRK 00, SEC 00 von $1100
                69   *
1038: A9 00     70            LDA  #$00
103A: 8D 0D 10  71            STA  TRACK
103D: 8D 0E 10  72            STA  SECTOR
1040: 8D 11 10  73            STA  BUFLOW
1043: A9 11     74            LDA  #$11
1045: 8D 12 10  75            STA  BUFHIGH
1048: A9 10     76            LDA  #>IOB
104A: A0 09     77            LDY  #<IOB
104C: 20 D9 03  78            JSR  RWTS
104F: B0 20     79            BCS  READERR
                80   *
                81   * Write TRK 11, SEC 00 von $1200
                82   *
1051: A9 11     83            LDA  #$11
1053: 8D 0D 10  84            STA  TRACK
```

```
1056: A9 00     85          LDA  #$00
1058: 8D 0E 10  86          STA  SECTOR
105B: 8D 11 10  87          STA  BUFLOW
105E: A9 12     88          LDA  #$12
1060: 8D 12 10  89          STA  BUFHIGH
1063: A9 10     90          LDA  #>IOB
1065: A0 09     91          LDY  #<IOB
1067: 20 D9 03  92          JSR  RWTS
106A: B0 05     93          BCS  READERR
                94  *
106C: A9 00     95  EXIT    LDA  #0
106E: 85 48     96          STA  $48        ;P-REG.
1070: 60        97          RTS
                98  *
1071: 20 58 FC  99  READERR JSR  HOME
1074: AD 16 10  100         LDA  DOSERROR
1077: 20 DA FD  101         JSR  HEXOUT
107A: 4C 6C 10  102         JMP  EXIT
                103 *
                104 * Page-Füller bis $08FF
                105 *
                106         DS   130
                107 *
10FF: 00        108         HEX  00
                109 *
                110 ********************************
                111 *
                112         ORG  $800
                113 *
                114 * Muß sich in $1100-11FF befinden
                115 *
                116 * Track 00, Sector 00 = Boot
                117 *
0800: 01        118 TRKOSECO HEX  01
0801: 9D 88 C0  119         STA  CONTROL,X  ;Motor aus
                120 *
                121 * 3E-3F: $C600 Boot-up
                122 *
0804: A5 2B     123         LDA  $2B
0806: 4A        124         LSR
0807: 4A        125         LSR
0808: 4A        126         LSR
0809: 4A        127         LSR
080A: 09 C0     128         ORA  #$C0
080C: 85 3F     129         STA  $3F
080E: A9 00     130         LDA  #$00
0810: 85 3E     131         STA  $3E
                132 *
0812: D8        133         CLD
0813: 20 84 FE  134         JSR  SETNORM
0816: 20 2F FB  135         JSR  INIT
0819: 20 89 FE  136         JSR  SETKBD
081C: 20 93 FE  137         JSR  SETVID
081F: 20 58 FC  138         JSR  HOME
```

```
                    139  *
                    140  * Boot-Meldung anzeigen
                    141  *
0822: A0 00         142          LDY  #$00
0824: B9 35 08      143  MESSLOOP LDA  MESSAGE,Y
0827: F0 06         144          BEQ  MESSEND
0829: 20 F0 FD      145          JSR  COUT1
082C: C8            146          INY
082D: D0 F5         147          BNE  MESSLOOP
082F: 20 0C FD      148  MESSEND  JSR  RDKEY
0832: 6C 3E 00      149          JMP  (BOOTSLOT)
                    150  *
0835: 8D 8D         151  MESSAGE  HEX  8D8D
0837: CE C9 C3      152          ASC  "NICHT "
083A: C8 D4 A0
083D: C2 CF CF      153          ASC  "BOOTFAEHIGE "
0840: D4 C6 C1 C5 C8 C9 C7 C5
0848: A0
0849: C4 C1 D4      154          ASC  "DATENDISKETTE"
084C: C5 CE C4 C9 D3 CB C5 D4
0854: D4 C5
0856: 8D 8D         155          HEX  8D8D
0858: D0 D2 CF      156          ASC  "PROGRAMM"
085B: C7 D2 C1 CD CD
0860: C4 C9 D3      157          ASC  "DISKETTE "
0863: CB C5 D4 D4 C5 A0
0869: C5 C9 CE      158          ASC  "EINLEGEN "
086C: CC C5 C7 C5 CE A0
0872: 8D 8D         159          HEX  8D8D
0874: D5 CE C4      160          ASC  "UND "
0877: A0
0878: D2 C5 D4      161          ASC  "RETURN "
087B: D5 D2 CE A0
087F: C4 D2 D5      162          ASC  "DRUECKEN "
0882: C5 C3 CB C5 CE A0
0888: 00 00 00      163          HEX  0000000000
088B: 00 00
088D: 00 00 00      164          HEX  0000000000
0890: 00 00
0892: 00 00 00      165          HEX  0000000000
0895: 00 00
0897: 00 00 00      166          HEX  0000000000
089A: 00 00
089C: 00 00 00      167          HEX  0000000000
089F: 00 00
08A1: 00 00 00      168          HEX  0000000000
08A4: 00 00
08A6: 00 00 00      169          HEX  0000000000
08A9: 00 00
08AB: 00 00 00      170          HEX  0000000000
08AE: 00 00
08B0: 00 00 00      171          HEX  0000000000
08B3: 00 00
08B5: 00 00 00      172          HEX  0000000000
```

```
08B8: 00 00
08BA: 00 00 00  173           HEX  0000000000
08BD: 00 00
08BF: 00 00 00  174           HEX  0000000000
08C2: 00 00
08C4: 00 00 00  175           HEX  0000000000
08C7: 00 00
08C9: 00 00 00  176           HEX  0000000000
08CC: 00 00
08CE: 00 00 00  177           HEX  0000000000
08D1: 00 00
08D3: 00 00 00  178           HEX  0000000000
08D6: 00 00
08D8: 00 00 00  179           HEX  0000000000
08DB: 00 00
08DD: 00 00 00  180           HEX  0000000000
08E0: 00 00
08E2: 00 00 00  181           HEX  0000000000
08E5: 00 00
08E7: 00 00 00  182           HEX  0000000000
08EA: 00 00
08EC: 00 00 00  183           HEX  0000000000
08EF: 00 00
08F1: 00 00 00  184           HEX  0000000000
08F4: 00 00
08F6: 00 00 00  185           HEX  0000000000
08F9: 00 00
08FB: 00 00 00  186           HEX  0000000000
08FE: 00 00
                187  *
                188  *********************************
                189  *
                190  * Muß sich in $1200-12FF befinden
                191  *
                192  * Track 11, Sector 00 = VTOC
                193  *
0900: 04        194  TR11SECF HEX  04          ;NOT US
0901: 11 0F     195           HEX  110F        ;T-S-CAT
0903: 03        196           HEX  03          ;3.3
0904: 00 00     197           HEX  0000
0906: FE        198           HEX  FE          ;254
0907: 00 00 00  199           HEX  0000000000
090A: 00 00
090C: 00 00 00  200           HEX  0000000000
090F: 00 00
0911: 00 00 00  201           HEX  0000000000
0914: 00 00
0916: 00 00 00  202           HEX  0000000000
0919: 00 00
091B: 00 00 00  203           HEX  0000000000
091E: 00 00
0920: 00 00 00  204           HEX  0000000000
0923: 00 00
0925: 00 00     205           HEX  0000
```

```
0927: 7A          206          HEX  7A          ;122MAX.
0928: 00 00 00    207          HEX  0000000000
092B: 00 00
092D: 00 00 00    208          HEX  000000
0930: 11          209          HEX  11          ;LAST.T.
0931: 01          210          HEX  01          ;DIRECT.
0932: 00 00       211          HEX  0000
0934: 23          212          HEX  23          ;35 TRK.
0935: 10          213          HEX  10          ;16 SEC.
0936: 00 01       214          HEX  0001        ;$0100
0938: 00 00 00    215          HEX  00000000    ;T.00
093B: 00
093C: FF FF 00    216          HEX  FFFF0000    ;T.01
093F: 00
0940: FF FF 00    217          HEX  FFFF0000    ;T.02
0943: 00
0944: FF FF 00    218          HEX  FFFF0000    ;T.03
0947: 00
0948: FF FF 00    219          HEX  FFFF0000    ;T.04
094B: 00
094C: FF FF 00    220          HEX  FFFF0000    ;T.05
094F: 00
0950: FF FF 00    221          HEX  FFFF0000    ;T.06
0953: 00
0954: FF FF 00    222          HEX  FFFF0000    ;T.07
0957: 00
0958: FF FF 00    223          HEX  FFFF0000    ;T.08
095B: 00
095C: FF FF 00    224          HEX  FFFF0000    ;T.09
095F: 00
0960: FF FF 00    225          HEX  FFFF0000    ;T.0A
0963: 00
0964: FF FF 00    226          HEX  FFFF0000    ;T.0B
0967: 00
0968: FF FF 00    227          HEX  FFFF0000    ;T.0C
096B: 00
096C: FF FF 00    228          HEX  FFFF0000    ;T.0D
096F: 00
0970: FF FF 00    229          HEX  FFFF0000    ;T.0E
0973: 00
0974: FF FF 00    230          HEX  FFFF0000    ;T.0F
0977: 00
0978: FF FF 00    231          HEX  FFFF0000    ;T.10
097B: 00
097C: 00 00 00    232          HEX  00000000    ;CAT.
097F: 00
0980: FF FF 00    233          HEX  FFFF0000    ;T.12
0983: 00
0984: FF FF 00    234          HEX  FFFF0000    ;T.13
0987: 00
0988: FF FF 00    235          HEX  FFFF0000    ;T.14
098B: 00
098C: FF FF 00    236          HEX  FFFF0000    ;T.15
098F: 00
0990: FF FF 00    237          HEX  FFFF0000    ;T.16
```

```
0993: 00
0994: FF FF 00  238          HEX  FFFF0000   ;T.17
0997: 00
0998: FF FF 00  239          HEX  FFFF0000   ;T.18
099B: 00
099C: FF FF 00  240          HEX  FFFF0000   ;T.19
099F: 00
09A0: FF FF 00  241          HEX  FFFF0000   ;T.1A
09A3: 00
09A4: FF FF 00  242          HEX  FFFF0000   ;T.1B
09A7: 00
09A8: FF FF 00  243          HEX  FFFF0000   ;T.1C
09AB: 00
09AC: FF FF 00  244          HEX  FFFF0000   ;T.1D
09AF: 00
09B0: FF FF 00  245          HEX  FFFF0000   ;T.1E
09B3: 00
09B4: FF FF 00  246          HEX  FFFF0000   ;T.1F
09B7: 00
09B8: FF FF 00  247          HEX  FFFF0000   ;T.20
09BB: 00
09BC: FF FF 00  248          HEX  FFFF0000   ;T.21
09BF: 00
09C0: FF FF 00  249          HEX  FFFF0000   ;T.22
09C3: 00
09C4: 00 00 00  250          HEX  00000000   ;ADDIT.
09C7: 00
09C8: 00 00 00  251          HEX  00000000
09CB: 00
09CC: 00 00 00  252          HEX  00000000
09CF: 00
09D0: 00 00 00  253          HEX  00000000
09D3: 00
09D4: 00 00 00  254          HEX  00000000
09D7: 00
09D8: 00 00 00  255          HEX  00000000
09DB: 00
09DC: 00 00 00  256          HEX  00000000
09DF: 00
09E0: 00 00 00  257          HEX  00000000
09E3: 00
09E4: 00 00 00  258          HEX  00000000
09E7: 00
09E8: 00 00 00  259          HEX  00000000
09EB: 00
09EC: 00 00 00  260          HEX  00000000
09EF: 00
09F0: 00 00 00  261          HEX  00000000
09F3: 00
09F4: 00 00 00  262          HEX  00000000
09F7: 00
09F8: 00 00 00  263          HEX  00000000
09FB: 00
09FC: 00 00 00  264          HEX  00000000
09FF: 00
```

6.2.9. RAM-Disk-Driver mit Kopierprogramm

RAMDISK64

```
BSAVE RAMDISK64, A$6080, L882
```

```
1             ORG  $6080
2    *
3    *
4    * RAMDISK64
5    * =========
6    *
7    * RAM-Diskdriver für 64K-Karte
8    * Apple IIe oder IIc für DOS 3.3
9    * in den unteren 48K/U.Stiehl/85
10   *
11   * Speicherkapazität
12   * 247  reine Datensektoren
13   *   3  Catalog-Sektoren: 21 Namen
14   *   1  VTOC-Sektor
15   *
16   * Übertragungsrate pro Sekunde
17   * 56.2K bei reiner RWTS (R/W)
18   * 40.0K bei Diversi-DOS (BLOAD)
19   *  3.4K bei DOS 3.3 (BLOAD)
20   *
21   * Speicherverteilung 64K-Karte
22   * $0000-$00FF: RAM-Diskdriver
23   * $0100-$FFFF: RAM-Disk, davon
24   * $0400-$07FF: frei (80 Z/Z)
25   *
26   * Sprungadressen in unteren 48K
27   * $BD00-$BD03: nach $03C0
28   * $03C0-$03CF: auf/von Karte
29   *
30   * Zum Vergleich RAM-Diskdriver
31   * der Firma Wagner Datentechnik:
32   * a) Es kann nur normales DOS 3.3
33   *    verwendet werden.
34   * b) Datenübertragung bei BLOAD
35   *    2.9K/s, bei Rwts 14.5K/s
36   * c) 240 Datensektoren, 5 Catalog-
37   *    sektoren.
38   *
39   PTR      EQU  $CE        ;-$CF
40   DOSWARM  EQU  $03D0
41   PRINT    EQU  $FDED
42   RDKEY    EQU  $FD0C
43   HOME     EQU  $FC58
44   *
```

```
6080: A2 00     45           LDX  #0
6082: BD B4 60  46   MITINIT1 LDA  MENUE,X
6085: F0 06     47           BEQ  JANEIN
6087: 20 ED FD  48           JSR  PRINT
608A: E8        49           INX
608B: D0 F5     50           BNE  MITINIT1
608D: 20 0C FD  51   JANEIN   JSR  RDKEY
6090: C9 CE     52           CMP  #"N"
6092: F0 11     53           BEQ  NURSOFT
6094: C9 EE     54           CMP  #"n"
6096: F0 0D     55           BEQ  NURSOFT
6098: C9 CA     56           CMP  #"J"
609A: F0 06     57           BEQ  MITINIT
609C: C9 EA     58           CMP  #"j"
609E: F0 02     59           BEQ  MITINIT
60A0: D0 EB     60           BNE  JANEIN
                61   *
60A2: 20 3F 61  62   MITINIT  JSR  INIT64
60A5: 20 CD 60  63   NURSOFT  JSR  SOFT64
                64   *
                65   * DOS-Patch, um auch die Catalog-
                66   * Sektoren $01-$0C von Track $11
                67   * als Datensektoren nutzen zu
                68   * können. Ggf. abschalten durch
                69   * 4CB060 anstelle von EAEAEA
                70   *
                71   * $B291: 18 69 11 CLC ADC #$11 alt
                72   * $B291: 18 69 11 CLC LDA #$11 neu
                73   *
60A8: EA EA EA  74           HEX  EAEAEA     ;4CB060
                75   *
60AB: A9 A9     76           LDA  #$A9       ;ADC>LDA
60AD: 8D 92 B2  77           STA  $B292
                78   *
                79   * Man ersetze EA durch 60, falls
                80   * RAMDISK64 aus einem Applesoft-
                81   * programm heraus mit BRUN RAMDISK64
                82   * gestartet werden soll.
                83   *
60B0: EA        84   INITENDE HEX  EA          ;60
                85   *
60B1: 4C D0 03  86           JMP  DOSWARM
                87   *
60B4: 8D        88   MENUE    HEX  8D
60B5: D2 C1 CD  89           ASC  "RAMDISK64 mit Init"
60B8: C4 C9 D3 CB B6 B4 A0 ED
60C0: E9 F4 A0 C9 EE E9 F4
60C7: A0 CA AF  90           ASC  " J/N "
60CA: CE A0
60CC: 00        91           HEX  00
                92   *
                93   *================================
                94   *
```

```
                     95  * SOFT64
                     96  * ======
                     97  *
                     98  * DOS-Patch bei RWTS-Entry
                     99  *
                    100  * $BD00: 84 48    STY $48
                    101  * $BD02: 85 49    STA $49
                    102  *
                    103  * $BD00: 4C C0 03 JMP $03C0
                    104  * $BD03: 00       BRK
                    105  *
60CD: A2 03         106  SOFT64   LDX  #3
60CF: BD 00 BD      107  DOSOKAY  LDA  $BD00,X
60D2: DD F3 60      108          CMP  BD00ALT,X
60D5: D0 34         109          BNE  NOTDOS
60D7: CA            110          DEX
60D8: 10 F5         111          BPL  DOSOKAY
                    112  *
60DA: A2 03         113          LDX  #3
60DC: BD F7 60      114  PATCH1   LDA  BD00NEU,X
60DF: 9D 00 BD      115          STA  $BD00,X
60E2: CA            116          DEX
60E3: 10 F7         117          BPL  PATCH1
                    118  *
                    119  * Sprünge auf/von Karte anlegen
                    120  *
                    121  * Sprung auf Karte (von RWTS)
                    122  * $03C0: 8E 09 C0 STX $C009
                    123  * $03C3: 4C 00 C0 JMP $0000
                    124  *
                    125  * Sprung von Karte (Non-RAM-Disk)
                    126  * $03C6: 8E 08 C0 STX $C008
                    127  * $03C9: 4C 04 BD JMP $BD04
                    128  *
                    129  * Sprung von Karte (I/O-Error)
                    130  * $03CC: 8E 08 C0 STX $C008
                    131  * $03CF: 60       RTS
                    132  *
60E5: A2 0F         133          LDX  #15
60E7: BD FB 60      134  PATCH2  LDA  03C0NEU,X
60EA: 9D C0 03      135          STA  $03C0,X
60ED: CA            136          DEX
60EE: 10 F7         137          BPL  PATCH2
60F0: 4C 2E 61      138          JMP  MOVER1
                    139  *
60F3: 84 48 85      140  BD00ALT  HEX  84488549
60F6: 49
60F7: 4C C0 03      141  BD00NEU  HEX  4CC00300
60FA: 00
                    142  *
60FB: 8E 09 C0      143  03C0NEU  HEX  8E09C0
60FE: 4C 00 00      144          HEX  4C0000
6101: 8E 08 C0      145          HEX  8E08C0
6104: 4C 04 BD      146          HEX  4C04BD
```

```
6107: 8E 08 C0  147           HEX  8E08C0
610A: 60        148           HEX  60
                149  *
                150  *********************************
                151  *
610B: A2 00     152  NOTDOS    LDX  #$00
610D: BD 1B 61  153  NOTDOS1   LDA  NOTDOS3,X
6110: F0 06     154           BEQ  NOTDOS2
6112: 20 ED FD  155           JSR  PRINT
6115: E8        156           INX
6116: D0 F5     157           BNE  NOTDOS1
6118: 4C B0 60  158  NOTDOS2   JMP  INITENDE
611B: 8D        159  NOTDOS3   HEX  8D
611C: C4 CF D3  160           ASC  "DOS modifiziert"
611F: A0 ED EF E4 E9 E6 E9 FA
6127: E9 E5 F2 F4
612B: 87 8D 00  161           HEX  878D00
                162  *
                163  *********************************
                164  *
                165  * Verschiebe "Driver" nach $0000
                166  *
612E: 8E 09 C0  167  MOVER1   STX  $C009      ;AUXZP
6131: A2 00     168           LDX  #$00
6133: BD 00 63  169  MOVER2   LDA  $6300,X
6136: 95 00     170           STA  $0000,X
6138: E8        171           INX
6139: D0 F8     172           BNE  MOVER2
613B: 8E 08 C0  173           STX  $C008      ;MAINZP
613E: 60        174           RTS             ;Exit
                175  *
                176  *================================
                177  *
                178  * INIT64
                179  * ======
                180  *
                181  * Softswitches normalisieren,
                182  * dann $0100-$FFFF auf 0 setzen
                183  *
613F: A9 00     184  INIT64   LDA  #$00       ;Clear
6141: 85 CE     185           STA  PTR        ;Lowbyte
                186  *
6143: 8C 00 C0  187           STY  $C000      ;OFF80
6146: 8C 02 C0  188           STY  $C002      ;MAINRD
6149: 8C 04 C0  189           STY  $C004      ;MAINWR
614C: 8C 08 C0  190           STY  $C008      ;MAINZP
                191  *
                192  * Bereich $0100-$01FF löschen
                193  *
614F: 8C 09 C0  194           STY  $C009      ;AUXZP
6152: A8        195           TAY
6153: 99 00 01  196  L0        STA  $0100,Y
6156: C8        197           INY
6157: D0 FA     198           BNE  L0
```

```
6159: 8C 08 C0  199          STY  $C008      ;MAINZP
                200  *
                201  * Bereich $0200-BFFF löschen
                202  *
615C: A0 02     203          LDY  #$02       ;$0200
615E: 84 CF     204          STY  PTR+1
6160: 8C 05 C0  205          STY  $C005      ;AUXWR
6163: A8        206          TAY
6164: 91 CE     207  L1      STA  (PTR),Y
6166: C8        208          INY
6167: D0 FB     209          BNE  L1
6169: E6 CF     210          INC  PTR+1
616B: A6 CF     211          LDX  PTR+1
616D: E0 C0     212          CPX  #$C0       ;$C000
616F: D0 F3     213          BNE  L1
6171: 8C 04 C0  214          STY  $C004      ;MAINWR
                215  *
                216  * Bereich $D000-FFFF Bk2 löschen
                217  *
6174: 8C 09 C0  218          STY  $C009      ;AUXZP
6177: AC 83 C0  219          LDY  $C083
617A: AC 83 C0  220          LDY  $C083      ;AUXBK2
617D: 8D 87 61  221          STA  L2+1
6180: A0 D0     222          LDY  #$D0
6182: 8C 88 61  223          STY  L2+2
6185: A8        224          TAY
6186: 99 00 D0  225  L2      STA  $D000,Y
6189: C8        226          INY
618A: D0 FA     227          BNE  L2
618C: EE 88 61  228          INC  L2+2
618F: D0 F5     229          BNE  L2         ;$FFFF+1
                230  *
                231  * Bereich $D000-DFFF Bk1 löschen
                232  *
6191: AC 8B C0  233          LDY  $C08B
6194: AC 8B C0  234          LDY  $C08B      ;AUXBK1
6197: 8D A1 61  235          STA  L3+1
619A: A0 D0     236          LDY  #$D0
619C: 8C A2 61  237          STY  L3+2
619F: A8        238          TAY
61A0: 99 00 D0  239  L3      STA  $D000,Y
61A3: C8        240          INY
61A4: D0 FA     241          BNE  L3
61A6: EE A2 61  242          INC  L3+2
61A9: AE A2 61  243          LDX  L3+2
61AC: E0 E0     244          CPX  #$E0       ;$E000
61AE: D0 F0     245          BNE  L3
                246  *
                247  * Language Card abstellen
                248  *
61B0: 8C 08 C0  249          STY  $C008      ;MAINZP
61B3: AC 81 C0  250          LDY  $C081      ;RDROM
61B6: AC 81 C0  251          LDY  $C081      ;WRBK2
                252  *
                253  **********************************
```

```
                    254  *
                    255  * Catalog-Spur anlegen
                    256  *
                    257  * $0200-$BFFF Aux. 64K schreiben
                    258  * $0200-$BFFF Main 64K lesen
                    259  *
61B9: 8C 05 C0      260          STY  $C005      ;AUXWR
                    261  *
                    262  * Track $11, Sektor $00 = VTOC
                    263  * nach $1000-$1FFF verschieben
                    264  *
61BC: A2 00         265          LDX  #0
61BE: BD E3 61      266  VTOC1   LDA  VTOC2,X
61C1: 9D 00 10      267          STA  $1000,X
61C4: E8            268          INX
61C5: D0 F7         269          BNE  VTOC1
                    270  *
                    271  * Link-Bytes der Catalog-Spur
                    272  * Nur 3 Sektoren für Dateinamen,
                    273  * d.h. 3 mal 7 = 21 Dateien max.
                    274  *
                    275  *         Byte: 0. 1. 2.  Adresse
                    276  * T 11, Sek 0F: 00 11 0E  $1F00
                    277  * T 11, Sek 0E: 00 11 0D  $1E00
                    278  * T 11, Sek 0D: 00 00 00  $1D00
                    279  *
61C7: A9 11         280          LDA  #$11       ;Track
61C9: 8D 01 1F      281          STA  $1F01
61CC: 8D 01 1E      282          STA  $1E01
61CF: A9 0E         283          LDA  #$0E       ;Sektor
61D1: 8D 02 1F      284          STA  $1F02
61D4: A9 0D         285          LDA  #$0D       ;Sektor
61D6: 8D 02 1E      286          STA  $1E02
                    287  *
61D9: 8C 04 C0      288          STY  $C004      ;MAINWR
61DC: 8C 01 C0      289          STY  $C001      ;ON80
61DF: 20 58 FC      290          JSR  HOME
61E2: 60            291          RTS             ;Exit
                    292  *
                    293  ********************************
                    294  *
                    295  * Track $11, Sektor $00 = VTOC
                    296  *
61E3: 00            297  VTOC2   HEX  00         ;unused
                    298  *
                    299  * Erster Catalog-Sektor T.11 S.0F
                    300  *
61E4: 11            301          HEX  11         ;T. $11
61E5: 0F            302          HEX  0F         ;S. $0F
                    303  *
61E6: 03            304          HEX  03         ;DOS 3.3
61E7: 00 00         305          HEX  0000       ;unused
61E9: FE            306          HEX  FE         ;Vol.254
                    307  *
```

```
61EA: 00 00 00  308          HEX  0000000000 ;unused
61ED: 00 00
61EF: 00 00 00  309          HEX  0000000000
61F2: 00 00
61F4: 00 00 00  310          HEX  0000000000
61F7: 00 00
61F9: 00 00 00  311          HEX  0000000000
61FC: 00 00
61FE: 00 00 00  312          HEX  0000000000
6201: 00 00
6203: 00 00 00  313          HEX  0000000000
6206: 00 00
6208: 00 00     314          HEX  0000
                315  *
                316  * Anzahl der T/S-Paare pro TSL
                317  *
620A: 7A        318          HEX  7A         ;122max.
620B: 00 00 00  319          HEX  00000000   ;unused
620E: 00
620F: 00 00 00  320          HEX  00000000
6212: 00
6213: 11        321          HEX  11         ;letzter T.
6214: 01        322          HEX  01         ;Richtung
6215: 00 00     323          HEX  0000       ;unused
6217: 23        324          HEX  23         ;35 Trk.
6218: 10        325          HEX  10         ;16 Sek.
6219: 00 01     326          HEX  0001       ;$0100
                327  *
                328  * Spurenbelegung-Bitmap
                329  * =====================
                330  *
                331  * Bit 1 = frei, Bit 0 = belegt
                332  *
                333  * Nichtbenutzte Spuren: $00-$0F
                334  *
                335  * FEDCBA9876543210
                336  * 0000000000000000 belegt
                337  *
621B: 00 00 00  338          HEX  00000000   ;T.00
621E: 00
621F: 00 00 00  339          HEX  00000000   ;T.01
6222: 00
6223: 00 00 00  340          HEX  00000000   ;T.02
6226: 00
6227: 00 00 00  341          HEX  00000000   ;T.03
622A: 00
622B: 00 00 00  342          HEX  00000000   ;T.04
622E: 00
622F: 00 00 00  343          HEX  00000000   ;T.05
6232: 00
6233: 00 00 00  344          HEX  00000000   ;T.06
6236: 00
6237: 00 00 00  345          HEX  00000000   ;T.07
623A: 00
```

```
623B: 00 00 00  346          HEX  00000000   ;T.08
623E: 00
623F: 00 00 00  347          HEX  00000000   ;T.09
6242: 00
6243: 00 00 00  348          HEX  00000000   ;T.0A
6246: 00
6247: 00 00 00  349          HEX  00000000   ;T.0B
624A: 00
624B: 00 00 00  350          HEX  00000000   ;T.0C
624E: 00
624F: 00 00 00  351          HEX  00000000   ;T.0D
6252: 00
6253: 00 00 00  352          HEX  00000000   ;T.0E
6256: 00
6257: 00 00 00  353          HEX  00000000   ;T.0F
625A: 00
                354  *
                355  * Teilbenutzte Spuren: $10-$11
                356  *
                357  * Auf Track $10 = $0000-$0FFF ist
                358  * Sektor    $00 = $0000-$00FF
                359  * stets belegt, doch könnte man
                360  * auch den Bereich $0400-$07FF
                361  * nutzen, falls keine 80 Z/Z
                362  * benötigt wird.
                363  *
                364  * FEDCBA9876543210
                365  * 1111111100001110 T.10 FF0E alt
                366  * 1111111111111110 T.10 FFFE neu
                367  *
625B: FF 0E 00  368  T10      HEX  FF0E0000   ;T.10
625E: 00
                369  *
                370  * Catalog: Es gibt 3 Dateinamen-
                371  * sektoren $0F, $0E und $0D sowie
                372  * einen VTOC-Sektor $00
                373  *
                374  * FEDCBA9876543210
                375  * 0001111111111110 T.11 Catalog
                376  *
625F: 1F FE 00  377  T11      HEX  1FFE0000   ;Catalog
6262: 00
                378  *
                379  * Vollbenutzte Spuren: $12-$1F
                380  *
                381  * FEDCBA9876543210
                382  * 1111111111111111 ganz frei
                383  *
                384  * T.12 = $2000-$2FFF
                385  * T.13 = $3000-$3FFF
                386  * usw.
                387  * T.1B = $B000-$BFFF
                388  * T.1C = $D000-$DFFF Bank1
                389  * T.1D = $D000-$DFFF Bank2
```

```
                      390  * usw.
                      391  * T.1F = $F000-$FFFF Bank2
                      392  *
6263: FF FF 00        393           HEX  FFFF0000   ;T.12
6266: 00
6267: FF FF 00        394           HEX  FFFF0000   ;T.13
626A: 00
626B: FF FF 00        395           HEX  FFFF0000   ;T.14
626E: 00
626F: FF FF 00        396           HEX  FFFF0000   ;T.15
6272: 00
6273: FF FF 00        397           HEX  FFFF0000   ;T.16
6276: 00
6277: FF FF 00        398           HEX  FFFF0000   ;T.17
627A: 00
627B: FF FF 00        399           HEX  FFFF0000   ;T.18
627E: 00
627F: FF FF 00        400           HEX  FFFF0000   ;T.19
6282: 00
6283: FF FF 00        401           HEX  FFFF0000   ;T.1A
6286: 00
6287: FF FF 00        402           HEX  FFFF0000   ;T.1B
628A: 00
628B: FF FF 00        403           HEX  FFFF0000   ;T.1C
628E: 00
628F: FF FF 00        404           HEX  FFFF0000   ;T.1D
6292: 00
6293: FF FF 00        405           HEX  FFFF0000   ;T.1E
6296: 00
6297: FF FF 00        406           HEX  FFFF0000   ;T.1F
629A: 00
                      407  *
                      408  * Nichtbenutzte Spuren: $20-$22
                      409  *
629B: 00 00 00        410           HEX  00000000   ;T.20
629E: 00
629F: 00 00 00        411           HEX  00000000   ;T.21
62A2: 00
62A3: 00 00 00        412           HEX  00000000   ;T.22
62A6: 00
62A7: 00 00 00        413           HEX  00000000   ;ADDIT.
62AA: 00
62AB: 00 00 00        414           HEX  00000000
62AE: 00
62AF: 00 00 00        415           HEX  00000000
62B2: 00
62B3: 00 00 00        416           HEX  00000000
62B6: 00
62B7: 00 00 00        417           HEX  00000000
62BA: 00
62BB: 00 00 00        418           HEX  00000000
62BE: 00
62BF: 00 00 00        419           HEX  00000000
62C2: 00
```

```
62C3: 00 00 00  420           HEX  00000000
62C6: 00
62C7: 00 00 00  421           HEX  00000000
62CA: 00
62CB: 00 00 00  422           HEX  00000000
62CE: 00
62CF: 00 00 00  423           HEX  000000
                424  *
                425  * RETTE IOB liegt in absoluter
                426  * Adresse bei $10EF-$10FF am Ende
                427  * der VTOC und nimmt 17 Bytes ein
                428  *
62D2: 8E 08 C0  429  RETTE    STX  $C008      ;MAINZP
62D5: 84 48     430           STY  $48        ;IOBLM
62D7: 85 49     431           STA  $49        ;IOBHM
62D9: 8E 09 C0  432           STX  $C009      ;AUXZP
62DC: 84 03     433           STY  $03        ;IOBL
62DE: 85 04     434           STA  $04        ;IOBH
62E0: 4C 11 00  435           JMP  $0011      ;PSTART3
                436  *
                437  **********************************
                438  *
62E3: D5 EC F2  439           ASC  "Ulrich Stiehl "
62E6: E9 E3 E8 A0 D3 F4 E9 E5
62EE: E8 EC A0
62F1: C8 E5 E9  440           ASC  "Heidelberg "
62F4: E4 E5 EC E2 E5 F2 E7 A0
62FC: B1 B9 B8  441           ASC  "1985"
62FF: B5
                442  *
                443  * Muß bei $62FF enden!
                444  *
                445  *================================
                446  *
                447  * Eigentlicher Driver
                448  *
                449           ORG  $0000      ;$6300!
                450  *
0000: 4C 0B 00  451  PSTART1  JMP  PSTART2    ;0000
                452  *
0003: 00        453  IOBL     HEX  00         ;0003
0004: 00        454  IOBH     HEX  00         ;0004
0005: 00        455  AUXL     HEX  00         ;NUR 00!
0006: 00        456  AUXH     HEX  00         ;0006
0007: 00        457  BUFL     HEX  00         ;0007
0008: 00        458  BUFH     HEX  00         ;0008
0009: 30        459  PSLOT    HEX  30         ;0009
000A: 02        460  PDRIVE   HEX  02         ;000A
                461  *
                462  * Rette IOB-Pointer
                463  *
000B: 8E 03 C0  464  PSTART2  STX  $C003      ;AUXRD
000E: 4C EF 10  465           JMP  $10EF      ;RETTE
                466  *
```

```
                    467  * Reentry PSOFT: $0011
                    468  *
0011: 8E 02 C0      469  PSTART3  STX  $C002      ;MAINRD
0014: AD 11 C0      470           LDA  $C011      ;BK1/2
0017: 85 E6         471           STA  BANKFLAG
0019: AD 12 C0      472           LDA  $C012      ;ROM/RAM
001C: 85 E7         473           STA  ROMFLAG
                    474  *
                    475  * Pseudo-Disk-Slot?
                    476  *
001E: A0 01         477  SLOT?    LDY  #$01
0020: B1 03         478           LDA  (IOBL),Y   ;Slot
0022: C5 09         479           CMP  PSLOT
                    480  *
                    481  * DRIVE? überspringen, falls
                    482  * Drive 1/2 gleichermaßen gilt
                    483  *
0024: F0 19         484           BEQ  RDWR?
                    485  *
                    486  * Zurück zur normalen Rwts
                    487  *
0026: A4 03         488  RWTS     LDY  IOBL       ;normale
0028: A5 04         489           LDA  IOBH
002A: 4C E9 00      490           JMP  EXIT1      ;Rwts
                    491  *
                    492  * Pseudo-Disk-Drive?
                    493  *
002D: A0 02         494  DRIVE?   LDY  #$02
002F: B1 03         495           LDA  (IOBL),Y   ;Drive
0031: C5 0A         496           CMP  PDRIVE
0033: F0 0A         497           BEQ  RDWR?
                    498  *
                    499  * Drive-Error
                    500  *
0035: A0 0D         501  IOERROR  LDY  #$0D
0037: A9 40         502           LDA  #$40       ;IO-Err.
0039: 91 03         503           STA  (IOBL),Y
003B: 38            504           SEC             ;Carry!
003C: 4C EC 00      505           JMP  EXIT2      ;IO-Err.
                    506  *
                    507  * Befehl: 1=Read oder 2=Write?
                    508  *
003F: A0 0C         509  RDWR?    LDY  #$0C
0041: B1 03         510           LDA  (IOBL),Y   ;Befehl
0043: C9 01         511           CMP  #$01       ;Read
0045: F0 04         512           BEQ  PUFFER
0047: C9 02         513           CMP  #$02       ;Write
0049: D0 EA         514           BNE  IOERROR
                    515  *
                    516  * Pufferadresse: $0200-$BF00
                    517  *
004B: 85 E8         518  PUFFER   STA  RDWRFLAG   ;RD/WR
004D: A0 08         519           LDY  #$08
004F: B1 03         520           LDA  (IOBL),Y   ;Puff-L
```

```
0051: 85 07     521           STA  BUFL
0053: C8        522           INY
0054: B1 03     523           LDA  (IOBL),Y   ;Puff-H
0056: 85 08     524           STA  BUFH
0058: C9 C0     525           CMP  #$C0       ;>$C000?
005A: B0 D9     526           BCS  IOERROR    ;No LC!
005C: C9 02     527           CMP  #$02       ;<$0200?
005E: 90 D5     528           BCC  IOERROR    ;Zero-P.
                529  *
                530  * Track und Sektor in Bereich
                531  * der 64K-Karte umwandeln
                532  *
                533  * T. $10 -> $0000 (nur teils)
                534  * T. $11 -> $1000
                535  * T. $12 -> $2000
                536  * T. $13 -> $3000
                537  * T. $14 -> $4000
                538  * T. $15 -> $5000
                539  * T. $16 -> $6000
                540  * T. $17 -> $7000
                541  * T. $18 -> $8000
                542  * T. $19 -> $9000
                543  * T. $1A -> $A000
                544  * T. $1B -> $B000
                545  * T. $1C -> $C000 wird Bk1 $D000
                546  * T. $1D -> $D000 Bk2
                547  * T. $1E -> $E000 Bk2
                548  * T. $1F -> $F000 Bk2
                549  *
0060: A0 04     550           LDY  #$04
0062: B1 03     551           LDA  (IOBL),Y   ;Track
0064: C9 20     552           CMP  #$20
0066: B0 CD     553           BCS  IOERROR    ;>$1F
0068: C9 10     554           CMP  #$10
006A: 90 C9     555           BCC  IOERROR    ;<$10
                556  *
006C: 0A        557           ASL             ;shiften
006D: 0A        558           ASL
006E: 0A        559           ASL
006F: 0A        560           ASL
                561  *
0070: C8        562           INY             ;Y=$05
0071: 18        563           CLC
0072: 71 03     564           ADC  (IOBL),Y   ;Sektor
0074: 85 06     565           STA  AUXH
0076: C9 01     566           CMP  #$01       ;>=$0100?
0078: 90 BB     567           BCC  IOERROR
                568  *
                569  * Bank 1? $C000 -> $D000
                570  *
007A: C9 C0     571  BANK1?    CMP  #$C0       ;$C000
007C: 90 17     572           BCC  READ?      ;<$C000
007E: C9 D0     573           CMP  #$D0       ;$D000
0080: B0 0D     574           BCS  BANK2      ;>=$D000
```

```
0082: 09 10     575           ORA  #%00010000 ;Cx->Dx
0084: 85 06     576           STA  AUXH
0086: AE 8B C0  577           LDX  $C08B
0089: AE 8B C0  578           LDX  $C08B
008C: 4C 95 00  579           JMP  READ?
                580 *
008F: AE 83 C0  581 BANK2     LDX  $C083
0092: AE 83 C0  582           LDX  $C083
                583 *
0095: A0 00     584 READ?     LDY  #0         ;Y=0
0097: A5 E8     585           LDA  RDWRFLAG
0099: C9 01     586           CMP  #1         ;1=Read
009B: D0 0F     587           BNE  WRITE
                588 *
                589 * Read = Aux. nach Main
                590 *
009D: 8E 03 C0  591 READ      STX  $C003      ;AUXRD
00A0: B1 05     592 READER    LDA  (AUXL),Y
00A2: 91 07     593           STA  (BUFL),Y
00A4: C8        594           INY
00A5: D0 F9     595           BNE  READER
00A7: 8E 02 C0  596           STX  $C002      ;MAINRD
00AA: F0 0D     597           BEQ  ZURUECK1
                598 *
                599 * Write = Main nach Aux.
                600 *
00AC: 8E 05 C0  601 WRITE     STX  $C005      ;AUXWR
00AF: B1 07     602 WRITER    LDA  (BUFL),Y
00B1: 91 05     603           STA  (AUXL),Y
00B3: C8        604           INY
00B4: D0 F9     605           BNE  WRITER
00B6: 8E 04 C0  606           STX  $C004      ;MAINWR
                607 *
00B9: 18        608 ZURUECK1  CLC             ;okay!
00BA: A0 0D     609           LDY  #$0D
00BC: A9 00     610           LDA  #$00
00BE: 91 03     611           STA  (IOBL),Y   ;o.Fehler
00C0: A0 0F     612           LDY  #$0F
00C2: A5 09     613           LDA  PSLOT
00C4: 91 03     614           STA  (IOBL),Y   ;Vorslot
                615 *
00C6: A5 E7     616           LDA  ROMFLAG
00C8: 30 08     617           BMI  ZURUECK2
00CA: AD 81 C0  618           LDA  $C081      ;RDROM
00CD: AD 81 C0  619           LDA  $C081      ;WRBK2
00D0: 90 1D     620           BCC  EXIT3
00D2: A5 E6     621 ZURUECK2  LDA  BANKFLAG
00D4: 10 08     622           BPL  ZURUECK3
00D6: AD 83 C0  623           LDA  $C083      ;BK2
00D9: AD 83 C0  624           LDA  $C083      ;RDWR
00DC: 90 11     625           BCC  EXIT3
00DE: AD 8B C0  626 ZURUECK3  LDA  $C08B      ;BK1
00E1: AD 8B C0  627           LDA  $C08B      ;RDWR
00E4: 90 09     628           BCC  EXIT3
```

```
                    629  *
00E6: 00            630  BANKFLAG HEX  00
00E7: 00            631  ROMFLAG  HEX  00
00E8: 00            632  RDWRFLAG HEX  00
                    633  *
                    634  **********************************
                    635  *
                    636  * Bei $63E9 vor der Verschiebung
                    637  *
00E9: 4C C6 03      638  EXIT1    JMP  $03C6      ;Rwts
00EC: 4C CC 03      639  EXIT2    JMP  $03CC      ;IO-Err.
00EF: 4C CC 03      640  EXIT3    JMP  $03CC      ;Okay
```

RAMDISK64.MOVER

```
BSAVE RAMDISK64.MOVER, A$0803, L1278

                      1            ORG  $0803
                      2  *
                      3  * RAMDISK64.MOVER
                      4  * ===============
                      5  *
                      6  * Zweck: Speichern/Laden des
                      7  * kompletten RAM-Disk-Inhalts
                      8  * auf Diskette/von Diskette.
                      9  *
                     10  * Auf der physischen Diskette
                     11  * werden 2 Binärfiles namens
                     12  * RAM1 (127 Sektoren) und
                     13  * RAM2 (130 Sektoren) angelegt
                     14  *
                     15  * Das L(aden) dauert nur 15s,
                     16  * während das S(peichern) vom
                     17  * verwendeten DOS abhängt.
                     18  *
                     19  * 24.11.85/U.Stiehl
                     20  *
                     21  IND      EQU  $CE
                     22  DOSCOLD  EQU  $03D3
                     23  RWTS     EQU  $03D9
                     24  PRINT    EQU  $FDED
                     25  RDKEY    EQU  $FD0C
                     26  HEXOUT   EQU  $FDDA
                     27  HOME     EQU  $FC58
                     28  PUFFER   EQU  $1000
                     29  *--------------------------------
0803: 4C 1F 08       30           JMP  START
0806: 06             31  USERSL   HEX  06         ;2054
```

```
0807: 01          32  USERDR   HEX  01          ;2055
0808: 01          33  IOB      HEX  01          ;stets
0809: 30          34  SLOT     HEX  30          ;3
080A: 01          35  DRIVE    HEX  01          ;1
080B: 00          36  VOLUME   HEX  00
080C: 00          37  TRACK    HEX  00
080D: 00          38  SECTOR   HEX  00
080E: 19          39           DFB  #<DCT
080F: 08          40           DFB  #>DCT
0810: 00          41  BUFLOW   HEX  00
0811: 10          42  BUFHIGH  HEX  10
0812: 00 00       43           HEX  0000        ;stets
0814: 01          44  COMMAND  HEX  01          ;1/2
0815: 00          45  DOSERROR HEX  00
0816: 00          46  VORVOL   HEX  00
0817: 60          47  VORSLOT  HEX  60
0818: 01          48  VORDRIVE HEX  01
0819: 00 01 EF    49  DCT      HEX  0001EFD8    ;stets
081C: D8
081D: 00          50  XREG     HEX  00
081E: 00          51  FLAG     HEX  00          ;SAV/LOD
                  52  *--------------------------------
081F: A9 80       53  START    LDA  #$80        ;RUN
0821: 85 33       54           STA  $33
0823: 85 76       55           STA  $76
0825: 85 D9       56           STA  $D9
0827: AD 06 08    57           LDA  USERSL      ;SL/DR
082A: 18          58           CLC
082B: 69 B0       59           ADC  #176
082D: 8D 80 09    60           STA  SLOTDRIV+2
0830: AD 07 08    61           LDA  USERDR
0833: 18          62           CLC
0834: 69 B0       63           ADC  #176
0836: 8D 83 09    64           STA  SLOTDRIV+5
0839: 20 33 09    65           JSR  DISKIN1
083C: AD 1E 08    66           LDA  FLAG
083F: C9 D3       67           CMP  #"S"
0841: F0 05       68           BEQ  VOMRAM
0843: C9 CC       69           CMP  #"L"
0845: F0 15       70           BEQ  ZUMRAM
0847: 60          71           RTS
                  72  *--------------------------------
                  73  *        Main: Vom RAM zur Disk .
0848: A9 01       74  VOMRAM   LDA  #1          ;Read
084A: 8D 14 08    75           STA  COMMAND
084D: 20 AF 08    76           JSR  BATCH1
0850: 20 B1 09    77           JSR  SAV1
0853: 20 E9 08    78           JSR  BATCH2
0856: 20 BF 09    79           JSR  SAV2
0859: 4C D3 03    80           JMP  DOSCOLD
                  81  *--------------------------------
                  82  *        Main: Zum RAM von Disk
085C: A9 01       83  ZUMRAM   LDA  #1
085E: 8D 14 08    84           STA  COMMAND
```

```
0861: AD 06 08  85            LDA  USERSL
0864: 0A        86            ASL
0865: 0A        87            ASL
0866: 0A        88            ASL
0867: 0A        89            ASL
0868: 8D 09 08  90            STA  SLOT
086B: AD 07 08  91            LDA  USERDR
086E: 8D 0A 08  92            STA  DRIVE
0871: 20 CD 09  93            JSR  LOD1
0874: 20 42 0C  94            JSR  TSLREAD
0877: A9 02     95            LDA  #2
0879: 8D 14 08  96            STA  COMMAND
087C: A9 30     97            LDA  #$30
087E: 8D 09 08  98            STA  SLOT
0881: 20 AF 08  99            JSR  BATCH1
0884: A9 01     100           LDA  #1
0886: 8D 14 08  101           STA  COMMAND
0889: AD 06 08  102           LDA  USERSL
088C: 0A        103           ASL
088D: 0A        104           ASL
088E: 0A        105           ASL
088F: 0A        106           ASL
0890: 8D 09 08  107           STA  SLOT
0893: AD 07 08  108           LDA  USERDR
0896: 8D 0A 08  109           STA  DRIVE
0899: 20 DB 09  110           JSR  LOD2
089C: 20 42 0C  111           JSR  TSLREAD
089F: A9 02     112           LDA  #2
08A1: 8D 14 08  113           STA  COMMAND
08A4: A9 30     114           LDA  #$30
08A6: 8D 09 08  115           STA  SLOT
08A9: 20 E9 08  116           JSR  BATCH2
08AC: 4C D3 03  117           JMP  DOSCOLD
                118 *--------------------------------
08AF: A9 00     119 BATCH1    LDA  #<PUFFER
08B1: 8D 10 08  120           STA  BUFLOW
08B4: A9 10     121           LDA  #>PUFFER
08B6: 8D 11 08  122           STA  BUFHIGH
08B9: A2 00     123           LDX  #0
08BB: 8E 1D 08  124           STX  XREG
08BE: BD 46 0A  125 BATCH1A   LDA  RAM1,X
08C1: C9 FF     126           CMP  #$FF       ;Endmark
08C3: F0 23     127           BEQ  BATCH1B
08C5: 8D 0C 08  128           STA  TRACK
08C8: E8        129           INX
08C9: BD 46 0A  130           LDA  RAM1,X
08CC: 8D 0D 08  131           STA  SECTOR
08CF: 8E 1D 08  132           STX  XREG
08D2: A9 08     133           LDA  #>IOB
08D4: A0 08     134           LDY  #<IOB
08D6: 20 D9 03  135           JSR  RWTS
08D9: B0 48     136           BCS  FEHLER
08DB: A9 00     137           LDA  #0
08DD: 85 48     138           STA  $48        ;P-REG
```

```
08DF: EE 11 08  139          INC  BUFHIGH
08E2: AE 1D 08  140          LDX  XREG
08E5: E8        141          INX
08E6: D0 D6     142          BNE  BATCH1A
08E8: 60        143 BATCH1B  RTS
                144 *
08E9: A9 00     145 BATCH2   LDA  #<PUFFER
08EB: 8D 10 08  146          STA  BUFLOW
08EE: A9 10     147          LDA  #>PUFFER
08F0: 8D 11 08  148          STA  BUFHIGH
08F3: A2 00     149          LDX  #0
08F5: 8E 1D 08  150          STX  XREG
08F8: BD 40 0B  151 BATCH2A  LDA  RAM2,X
08FB: C9 FF     152          CMP  #$FF       ;Endmark
08FD: F0 23     153          BEQ  BATCH2B
08FF: 8D 0C 08  154          STA  TRACK
0902: E8        155          INX
0903: BD 40 0B  156          LDA  RAM2,X
0906: 8D 0D 08  157          STA  SECTOR
0909: 8E 1D 08  158          STX  XREG
090C: A9 08     159          LDA  #>IOB
090E: A0 08     160          LDY  #<IOB
0910: 20 D9 03  161          JSR  RWTS
0913: B0 0E     162          BCS  FEHLER
0915: A9 00     163          LDA  #0
0917: 85 48     164          STA  $48        ;P-REG
0919: EE 11 08  165          INC  BUFHIGH
091C: AE 1D 08  166          LDX  XREG
091F: E8        167          INX
0920: D0 D6     168          BNE  BATCH2A
0922: 60        169 BATCH2B  RTS
                170 *
0923: 20 58 FC  171 FEHLER   JSR  HOME
0926: AD 15 08  172          LDA  DOSERROR
0929: 20 DA FD  173          JSR  HEXOUT
092C: A9 00     174          LDA  #0
092E: 85 48     175          STA  $48        ;P-REG
0930: 4C D3 03  176          JMP  DOSCOLD
                177 *
                178 * Diskette einlegen
                179 *
0933: 20 58 FC  180 DISKIN1  JSR  HOME
0936: A2 00     181          LDX  #0
0938: BD 6C 09  182 DISKIN2  LDA  DISKETTE,X
093B: F0 06     183          BEQ  DISKIN3
093D: 20 ED FD  184          JSR  PRINT
0940: E8        185          INX
0941: D0 F5     186          BNE  DISKIN2
0943: A2 00     187 DISKIN3  LDX  #0
0945: BD 86 09  188 DISKIN4  LDA  SAVLOD,X
0948: F0 06     189          BEQ  DISKIN5
094A: 20 ED FD  190          JSR  PRINT
094D: E8        191          INX
094E: D0 F5     192          BNE  DISKIN4
```

```
0950: 20 0C FD  193  DISKIN5  JSR  RDKEY
0953: C9 D3     194           CMP  #"S"       ;Save
0955: F0 04     195           BEQ  DISKIN6
0957: C9 CC     196           CMP  #"L"       ;Load
0959: D0 F5     197           BNE  DISKIN5
095B: 8D 1E 08  198  DISKIN6  STA  FLAG
095E: 60        199           RTS
                200  *
095F: A0 00     201  PRINTER  LDY  #0
0961: B1 CE     202  PRINTER1 LDA  (IND),Y
0963: F0 06     203           BEQ  PRINTER2
0965: 20 ED FD  204           JSR  PRINT
0968: C8        205           INY
0969: D0 F6     206           BNE  PRINTER1
096B: 60        207  PRINTER2 RTS
                208  *
096C: 8D        209  DISKETTE HEX  8D
096D: C4 E9 F3  210           ASC  "Diskette "
0970: EB E5 F4 F4 E5 A0
0976: E5 E9 EE  211           ASC  "einlegen"
0979: EC E5 E7 E5 EE
097E: AC D3 B6  212  SLOTDRIV ASC  ",S6,D1 "
0981: AC C4 B1 A0
0985: 00        213           HEX  00
                214  *
0986: 8D 8D 8D  215  SAVLOD   HEX  8D8D8D
0989: D3 A9 F0  216           ASC  "S)peichern"
098C: E5 E9 E3 E8 E5 F2 EE
0993: 8D 8D 8D  217           HEX  8D8D8D
0996: CC A9 E1  218           ASC  "L)aden"
0999: E4 E5 EE
099C: 8D 8D 8D  219           HEX  8D8D8D00
099F: 00
                220  *
09A0: A9 7E     221  SLDR     LDA  #<SLOTDRIV
09A2: 85 CE     222           STA  IND
09A4: A9 09     223           LDA  #>SLOTDRIV
09A6: 85 CF     224           STA  IND+1
09A8: 20 5F 09  225           JSR  PRINTER
09AB: A9 8D     226           LDA  #$8D
09AD: 20 ED FD  227           JSR  PRINT
09B0: 60        228           RTS
                229  *
09B1: A9 E9     230  SAV1     LDA  #<SAVRAM1
09B3: 85 CE     231           STA  IND
09B5: A9 09     232           LDA  #>SAVRAM1
09B7: 85 CF     233           STA  IND+1
09B9: 20 5F 09  234           JSR  PRINTER
09BC: 4C A0 09  235           JMP  SLDR
                236  *
09BF: A9 04     237  SAV2     LDA  #<SAVRAM2
09C1: 85 CE     238           STA  IND
09C3: A9 0A     239           LDA  #>SAVRAM2
09C5: 85 CF     240           STA  IND+1
```

```
09C7: 20 5F 09  241           JSR  PRINTER
09CA: 4C A0 09  242           JMP  SLDR
                243  *
09CD: A9 1F     244  LOD1      LDA  #<LODRAM1
09CF: 85 CE     245           STA  IND
09D1: A9 0A     246           LDA  #>LODRAM1
09D3: 85 CF     247           STA  IND+1
09D5: 20 5F 09  248           JSR  PRINTER
09D8: 4C A0 09  249           JMP  SLDR
                250  *
09DB: A9 38     251  LOD2      LDA  #<LODRAM2
09DD: 85 CE     252           STA  IND
09DF: A9 0A     253           LDA  #>LODRAM2
09E1: 85 CF     254           STA  IND+1
09E3: 20 5F 09  255           JSR  PRINTER
09E6: 4C A0 09  256           JMP  SLDR
                257  *
09E9: 8D 84     258  SAVRAM1  HEX  8D84
09EB: C2 D3 C1  259           ASC  "BSAVE "
09EE: D6 C5 A0
09F1: D2 C1 CD  260           ASC  "RAM1,"
09F4: B1 AC
09F6: C1 A4 B1  261           ASC  "A$1000,L$7C00"
09F9: B0 B0 B0 AC CC A4 B7 C3
0A01: B0 B0
0A03: 00        262           HEX  00
                263  *
0A04: 8D 84     264  SAVRAM2  HEX  8D84
0A06: C2 D3 C1  265           ASC  "BSAVE "
0A09: D6 C5 A0
0A0C: D2 C1 CD  266           ASC  "RAM2,"
0A0F: B2 AC
0A11: C1 A4 B1  267           ASC  "A$1000,L$7F00"
0A14: B0 B0 B0 AC CC A4 B7 C6
0A1C: B0 B0
0A1E: 00        268           HEX  00
                269  *
0A1F: 8D 84     270  LODRAM1  HEX  8D84
0A21: CD C1 D8  271           ASC  "MAXFILES3"
0A24: C6 C9 CC C5 D3 B3
0A2A: 8D 84     272           HEX  8D84
0A2C: D5 CE CC  273           ASC  "UNLOCK "
0A2F: CF C3 CB A0
0A33: D2 C1 CD  274           ASC  "RAM1"
0A36: B1
0A37: 00        275           HEX  00
                276  *
0A38: 8D 84     277  LODRAM2  HEX  8D84
0A3A: D5 CE CC  278           ASC  "UNLOCK "
0A3D: CF C3 CB A0
0A41: D2 C1 CD  279           ASC  "RAM2"
0A44: B2
0A45: 00        280           HEX  00
```

```
                        281  *
                        282  * 1. Teil: 124 Sektoren + 3
                        283  *
                        284  * $1000-$8BFF: 31744 Bytes
                        285  *
                        286  * RAM1, A$1000, L$7C00
                        287  *
0A46: 10 01             288  RAM1     HEX  1001       ;T.00
0A48: 10 02             289           HEX  1002
0A4A: 10 03             290           HEX  1003
0A4C: 10 08             291           HEX  1008
0A4E: 10 09             292           HEX  1009
0A50: 10 0A             293           HEX  100A
0A52: 10 0B             294           HEX  100B
0A54: 10 0C             295           HEX  100C
0A56: 10 0D             296           HEX  100D
0A58: 10 0E             297           HEX  100E
0A5A: 10 0F             298           HEX  100F
0A5C: 11 00             299           HEX  1100       ;T.01
0A5E: 11 01             300           HEX  1101
0A60: 11 02             301           HEX  1102
0A62: 11 03             302           HEX  1103
0A64: 11 04             303           HEX  1104
0A66: 11 05             304           HEX  1105
0A68: 11 06             305           HEX  1106
0A6A: 11 07             306           HEX  1107
0A6C: 11 08             307           HEX  1108
0A6E: 11 09             308           HEX  1109
0A70: 11 0A             309           HEX  110A
0A72: 11 0B             310           HEX  110B
0A74: 11 0C             311           HEX  110C
0A76: 11 0D             312           HEX  110D
0A78: 11 0E             313           HEX  110E
0A7A: 11 0F             314           HEX  110F
0A7C: 12 00             315           HEX  1200       ;T.12
0A7E: 12 01             316           HEX  1201
0A80: 12 02             317           HEX  1202
0A82: 12 03             318           HEX  1203
0A84: 12 04             319           HEX  1204
0A86: 12 05             320           HEX  1205
0A88: 12 06             321           HEX  1206
0A8A: 12 07             322           HEX  1207
0A8C: 12 08             323           HEX  1208
0A8E: 12 09             324           HEX  1209
0A90: 12 0A             325           HEX  120A
0A92: 12 0B             326           HEX  120B
0A94: 12 0C             327           HEX  120C
0A96: 12 0D             328           HEX  120D
0A98: 12 0E             329           HEX  120E
0A9A: 12 0F             330           HEX  120F
0A9C: 13 00             331           HEX  1300       ;T.13
0A9E: 13 01             332           HEX  1301
0AA0: 13 02             333           HEX  1302
0AA2: 13 03             334           HEX  1303
```

```
0AA4: 13 04     335         HEX  1304
0AA6: 13 05     336         HEX  1305
0AA8: 13 06     337         HEX  1306
0AAA: 13 07     338         HEX  1307
0AAC: 13 08     339         HEX  1308
0AAE: 13 09     340         HEX  1309
0AB0: 13 0A     341         HEX  130A
0AB2: 13 0B     342         HEX  130B
0AB4: 13 0C     343         HEX  130C
0AB6: 13 0D     344         HEX  130D
0AB8: 13 0E     345         HEX  130E
0ABA: 13 0F     346         HEX  130F
0ABC: 14 00     347         HEX  1400       ;T.14
0ABE: 14 01     348         HEX  1401
0AC0: 14 02     349         HEX  1402
0AC2: 14 03     350         HEX  1403
0AC4: 14 04     351         HEX  1404
0AC6: 14 05     352         HEX  1405
0AC8: 14 06     353         HEX  1406
0ACA: 14 07     354         HEX  1407
0ACC: 14 08     355         HEX  1408
0ACE: 14 09     356         HEX  1409
0AD0: 14 0A     357         HEX  140A
0AD2: 14 0B     358         HEX  140B
0AD4: 14 0C     359         HEX  140C
0AD6: 14 0D     360         HEX  140D
0AD8: 14 0E     361         HEX  140E
0ADA: 14 0F     362         HEX  140F
0ADC: 15 00     363         HEX  1500       ;T.15
0ADE: 15 01     364         HEX  1501
0AE0: 15 02     365         HEX  1502
0AE2: 15 03     366         HEX  1503
0AE4: 15 04     367         HEX  1504
0AE6: 15 05     368         HEX  1505
0AE8: 15 06     369         HEX  1506
0AEA: 15 07     370         HEX  1507
0AEC: 15 08     371         HEX  1508
0AEE: 15 09     372         HEX  1509
0AF0: 15 0A     373         HEX  150A
0AF2: 15 0B     374         HEX  150B
0AF4: 15 0C     375         HEX  150C
0AF6: 15 0D     376         HEX  150D
0AF8: 15 0E     377         HEX  150E
0AFA: 15 0F     378         HEX  150F
0AFC: 16 00     379         HEX  1600       ;T.16
0AFE: 16 01     380         HEX  1601
0B00: 16 02     381         HEX  1602
0B02: 16 03     382         HEX  1603
0B04: 16 04     383         HEX  1604
0B06: 16 05     384         HEX  1605
0B08: 16 06     385         HEX  1606
0B0A: 16 07     386         HEX  1607
0B0C: 16 08     387         HEX  1608
0B0E: 16 09     388         HEX  1609
```

```
0B10: 16 0A     389           HEX  160A
0B12: 16 0B     390           HEX  160B
0B14: 16 0C     391           HEX  160C
0B16: 16 0D     392           HEX  160D
0B18: 16 0E     393           HEX  160E
0B1A: 16 0F     394           HEX  160F
0B1C: 17 00     395           HEX  1700       ;T.17
0B1E: 17 01     396           HEX  1701
0B20: 17 02     397           HEX  1702
0B22: 17 03     398           HEX  1703
0B24: 17 04     399           HEX  1704
0B26: 17 05     400           HEX  1705
0B28: 17 06     401           HEX  1706
0B2A: 17 07     402           HEX  1707
0B2C: 17 08     403           HEX  1708
0B2E: 17 09     404           HEX  1709
0B30: 17 0A     405           HEX  170A
0B32: 17 0B     406           HEX  170B
0B34: 17 0C     407           HEX  170C
0B36: 17 0D     408           HEX  170D
0B38: 17 0E     409           HEX  170E
0B3A: 17 0F     410           HEX  170F
0B3C: 18 00     411           HEX  1800       ;T.18
0B3E: FF FF     412           HEX  FFFF       ;Endmark
                413  *
                414  ********************************
                415  *
                416  * 2. Teil: 127 Sektoren + 3
                417  *
                418  * $1000-$8EFF: 32512 Bytes
                419  *
                420  * RAM2, A$1000, L$7F00
                421  *
0B40: 18 01     422  RAM2     HEX  1801       ;T.18
0B42: 18 02     423           HEX  1802
0B44: 18 03     424           HEX  1803
0B46: 18 04     425           HEX  1804
0B48: 18 05     426           HEX  1805
0B4A: 18 06     427           HEX  1806
0B4C: 18 07     428           HEX  1807
0B4E: 18 08     429           HEX  1808
0B50: 18 09     430           HEX  1809
0B52: 18 0A     431           HEX  180A
0B54: 18 0B     432           HEX  180B
0B56: 18 0C     433           HEX  180C
0B58: 18 0D     434           HEX  180D
0B5A: 18 0E     435           HEX  180E
0B5C: 18 0F     436           HEX  180F
0B5E: 19 00     437           HEX  1900       ;T.19
0B60: 19 01     438           HEX  1901
0B62: 19 02     439           HEX  1902
0B64: 19 03     440           HEX  1903
0B66: 19 04     441           HEX  1904
0B68: 19 05     442           HEX  1905
```

```
0B6A: 19 06     443         HEX  1906
0B6C: 19 07     444         HEX  1907
0B6E: 19 08     445         HEX  1908
0B70: 19 09     446         HEX  1909
0B72: 19 0A     447         HEX  190A
0B74: 19 0B     448         HEX  190B
0B76: 19 0C     449         HEX  190C
0B78: 19 0D     450         HEX  190D
0B7A: 19 0E     451         HEX  190E
0B7C: 19 0F     452         HEX  190F
0B7E: 1A 00     453         HEX  1A00       ;T.1A
0B80: 1A 01     454         HEX  1A01
0B82: 1A 02     455         HEX  1A02
0B84: 1A 03     456         HEX  1A03
0B86: 1A 04     457         HEX  1A04
0B88: 1A 05     458         HEX  1A05
0B8A: 1A 06     459         HEX  1A06
0B8C: 1A 07     460         HEX  1A07
0B8E: 1A 08     461         HEX  1A08
0B90: 1A 09     462         HEX  1A09
0B92: 1A 0A     463         HEX  1A0A
0B94: 1A 0B     464         HEX  1A0B
0B96: 1A 0C     465         HEX  1A0C
0B98: 1A 0D     466         HEX  1A0D
0B9A: 1A 0E     467         HEX  1A0E
0B9C: 1A 0F     468         HEX  1A0F
0B9E: 1B 00     469         HEX  1B00       ;T.1B
0BA0: 1B 01     470         HEX  1B01
0BA2: 1B 02     471         HEX  1B02
0BA4: 1B 03     472         HEX  1B03
0BA6: 1B 04     473         HEX  1B04
0BA8: 1B 05     474         HEX  1B05
0BAA: 1B 06     475         HEX  1B06
0BAC: 1B 07     476         HEX  1B07
0BAE: 1B 08     477         HEX  1B08
0BB0: 1B 09     478         HEX  1B09
0BB2: 1B 0A     479         HEX  1B0A
0BB4: 1B 0B     480         HEX  1B0B
0BB6: 1B 0C     481         HEX  1B0C
0BB8: 1B 0D     482         HEX  1B0D
0BBA: 1B 0E     483         HEX  1B0E
0BBC: 1B 0F     484         HEX  1B0F
0BBE: 1C 00     485         HEX  1C00       ;T.1C
0BC0: 1C 01     486         HEX  1C01
0BC2: 1C 02     487         HEX  1C02
0BC4: 1C 03     488         HEX  1C03
0BC6: 1C 04     489         HEX  1C04
0BC8: 1C 05     490         HEX  1C05
0BCA: 1C 06     491         HEX  1C06
0BCC: 1C 07     492         HEX  1C07
0BCE: 1C 08     493         HEX  1C08
0BD0: 1C 09     494         HEX  1C09
0BD2: 1C 0A     495         HEX  1C0A
0BD4: 1C 0B     496         HEX  1C0B
```

```
0BD6: 1C 0C     497           HEX  1C0C
0BD8: 1C 0D     498           HEX  1C0D
0BDA: 1C 0E     499           HEX  1C0E
0BDC: 1C 0F     500           HEX  1C0F
0BDE: 1D 00     501           HEX  1D00       ;T.1D
0BE0: 1D 01     502           HEX  1D01
0BE2: 1D 02     503           HEX  1D02
0BE4: 1D 03     504           HEX  1D03
0BE6: 1D 04     505           HEX  1D04
0BE8: 1D 05     506           HEX  1D05
0BEA: 1D 06     507           HEX  1D06
0BEC: 1D 07     508           HEX  1D07
0BEE: 1D 08     509           HEX  1D08
0BF0: 1D 09     510           HEX  1D09
0BF2: 1D 0A     511           HEX  1D0A
0BF4: 1D 0B     512           HEX  1D0B
0BF6: 1D 0C     513           HEX  1D0C
0BF8: 1D 0D     514           HEX  1D0D
0BFA: 1D 0E     515           HEX  1D0E
0BFC: 1D 0F     516           HEX  1D0F
0BFE: 1E 00     517           HEX  1E00       ;T.1E
0C00: 1E 01     518           HEX  1E01
0C02: 1E 02     519           HEX  1E02
0C04: 1E 03     520           HEX  1E03
0C06: 1E 04     521           HEX  1E04
0C08: 1E 05     522           HEX  1E05
0C0A: 1E 06     523           HEX  1E06
0C0C: 1E 07     524           HEX  1E07
0C0E: 1E 08     525           HEX  1E08
0C10: 1E 09     526           HEX  1E09
0C12: 1E 0A     527           HEX  1E0A
0C14: 1E 0B     528           HEX  1E0B
0C16: 1E 0C     529           HEX  1E0C
0C18: 1E 0D     530           HEX  1E0D
0C1A: 1E 0E     531           HEX  1E0E
0C1C: 1E 0F     532           HEX  1E0F
0C1E: 1F 00     533           HEX  1F00       ;T.1F
0C20: 1F 01     534           HEX  1F01
0C22: 1F 02     535           HEX  1F02
0C24: 1F 03     536           HEX  1F03
0C26: 1F 04     537           HEX  1F04
0C28: 1F 05     538           HEX  1F05
0C2A: 1F 06     539           HEX  1F06
0C2C: 1F 07     540           HEX  1F07
0C2E: 1F 08     541           HEX  1F08
0C30: 1F 09     542           HEX  1F09
0C32: 1F 0A     543           HEX  1F0A
0C34: 1F 0B     544           HEX  1F0B
0C36: 1F 0C     545           HEX  1F0C
0C38: 1F 0D     546           HEX  1F0D
0C3A: 1F 0E     547           HEX  1F0E
0C3C: 1F 0F     548           HEX  1F0F
0C3E: FF FF     549           HEX  FFFF       ;Endmark
                550 *-------------------------------
```

```
                    551  TOTALTSL EQU  $9000      ;-$94C4
                    552  MACPUFF  EQU  $1000-4    ;-$8A00
                    553  *
                    554  * Maxfiles muß 3 sein!
                    555  *
                    556  TSLPUFF  EQU  $9700      ;Anfang
                    557  TSLPTR   EQU  $9701      ;Pointer
                    558  TSL      EQU  $970C      ;eig.TSL
                    559  *
0C40: 00            560  YSAVER   HEX  00
0C41: 00            561  XSAVER   HEX  00
                    562  *
                    563  * Total-TSL initialisieren
                    564  *
0C42: A9 00         565  TSLREAD  LDA  #<TOTALTSL
0C44: 85 CE         566           STA  IND
0C46: A9 90         567           LDA  #>TOTALTSL
0C48: 85 CF         568           STA  IND+1
0C4A: A9 00         569           LDA  #<TSLPUFF
0C4C: 8D 10 08      570           STA  BUFLOW
0C4F: A9 97         571           LDA  #>TSLPUFF
0C51: 8D 11 08      572           STA  BUFHIGH
0C54: 4C 6E 0C      573           JMP  SSTORE1     ;ERSTE
0C57: AD 01 97      574  NEXTTSL  LDA  TSLPTR
0C5A: F0 35         575           BEQ  BATCHER1    ;LETZTE
0C5C: 8D 0C 08      576           STA  TRACK
0C5F: AD 02 97      577           LDA  TSLPTR+1
0C62: 8D 0D 08      578           STA  SECTOR
0C65: A9 08         579           LDA  #>IOB
0C67: A0 08         580           LDY  #<IOB
0C69: 20 D9 03      581           JSR  RWTS
0C6C: B0 17         582           BCS  ERROR1
0C6E: A2 00         583  SSTORE1  LDX  #$00
0C70: A0 00         584           LDY  #$00
0C72: BD 0C 97      585  SSTORE2  LDA  TSL,X
0C75: 91 CE         586           STA  (IND),Y
0C77: E6 CE         587           INC  IND
0C79: D0 02         588           BNE  SSTORE3
0C7B: E6 CF         589           INC  IND+1
0C7D: E8            590  SSTORE3  INX
0C7E: E0 F4         591           CPX  #244        ;122*2
0C80: D0 F0         592           BNE  SSTORE2     ;MAX.
0C82: 4C 57 0C      593           JMP  NEXTTSL
                    594  *
                    595  * Lesefehler
                    596  *
0C85: 20 58 FC      597  ERROR1   JSR  HOME
0C88: AD 15 08      598           LDA  DOSERROR
0C8B: 20 DA FD      599           JSR  HEXOUT
0C8E: 4C D3 03      600           JMP  DOSCOLD
                    601  *
                    602  * Alle Sektoren auf einmal lesen
                    603  *
```

```
0C91: A0 00     604  BATCHER1 LDY  #0
0C93: 8C 40 0C  605           STY  YSAVER
0C96: A9 00     606           LDA  #<TOTALTSL
0C98: 85 CE     607           STA  IND
0C9A: A9 90     608           LDA  #>TOTALTSL
0C9C: 85 CF     609           STA  IND+1
                610  *
0C9E: AC 40 0C  611           LDY  YSAVER
0CA1: B1 CE     612           LDA  (IND),Y
0CA3: D0 07     613           BNE  BATCHER2
0CA5: A9 00     614           LDA  #0
0CA7: 85 48     615           STA  $48        ;P-REG
0CA9: 4C D3 03  616           JMP  DOSCOLD    ;leer!
                617  *
0CAC: A2 00     618  BATCHER2 LDX  #0
0CAE: 8D 41 0C  619           STA  XSAVER
0CB1: A9 FC     620           LDA  #<MACPUFF
0CB3: 8D 10 08  621           STA  BUFLOW
0CB6: A9 0F     622           LDA  #>MACPUFF
0CB8: 8D 11 08  623           STA  BUFHIGH
0CBB: CE 11 08  624           DEC  BUFHIGH
                625  *
0CBE: AC 40 0C  626  BATCHER3 LDY  YSAVER
0CC1: B1 CE     627           LDA  (IND),Y
0CC3: F0 37     628           BEQ  BATCHER7
0CC5: EE 11 08  629           INC  BUFHIGH
0CC8: 8D 0C 08  630           STA  TRACK
0CCB: EE 40 0C  631           INC  YSAVER
0CCE: AC 40 0C  632           LDY  YSAVER
0CD1: D0 02     633           BNE  BATCHER4
0CD3: E6 CF     634           INC  IND+1
0CD5: B1 CE     635  BATCHER4 LDA  (IND),Y
0CD7: 8D 0D 08  636           STA  SECTOR
0CDA: A9 08     637           LDA  #>IOB
0CDC: A0 08     638           LDY  #<IOB
0CDE: 20 D9 03  639           JSR  RWTS
0CE1: 90 03     640           BCC  BATCHER5
0CE3: 4C 85 0C  641           JMP  ERROR1
0CE6: EE 40 0C  642  BATCHER5 INC  YSAVER
0CE9: AC 40 0C  643           LDY  YSAVER
0CEC: D0 02     644           BNE  BATCHER6
0CEE: E6 CF     645           INC  IND+1
0CF0: AE 41 0C  646  BATCHER6 LDX  XSAVER
0CF3: E8        647           INX
0CF4: E8        648           INX
0CF5: 8E 41 0C  649           STX  XSAVER
0CF8: E0 FF     650           CPX  #255
0CFA: 90 C2     651           BCC  BATCHER3
0CFC: A9 00     652  BATCHER7 LDA  #0
0CFE: 85 48     653           STA  $48        ;P-REG
0D00: 60        654           RTS
```

Register

Begleitdiskette zu „Apple DOS 3.3 — Tips und Tricks“

DM 28,—

ISBN 3-7785-1033-9

A 002 STIEHL
T 004 BLOAD.FINDER
T 003 LIST.MAKER
B 002 TRACK.SECTOR.TRACER
T 004 T.TRACK.SECTOR.TRACER
B 002 FREIE.SEKTOREN
T 003 T.FREIE.SEKTOREN
B 002 FUSION.TASC
T 006 T.FUSION.TASC
A 004 CPM.REFINER
B 002 CPM.REFINER.OBJ
T 006 T.CPM.REFINER.OBJ
B 002 DOS.OUTPUT.VEKTOR
T 008 T.DOS.OUTPUT.VEKTOR
B 002 RWTS.MUSTER
T 011 T.RWTS.MUSTER
B 002 RWTS.TEST.WARTESCHLEIFE
T 009 T.RWTS.TEST.WARTESCHLEIFE
B 002 RWTS.TEST.SKEWING
T 005 T.RWTS.TEST.SKEWING
A 003 DISK.LESER
B 002 DISK.LESER.OBJ
T 007 T.DISK.LESER.OBJ
A 004 FILE.LESER
B 003 FILE.LESER.OBJ
T 013 T.FILE.LESER.OBJ
B 004 KOPIERPROGRAMM.2DRIVE
T 018 T.KOPIERPROGRAMM.2DRIVE
B 005 KOPIERPROGRAMM.1DRIVE
T 020 T.KOPIERPROGRAMM.1DRIVE
B 004 DISK.COMPARER
T 016 T.DISK.COMPARER
B 003 BAD.SECTOR.ROUTINE
T 010 T.BAD.SECTOR.ROUTINE
A 003 DOS.KOPIE
B 002 DOS.KOPIE.OBJ
T 008 T.DOS.KOPIE.OBJ
A 002 DOSLOS
B 004 DOSLOS.OBJ
T 017 T.DOSLOS.OBJ
B 005 RAMDISK64
T 040 T.RAMDISK64
B 007 RAMDISK64.MOVER
T 032 T.RAMDISK64.MOVER
A 024 HELLO
B 114 ASMDIV
A 002 DIVERSI.START

Hans Gabriel

Hüthig

Das Buch zum Apple-Writer II/IIe

1986, ca. 200 S., kart.,
ca. DM 55,—
ISBN 3-7785-1234-X

„Das Buch zum Apple Writer II/IIe" wendet sich an alle, die dieses Textverarbeitungssystem schon einsetzen oder einsetzen wollen.

In einzelnen, in sich geschlossenen Kapiteln erlernt der Leser alle Funktionen und erzielt schnelle Erfolgserlebnisse. Im ersten Kapitel werden typische Arbeitsstellungen der Textverarbeitung und ihre systematische Bearbeitung vorgestellt. Das zweite Kapitel stellt in logischen Funktionsgruppen die Befehle vor mit denen der Applewriter während der Textarbeit direkt gesteuert werden kann. Kapitel 3 zeigt die Programmierung des Applewriters in der Text-Kommando-Sprache TKS (WPL). Dazu werden Beispiele analysiert. Auf der Begleitdiskette zum Buch, die seperat bezogen werden kann, sind darüber hinaus umfangreiche Schwerpunkterklärungen enthalten, die über die Help-Funktion vom Benutzer abgerufen werden können. Eine kleine Adreßdatenbank mit den dazugehörigen Dienstprogrammen zur Pflege der Daten und zu ihren Einsatz in Einzel- und Serienbriefen befindet sich ebenfalls auf Diskette. „Das Buch zum Apple Writer II/IIe" behandelt sowohl die alte Programmverison für den Apple IIplus als auch die neue Ausgabe, die 128-kByte auf dem Apple IIe oder Apple IIc unterstützt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Ulrich Stiehl

Hüthig

Apple-Assembler

1984, 200 S., 3 Abb., kart.,
DM 34,—
ISBN 3-7785-1047-9

„Apple Assembler" wendet sich an alle, die bereits Anfängerkenntnisse der 6502-Programmierung haben - z. B. aufgrund des Buches „Apple Maschinensprache" - und nunmehr ein Nachschlagewerk für ihren Apple II Plus/IIe/IIc suchen, in dem alle wichtigen ROM-Routinen sowie eine Vielzahl sonstiger Hilfsprogramme in seiner systematischen Form zusammengestellt werden. Insgesamt umfaßt dieses Buch über 40 Utilities, darunter mehrere völlig neuartige Programme wie Double-Lores, Double Hilres, Screen-Format u.a.

Der erste Teil enthält ein Repetorium der wichtigsten Befehle, Adressierungsarten und sonstigen Besonderheiten des 6502.

Im zweiten Teil werden alle Adressen des Monitors zusammengestellt, die für Assembler-Programmierer von Nutzen sein können. Darüber hinaus findet der Leser Unterroutinen für hexadezimale Addition/Subtraktion/Multiplikation/Division, Binär-Hex-ASCII-Umwandlung usw.
Der dritte Teil befaßt sich mit der Speicherverwaltung der Language Card und der IIe-64K-Karte und enthält Move-Programme zum Verschieben von Daten in die und aus der Language Card sowie der 64K-Karte.

Der vierte Teil ist dem Applesoft-ROM gewidmet und listet eine große Anzahl nützlicher Interpreter-Adressen. Bei den Utility-Programmen liegt das Schwergewicht auf Fließkommamathematik einschließlich Print Using.

Der letzte Teil behandelt den Text- und Graphikspeicher. Neben einem professionellen Maskengeneratorprogramm werden auch Routinen zur Double-Lores und Double-Hires-Grafik vorgestellt.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Dies ist die erste deutschsprachige Darstellung des Diskettenbetriebssystems DOS 3.3 für den Apple II/II Plus/IIe, die sich sowohl an Applesoft- als auch an Assembler-Programmierer wendet. Sinngemäß ist das Buch zweigeteilt:

Der erste Teil behandelt ausführlich die dem Applesoft-Programmierer zur Verfügung stehenden DOS-Befehle, wobei die Textfiles wegen ihrer großen Bedeutung und der vergleichsweise komplizierten Handhabung besonders dargestellt werden. Viele Textfile-Tricks aus der langjährigen Programmiererfahrung des Autors werden hier zum erstenmal geschildert.

Aber auch im zweiten Teil findet der reine Applesoft-Programmierer insbesondere in dem Kapitel „Vermischte Tips, Tricks und Patches" zahlreiche Anregungen. Im übrigen ist der zweite Teil für Assembler-Programmierer gedacht, der hier vollständige RWTS-Anwendungsprogramme vorfindet, die für diese dritte Auflage völlig überarbeitet wurden. Dieses DOS-Buch ist deshalb der unentbehrlich Begleiter für jeden Apple-Programmierer.