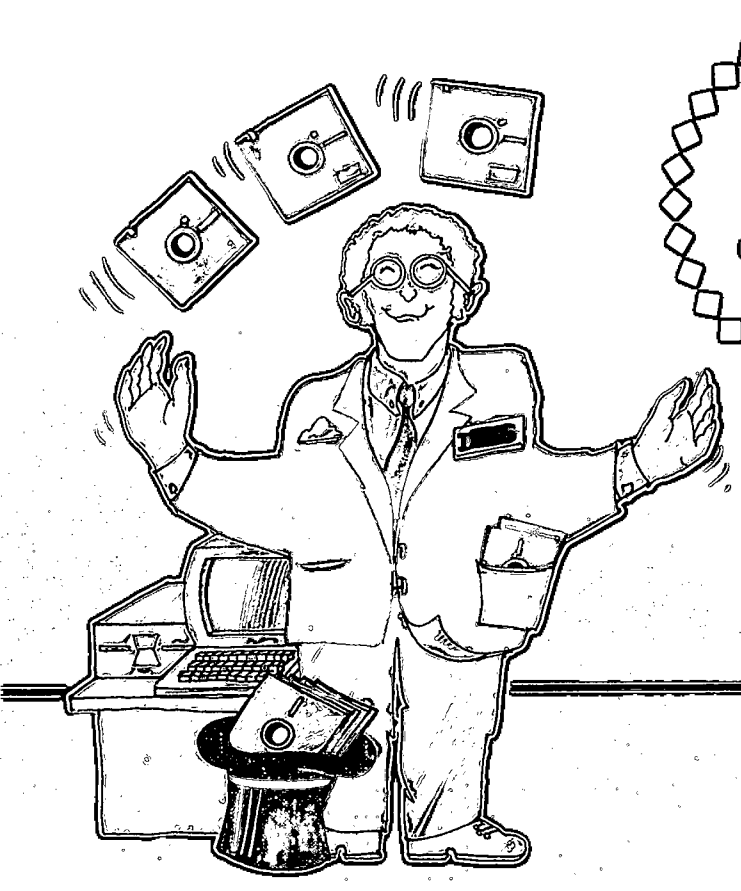
Apple ProDOS

für Aufsteiger

Mit ausführlichen Programmbeispielen

Ulrich Stiehl



Tips und Tricks

2

Hüthig

Ulrich Stiehl · ProDOS für Aufsteiger Band 2

Ulrich Stiehl

ProDOS für Aufsteiger

Mit ausführlichen Programmbeispielen

Band 2

Dr. Alfred Hüthig Verlag Heidelberg

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
„Begleitdiskette zu den ProDOS für Aufsteiger, Band 2"
ISBN 3-77895-1036-3

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stiehl, Ulrich
ProDOS für Aufsteiger, Band 2
Mit ausführlichen Programmbeispielen

– Heidelberg: Hüthig, 1985
ISBN 3-7785-1040-1

Vorwort

Während sich der erste Band dieses zweibändigen ProDOS-Buches vorwiegend mit der externen und internen Speicherorganisation sowie mit dem Machine Language Interface befaßte und damit notgedrungen relativ theoretisch ausfiel, ist dieser zweite Band wieder „aus der Praxis für die Praxis" geschrieben.
Zunächst wird eine ausführliche und bewußt leichtverständliche Einführung in das BASIC.SYSTEM gegeben, die all denen, die noch nicht das ältere Betriebssystem DOS 3.3 kennen, einen raschen Einstieg in die Applesoft-Programmierung unter ProDOS ermöglicht. Im Anschluß daran werden vermischte Tips und Tricks vermittelt, die einige der wenigen Unzulänglichkeiten des BASIC.SYSTEMs beheben.

Wer mit einem Betriebssystem vernünftig arbeiten will, braucht eine Reihe von Utilities, die aus Gründen der Effizienz in der Regel in Maschinensprache geschrieben werden müssen. Deshalb enthält der dritte Teil eine Sammlung von Assembler-Utilities, die teils „Stand-alone"-Programme und teils Hilfsroutinen sind, die sich in eigene Applesoft-Programme einbauen lassen. Zur *Anwendung* dieser Utilities sind keinerlei Assemblerkenntnisse erforderlich. Es wird empfohlen, die Begleitdiskette zu diesem Buch zu erwerben, da das Abtippen von Maschinenprogrammen relativ fehlerbehaftet ist.

Als der erste Band dieses Buches erschien, war noch nicht abzusehen, daß die Firma Apple in sehr kurzen Intervallen neue Versionen („Releases") zu PRODOS und zum BASIC.SYSTEM herausbringen würde. Insbesondere wurde nicht damit gerechnet, daß jede Neuversion völlig neu assembliert werden würde, selbst wenn praktisch nur hier und dort ein Bug zu patchen war. Folglich werden alle versionsabhängigen Programme mit dem Erscheinen einer neuen Version zur Makulatur, falls mit absoluten Adressen gearbeitet wird. Aus diesem Grunde wurde dafür Sorge getragen, daß *alle* Programme dieses Buches mit *allen* bislang bekannten PRODOS- und BASIC.SYSTEM-Versionen laufen.

Heidelberg, im Mai 1985 Ulrich Stiehl

Inhalt

1.	**ProDOS für Applesoft-Programmierer**	9
1.1.	Die ersten Schritte in ProDOS	9
1.2.	DOS 3.3 und ProDOS	13
1.3.	Monitor, Applesoft und ProDOS	14
1.4.	Bootvorgang	14
1.5.	Externe Speicherorganisation	16
1.6.	Interne Speicherorganisation	18
1.7.	Befehlstypen	19
1.8.	Parametertypen	23
1.9.	Dateitypen	24
1.10.	CAT, CATALOG, PREFIX, CREATE	27
1.11.	VERIFY, LOCK, UNLOCK, DELETE, RENAME	36
1.12.	RUN, LOAD, SAVE, CHAIN, STORE, RESTORE, FRE	39
1.13.	BRUN, BLOAD, BSAVE	44
1.14.	Strichbefehl, BYE	48
1.15.	EXEC	49
1.16.	PR# und IN#	52
1.17.	OPEN, READ, WRITE, CLOSE, FLUSH, APPEND, POSITION	53
2.	**Vermischte Tips und Tricks**	64
2.1.	BASIC.SYSTEM Global Page	65
2.2.	LOMEM, FRE und HGR	79
2.3.	HIMEM und Maschinenprogramme	81
2.4.	Open-Dateien bei 1-Drive-Besitzern	82
2.5.	Input von Strings mit Komma und Doppelpunkt	84
2.6.	Zahlen-Arrays als Binärfiles	87
2.7.	Simulierter MON-Befehl	91
2.8.	MON.COMMANDS	92
2.9.	FP.COMMAND	98
2.10.	BITMAP-Anzeige	103

3.	**ProDOS-Utilites**	106
3.1.	Catalogleseprogramme	106
3.1.1.	EINTRAG.SUCHER	106
3.1.2.	EINTRAG.ANALYSE	107
3.1.3.	CAT.ARRAY	109
3.1.4.	CAT.SAVER	110
3.2.	Dateileseprogramme	110
3.2.1.	FILE.READ.ASC und ASC	110
3.2.2.	FILE.READ.HEX und HEX	113
3.3.	Dateikopierprogramme	116
3.3.1.	PRODOS.FID.A und PRODOS.FID.O	116/118
3.3.2.	PROFID.DEMO und PROFID	127
3.4.	Diskettenkopierprogramme	147
3.4.1.	PRODOS.COPY.A. und PRODOS.COPY.O	147
3.4.2.	QUICKCOPYPUFFER und QUICKCOPY	154
3.5.	Formatierprogramm	165
3.5.1.	FORMAT.A und FORMAT.O	165/166
3.6.	Diskettenvergleichsprogramm	171
3.6.1.	DISKCOMPARER	171
3.7.	Bad-Block-Routine	177
3.7.1.	BAD.BLOCKS	177
3.8.	Blockeditor-Routine	179
3.8.1.	BLOCKEDIT	179
3.9.	DOS-ProDOS-Konvertierungsprogramm	183
3.9.1.	DOSTOPRO.A und DOSTOPRO.O	183
4.	**Tabellen**	196
4.1.	Speicherverteilung	196
4.2.	MLI-Befehle	197
4.4.	MLI-Fehlernummern	198
4.4.	BASIC.SYSTEM-Fehlernummern	199
4.5.	BASIC.SYSTEM-Befehle	200
4.6.	ASCII-Tabelle	202
4.7.	6502/65C02-Tabelle	203
4.8.	Applesoft-Tabelle	204
5.	**Begleitdiskette**	205
6.	**Register**	207

1. ProDOS für Applesoft-Programmierer

1.1. Die ersten Schritte in ProDOS

Falls Sie die ProDOS-Systemdiskette gerade erworben und/oder noch niemals unter DOS 3.3 oder ProDOS programmiert haben, so führen Sie folgendes Experiment durch, das Ihnen ein erstes „Feeling" für das neue Betriebssystem gibt. Sollten Sie kein Neuling mehr sein, so überspringen Sie dieses Teilkapitel.

Schritt 1: Schalten Sie den Apple aus, falls er bereits eingeschaltet war.

Schritt 2: Legen Sie die ProDOS-Systemdiskette in das Laufwerk 1. Je nachdem, ob Sie einen Apple IIe oder IIc besitzen, kann diese Systemdiskette unterschiedlich beschriftet sein. Beim Apple IIe heißt sie beispielsweise „ProDOS User's Disk".

Schritt 3: Schalten Sie nunmehr den Apple (und natürlich auch den Monitor) ein. Die Diskette im Laufwerk 1 beginnt zu rotieren, nachdem sie kurz „gerappelt" hat, was mit der Rekalibrierung des Lesekopfes zu tun hat. Es werden nunmehr verschiedene Systemdateien von der Diskette geladen, was etwa 10 Sekunden dauert.

Schritt 4: Am Schluß erscheint ein Menü am Bildschirm mit verschiedenen Optionen. Bei der Apple-IIe-ProDOS-Systemdiskette heißt es beispielsweise:
YOUR OPTIONS ARE:
(hier stehen diverse andere Optionen)
B - APPLESOFT BASIC

Wählen Sie aus dem Menü „Applesoft-Basic", indem Sie die gewünschte Ziffer oder den gewünschten Buchstaben, hier „B", tippen. Danach befinden Sie sich im Applesoft-Befehlsmodus unter dem ProDOS-Betriebssystem (= ProDOS-BASIC.SYSTEM). Das Applesoft-Basic und das ProDOS-BASIC.SYSTEM sind befehlswortorientiert, was besagt, daß Sie Befehle an Applesoft und das BASIC.SYSTEM erteilen können, indem Sie entweder das Befehlswort über die Tastatur eingeben (direkter Modus) oder über ein laufendes Applesoft-Programm an das Betriebssystem erteilen (indirekter Modus). Befehlswörter sollten normalerweise in Großbuchstaben eingegeben werden, wenngleich der Apple IIc unter ProDOS auch Kleinbuchstaben zuläßt. Rasten Sie deshalb die Shift-Taste vorsichtshalber stets ein.

Schritt 5: Tippen Sie auf die Return-Taste (entspricht der Wagenrücklauftaste bei der Schreibmaschine) und wiederholen Sie dies mehrmals. Sie werden sehen, daß mit jedem Return der Bildschirm um 1 Zeile nach oben scrollt (= rollt). Tippen Sie nunmehr das Wort NEW, gefolgt von der Return-Taste, also

NEW

Damit wird der Applesoft-Speicher gelöscht. Tippen Sie nun

PRINT 7 * 7

Danach müßte das Ergebnis dieser Multiplikation, also 49, angezeigt werden. Tippen Sie jetzt erneut

PRINT7*7

aber diesmal ohne Leertasten. Sie werden sehen, daß es genauso funktioniert wie oben. Tippen Sie jetzt absichtlich falsch

PRENT 7*7

und Sie werden die Fehlermeldung „SYNTAX ERROR" am Bildschirm sehen sowie einen Piepston hören. An Fehlermeldungen müssen Sie sich gewöhnen, denn jeder kann sich bei der Eingabe eines Befehlswortes vertippen.

In den nachfolgenden Kapiteln wird vorausgesetzt, daß Sie zumindest über Grundkenntnisse in Applesoft verfügen. Zur Wiederholung können Sie die Applesoft-Befehlsübersicht auf Seite 206 heranziehen.

Schritt 6: Die Befehle „NEW" und „PRINT 7 * 7" waren Applesoft-Befehle. Nunmehr wollen wir einige ProDOS-BASIC.SYSTEM-Befehle ausprobieren. Tippen Sie

CAT

Das Laufwerk läuft an und am Bildschirm erscheint das Inhaltsverzeichnis mit den Namen der Dateien, die sich auf der ProDOS-Diskette befinden, die im Laufwerk 1 liegt. Falls Sie über eine 80-Zeichenkarte verfügen, so schalten Sie diese jetzt mit dem Befehl

PR#3

ein. Tippen Sie jetzt, auch wenn Sie keine 80-Zeichenkarte haben,

CATALOG
anstelle von CAT, und Sie werden erneut das Diskетteninhaltsverzeichnis sehen, nunmehr jedoch mit mehr Informationen pro Zeile. Wir merken uns: „CAT" zeigt das Inhaltsverzeichnis in 40 Zeichen/Zeile (gleichviel ob die 80-Zeichenkarte eingeschaltet ist oder nicht) und „CATALOG" in 80 Zeichen/Zeile an. Wechseln Sie zwischen „CAT" und „CATALOG" ab, damit Sie ein Gefühl für diesen wichtigen Befehl bekommen.

Schritt 7: Das Inhaltsverzeichnis umfaßt die Namen verschiedener Dateien. Sie müssen unterscheiden zwischen dem Namen einer Datei und der Datei selbst. Desgleichen müssen Sie differenzieren zwischen dem Namen einer Diskette und der Diskette selbst. Der Name der ProDOS-Systemdiskette für den IIe lautet „USERS.DISK". Tippen Sie jetzt
CAT
und betrachten Sie den danach links oben am Bildschirm zuerst angezeigten Namen, der mit einem *Schrägstrich* beginnt, wahrscheinlich „/USERS.DISK" o.ä., je nachdem, welche Systemdiskette Sie benutzen. Setzen Sie anstelle von „USERS.DISK" nachfolgend denjenigen Namen ein, den Sie oben links am Bildschirm nach CAT gesehen haben. Wenn Sie nun
CAT/USERS.DISK
eingeben, wird – wie bei CAT allein – das Inhaltsverzeichnis angezeigt. Würden Sie
CAT/DISKETTE
eingeben, so würde das BASIC.SYSTEM eine eingelegte Diskette mit diesem Namen „DISKETTE" suchen, und zwar in allen angeschlossenen Laufwerken, d.h. auch in Laufwerk 2 usw. Probieren Sie es aus und stören Sie sich nicht an der Fehlermeldung, die danach angezeigt wird, weil die Diskette namens „DISKETTE" gar nicht eingelegt ist.
Auf der Systemdiskette befand sich mit Gewißheit eine Datei namens „STARTUP". Dabei handelt es sich um dasjenige Applesoft-Programm, das (ähnlich wie das Hello-Programm bei DOS 3.3) automatisch nach dem Einlesen des ProDOS-Betriebssystems gestartet wurde. Tippen Sie nunmehr
VERIFY STARTUP
und danach
VERIFY/USERS.DISK
Mit diesem Befehl, d.h. „VERIFY" plus Schrägstrich plus Dateiname bzw. Diskettenname können Sie feststellen, ob die gleichnamige Datei bzw. Diskette existiert. Wenn ja, erscheint keine Fehlermeldung, wenn nein, lautet die Meldung „PATH NOT FOUND" usw. Wir merken uns: Unter ProDOS haben sowohl Disketten als auch die darauf enthaltenen Dateien einen Namen. Vor Diskettennamen *muß* man den Schrägstrich setzen, vor Dateinamen nur in bestimmten Fällen (s.u.).

Schritt 8: Geben Sie nunmehr ein
LOAD/USERS.DISK/STARTUP
Mit diesem Befehl wird die *Datei* namens „STARTUP" von der *Diskette* namens „USERS.DISK" eingeladen. Nach dem Applesoft-Befehl
LIST
können Sie das Programm listen und sich somit überzeugen, daß die Programmdatei „STARTUP" tatsächlich eingelesen wurde. Wiederholen Sie den Vorgang durch die vier Befehle
NEW
LIST
LOAD/USERS.DISK/STARTUP
LIST
Nach NEW war das Programm gelöscht und nach erneutem LOAD wieder im Speicher. Probieren Sie nun noch etwas die gelernten Befehle CAT, CATALOG, VERIFY und LOAD aus. Wir merken uns: Bei „LOAD/USERS.DISK/STARTUP" folgen dem Befehlswort „LOAD" zwei Namen – durch Schrägstriche abgetrennt. Der erste Name ist der Diskettenname oder das *Präfix*, der letzte der eigentliche Dateiname oder *Filename*. Die Kombination aus Präfix und Filename heißt Pfadname oder *Pathname*.
Da die Systemdiskette zugeklebt ist, können Sie darauf keine neuen Dateien speichern. Falls Sie zu diesem Buch die Begleitdiskette erworben haben, so fertigen Sie von der Systemdiskette zunächst eine Arbeitskopie an. Legen Sie zu diesem Zweck die Begleitdiskette zu „ProDOS für Aufsteiger, Band 2" in Laufwerk 1 und geben Sie folgende Befehle ein:
PREFIX/STIEHL
RUN PRODOS.COPY.A
Danach entfernen Sie die Begleitdiskette und legen Sie erstens die Systemdiskete (ORIGINAL) und zweitens eine fabrikneue Leerdiskette (DUPLIKAT) bereit. Tippen Sie „J" für „START J/N" und legen Sie jeweils ORIGINAL bzw. DUPLIKAT gemäß den Bildschirmanweisungen abwechselnd in das Laufwerk 1. Nach einer guten Minute haben sie eine Sicherungskopie der Systemdiskette.
Verfahren Sie analog mit der Begleitdiskette, deren Schreibschutzkerbe Sie jedoch, falls nicht bereits geschehen, zuvor sicherheitshalber zukleben sollten.

1.2. DOS 3.3 und ProDOS

Die Vorteile von ProDOS gegenüber DOS 3.3 lassen sich wie folgt charakterisieren:

Speicherkapazität: Unter ProDOS kann ein einziger externer Datenträger bis zu 32M (= 32 Megabytes = ca. 32 Millionen Zeichen) umfassen, und eine einzige Datei kann bis zu 16M groß sein. Demgegenüber war DOS 3.3 von vornherein nur für kleinere Datenträger konzipiert gewesen und konnte ohne Modifikationen („Patches") lediglich Disketten mit 140K (= 140 Kilobytes = ca. 140.000 Zeichen) verwalten.

Geschwindigkeit: Die Datenübertragungsrate auf unterster Maschinenebene ist bei ProDOS – ähnlich wie bei dem ebenfalls blockstrukturierten Pascal – mit über 9000 Zeichen/Sekunde erheblich größer als beim alten DOS 3.3 mit über 7000 Zeichen/Sekunde. Diese Werte gelten für Disk-II-Laufwerke.

Leistungsfähigkeit: Gegenüber DOS 3.3 sind die Befehle und Dateitypen bei ProDOS ganz beachtlich erweitert worden. Zum einen ist für Assembler-Programmierer eine sehr leistungsfähige Schnittstelle (MLI = Machine Language Interface) geschaffen worden, die kaum Wünsche offenläßt. Zum anderen bieten sich dem Applesoft-Programmierer zahlreiche neue Möglichkeiten komfortablerer Programmierung. Stellvertretend sei hier ein Beispiel für DOS-Programmierer genannt: Man kann jetzt auch Textfiles so einladen, als wären es Binärfiles, und man kann darüber hinaus auch beliebige Teile von Binärfiles einlesen, so daß sich speziell für Random-Dateien völlig neue Programmiertechniken eröffnen. Diese erweiterte Leistungsfähigkeit ist in Verbindung mit der größeren Geschwindigkeit der Datenübertragung der entscheidene Vorteil von ProDOS gegenüber dem alten DOS 3.3.

Sprachkompatibilität: Das Betriebssystem ProDOS besteht aus zwei Teilen: PRODOS und BASIC.SYSTEM. Der Teil PRODOS – man beachte die Großschreibung gegenüber ProDOS – liegt in den oberen 16K des Apple-Arbeitsspeichers, d.h. in der sog. Language Card (= Sprachkarte), und dient dem eigentlichen Zugriff auf externe Datenträger (Lesen/Schreiben). Der Teil BASIC.SYSTEM liegt in demjenigen Bereich der unteren 48K des Apple-Arbeitsspeichers, wo sich früher das *gesamte* DOS 3.3 befand, und dient der Interpretierung und Abarbeitung der unter Applesoft-BASIC erteilten Befehle wie CATALOG, LOAD, SAVE usw. ProDOS ist damit etwa doppelt so groß wie das alte DOS 3.3. Da durch PRODOS die eigentlich zum Einladen anderer Programmiersprachen gedachte Sprachkarte bereits belegt ist, läuft das BASIC.SYSTEM nur unter Applesoft- und nicht mehr unter Integer-BASIC. Dies ist kein ernsthafter Nachteil, da Integer-BASIC ohnehin nur für Spielprogramme u.ä. verwendet wurde.

1.3. Monitor, Applesoft und ProDOS

Beim Apple lassen sich drei Arten von Systemprogrammen unterscheiden:
1. Monitor (ROM $F800-$FFFF)
2. Applesoft (ROM $D000-$F7FF)
3. ProDOS (RAM)

Der **Monitor** ist eine Sammlung von Maschinenroutinen, die u.a. die Bildschirmdarstellung und die Tastaturabfrage übernehmen. Der Monitor befindet sich im ROM (= Read Only Memory = Nur-Lese-Speicher = Festwertspeicher) und ist damit nach dem Einschalten des Apple sofort aktiv. Gäbe es kein Monitorprogramm, dann wäre der Apple nach dem Einschalten „tot", d.h. die Tastatur würde nicht reagieren und am Bildschirm würde man nichts oder nur einen „Buchstabensalat" sehen.

Das **Applesoft** ist eine Sammlung von Maschinenroutinen, das der Interpretierung (= Deutung) und Ausführung von Befehlen der Applesoft-Programmiersprache dient. Applesoft befindet sich ebenfalls im ROM und ist deshalb nach dem Einschalten des Apple wie der Monitor sofort aktiv.

Das **ProDOS** befindet sich nicht im ROM, sondern muß erst von der Diskette in das RAM (= Random Access Memory = Lese-Schreib-Speicher) eingelesen werden. Dieses Einlesen des Betriebssystems wird als **Booten** bezeichnet. Das Bootprogramm oder der sog. Urlader besteht aus mehreren Stufen oder Teilprogrammen.

1.4. Bootvorgang

Nach dem Einschalten des Apple wird der *Urlader Stufe 1* im Monitor (sog. Power-Up-Routine) aktiviert, dessen Zweck darin besteht, herauszufinden, an welchem der bis zu 7 Slots oder Steckplätze eine Controller-Karte für ein Diskettenlaufwerk angeschlosssen ist. Wird ein Controller entdeckt, dann wird das Programm auf der Controller-Karte aktiviert. Dies ist der *Urlader Stufe 2*, welcher zunächst den Sektor 0 von Spur 0 der ProDOS-Diskette einliest. Dieses Teilprogramm lädt dann Block 0 (und im Falle eines Apple III auch Block 1) von der ProDOS-Diskette, welcher zusammen mit einem Teil des Programms auf der Controller-Karte den *Urlader Stufe 3* darstellen, der seinerseits das Betriebssystem namens PRODOS nunmehr vollständig in das RAM überträgt. Nachdem das Betriebssystem PRODOS in der Language Card (= Sprachkarte) installiert worden ist, wird noch das sog. BASIC.SYSTEM eingeladen, das dann seinerseits das Hello-Programm namens STARTUP einliest, womit der Bootvorgang abgeschlossen wird.

Hinweis: Unter Hello-Programm versteht man ein Programm, das im Anschluß an die Installierung des Betriebssystems automatisch gestartet wird. Bei ProDOS muß das Hello-Programm STARTUP heißen, während bei DOS 3.3 ein beliebiger Dateiname gewählt werden konnte. „Booten" kommt von englisch „to bootstrap" und bedeutet „an der Stiefelschlaufe ziehen" oder im übertragenen Sinne „sich wie Münchhausen am eigenen Schopf herausziehen".

Um ProDOS zu starten, gibt es eine Reihe von Möglichkeiten, wobei jeweils vorausgesetzt wird, daß sich eine bootfähige ProDOS-Diskette in einem Laufwerk befindet. Für (1) und (2) gilt, daß sich die ProDOS-Diskette in Laufwerk 1 befinden muß. Demgegenüber kann sich bei (3) die ProDOS-Diskette auch in Laufwerk 2 befinden oder der externe Datenträger kann eine RAM-Disk oder ein Festplattenlaufwerk sein. Beim Apple IIc kann zudem ProDOS von dem externen Laufwerk mit PR#7 gestartet werden.

(1) Wenn der Apple noch nicht eingeschaltet ist (echter Kaltstart):
Einschalten des Stromes
bewirkt Kaltstart über Urlader Stufe 1; funktioniert immer.

(2) Wenn der Apple bereits eingeschaltet ist („lauwarmer" Kaltstart):
Ctrl-Offener Apfel-Reset
beim Apple IIc oder IIe. Dies entspricht dem Kaltstart über Urlader Stufe 1; funktioniert immer. Oder
CALL 64166
Dieser Applesoft-Befehl ruft den Urlader Stufe 1 im Monitor auf; funktioniert nur, wenn Applesoft aktiv ist. CALL 64166 ermittelt automatisch den Controller, der sich im Slot mit der höchten Nummer – meist Slot 6 – befindet. Oder
PR#6
oder
CALL 50688
Die Applesoft-Befehle PR#6 oder CALL 50688 rufen unter Umgehung des Urladers Stufe 1 direkt den Urlader Stufe 2 im Controller in Slot 6 auf. Wenn man sich gerade im Monitor-Modus befindet – erkennbar an dem *-Prompt –, so kann man statt PR#6 auch
6 Ctrl-P
sagen; funktioniert nur, wenn Slot 6 mit einem Controller belegt ist.

(3) Wenn sich ProDOS noch im Speicher befindet („warmer" Kaltstart):
-PRODOS
Dieser sog. Strichbefehl in Verbindung mit dem Dateinamen PRODOS startet ProDOS neu; funktioniert nur, wenn sich sowohl das PRODOS als auch das BASIC.SYSTEM noch intakt im Speicher befinden. Zweck: Booten von einem nicht über Kaltstart bootfähigen Datenspeicher wie Profile, 3,5-Zoll-Laufwerk usw. Oder

BYE

In der Language Card wird von ProDOS ein sog. Reboot-Programm installiert, das von Applesoft ab ProDOS 1.1.1 über den Befehl BYE bzw. bei älteren ProDOS-Versionen über ein kleines Maschinenprogramm aufgerufen werden kann:

POKE 768, 32: POKE 769, 0: POKE 770, 191: POKE 771, 101: POKE 772, 6: POKE 773, 3: POKE 774,4: CALL 768

Nach BYE bzw. nach den Pokes und nach CALL 768 erscheint zunächst die Meldung
ENTER PREFIX (PRESS „RETURN" TO ACCEPT)
worauf man das gewünschte Präfix eingeben muß. Danach erscheint als zweite Meldung
ENTER PATHNAME OF NEXT APPLICATION
worauf man den gewünschten Pfadnamen eingeben muß.
Das Reboot-Programm setzt voraus, daß sich PRODOS noch intakt in der Language Card befindet, während das BASIC.SYSTEM nicht benötigt wird.

Hinweis: Vom echten Kaltstart spricht man, wenn ein Systemprogramm und Systemwerte völlig neu initialisiert werden. Bei geschützten Anwenderprogrammen oder beim Wechsel von einem Prozessor zum anderen (z.B. vom Z80 zum 6502) geht dies oft nur über Ausschalten-Einschalten. In allen anderen Fällen sind unterschiedlich „warme" Kaltstartformen möglich. Vom echten Warmstart spricht man dann, wenn man zwangsweise, jedoch in der Regel ohne Zerstörung von bereits initialisierten Werten, in einen übergeordneten Befehlsmodus gelangt, z.B. wenn man ein Applesoft-Programm mit Ctrl-Reset oder Ctrl-C unterbricht.

1.5. Externe Speicherorganisation

Unter ProDOS können als externe Datenspeicher verwendet werden:
a) Disk-II-Laufwerke (140 Kilobytes)
b) 64K-Karte als RAM-Disk beim IIc/IIe
c) Profile-Festplattenlaufwerk (5 Megabytes)
d) Größere Laufwerke (3,5 Zoll Sony, 5,25 Zoll Teac usw. mit mehr als 300K)
e) Größere RAM-Karten als RAM-Disks (mit mehr als 128K)
f) Größere Festplattenlaufwerke (Corvus usw. mit mehr als 20M)
Für die kleineren externen Datenspeicher sind von der Firma Apple eigene Driver-Programme entwickelt worden (Disk-Driver = Treiber- oder Steuerprogramm für Controller-Karten). Für die leistungsfähigeren und größeren Datenspeicher sind Driver-Programme von Fremdfirmen lieferbar. Für manche Laufwerke wie etwa die Sony-Drives sind nicht einmal gesonderte Driver erforderlich. Hier genügt ein entsprechendes Formatierungsprogamm, wie es in diesem Buch beschrieben wird.

Nachfolgend beschränken wir uns zunächst auf die normale Apple-Disketten für Disk-II-Laufwerke. Für diese 5,25-Zoll-Disketten gilt:

- Jede Diskette hat 35 konzentrische Spuren, numeriert von 0-34.
- Jede Spur ist in 8 Blocks unterteilt.
- Jeder Block umfaßt 512 Bytes oder Zeichen

Eine Diskette enthält damit 35 Spuren mal 8 Blocks = insgesamt 280 Blocks, die von 0-279 durchnumeriert sind und von denen die ersten 7 Blocks auf Spur 0 einen festen Inhalt haben:

Blocks 0 bis 1 enthalten den Urlader Stufe 3. Block 1 wird bei einem Apple II ignoriert. Dieser Block war für den Apple III gedacht.
Blocks 2 bis 5 enthalten das sog. Volume Directory oder Hauptinhaltsverzeichnis.
Block 6 enthält die sog. Volume Bit Map, in der über die belegten Blocks „Buch geführt" wird.

Die restlichen 273 Blocks sind theoretisch für Programme und Daten frei. Eine unter Applesoft bootfähige Diskette muß indessen neben den 7 Blocks für Urlader usw. noch mindestens 3 weitere Dateien enthalten, nämlich

a) Datei PRODOS (31 Blocks bei PRODOS 1.0.2, 30 Blocks bei PRODOS 1.1.1), die vom Urlader Stufe 2 geladen wird.
b) Datei BASIC.SYSTEM (21 Blocks), die vom PRODOS geladen wird.
c) Datei STARTUP (mindestens 2 Blocks), die vom BASIC.SYSTEM geladen wird.

Damit gilt folgende Raumaufteilung:
7 Blocks für Urlader usw.
31 Blocks für PRODOS
21 Blocks für BASIC.SYSTEM
2 Blocks für STARTUP
Dies sind zusammen 61 Blocks. Damit verbleiben nur noch 280 minus 61 = 219 Blocks für sonstige Programme und Daten. Dies sind 219 mal 512 = 112.128 Bytes = 109.5K Nettospeicherkapazität, also nicht allzu viel für ein Betriebssystem, das 32M brutto verwalten könnte.

Hinweis: 1 Byte = 1 Zeichen (z.B. Buchstabe, Ziffer, Code). 1K = 1 Kilobyte = 1024 Bytes. 1M = 1 Megabyte = 1024K. Merke ferner als applespezifisch: 1 Block = 512 Bytes. 1 Sektor = 256 Bytes.

1.6. Interne Speicherorganisation

Applesoft-Programmierer müssen zumindest einen groben Überblick über die interne Speicherorganisation von ProDOS haben. Bekanntlich kann der Mikroprozessor des Apple II 64K oder exakt 65536 Speicherstellen adressieren bzw. verwalten. Diese Speicherstellen sind von 0-65535 durchnumeriert. Unter ProDOS und Applesoft gilt dann folgende Einteilung (Adressen in hexadezimal und dezimal):

$0000-$00FF (0-255): Nullseite mit wichtigen Systemwerten; normalerweise für Applesoft tabu.
$0100-$01FF (256-511): Prozessor-Stapelspeicher; normalerweise für Applesoft tabu.
$0200-$02FF (512-767): Tastatur-Eingabepuffer; wird außerdem vom BASIC.SYSTEM während des Diskettenzugriffs zur Zwischenspeicherung des Dateinamens usw. benutzt. Deshalb darf insbesondere die zweite Hälfte ab $0280 nicht mehr – wie man es gelegentlich unter DOS 3.3 tat – für eigene Programmzwecke benutzt werden. Zudem darf man kein Maschinenprogramm in den Eingabepuffer einlesen.
$0300-$03CF (768-975): Frei für kurze Maschinenprogramme.
$03D0-$03FF (976-1023): Vektoren für ProDOS-Warmstart, Reset, Interrupt usw. (gegenüber DOS 3.3 nur noch teilweise benutzt).
$0400-$07FF (1024-2047): 40-Z/Z-Bildschirmspeicher.
$0800-$95FF (2048-38399): Frei für Applesoft-Programm, Hires-Grafik usw.
$9600-$99FF (38400-39423): HIMEM (Highest Memory = Obergrenze des freien Speichers) und zugleich Beginn des ersten BASIC.SYSTEM-Puffers für Input/Output. Für jede geöffnete Textdatei verschiebt sich HIMEM vorübergehend um je 1024 Bytes pro Puffer nach unten.
$9A00-$BCFF (39424-48639): BASIC.SYSTEM mit internen Puffern.
$BE00-$BEFF (48640-48895): Sog. BASIC.SYSTEM Global Page mit Systemwerten.
$BF00-$BFFF (48896-49151): Sog. PRODOS Global Page mit Systemwerten.
$C000-$CFFF (49152-53247): ROM-Bereich für die Slots 0-7 beim Apple II Plus bzw. internes ROM beim Apple IIc. Der Apple IIe hat demgegenüber sowohl internes ROM wie auch Slot-ROM (je nach Interface-Karten).
$D000-$FFFF (53248-65535): ROM-Bereich des Applesoft-Interpreters und des Monitors.
$D000-$FFFF (53248-65535): RAM-Bereich der Sprachkarte Bank 1 mit dem eigentlichen PRODOS.
$D000-$DFFF (53248-57343): RAM-Bereich der Sprachkarte Bank 2 mit Reboot-Programm (BYE-Befehl).

Damit steht dem Applesoft-Programmierer wie unter DOS 3.3 erstens der Bereich $0800-$95FF = 2048-38399 für das eigentliche Applesoft-Programm sowie der Bereich

$0300-$03CF = 768-975 für kurze Maschinenprogramme zur Verfügung. Der unter DOS 3.3 früher unbenutzte 16K-Bereich der Sprachkarte ist nunmehr durch PRODOS sowie das Reboot-Programm belegt. Damals konnte man z.B. das DOS 3.3 komplett in die Sprachkarte schieben und hatte dann die unteren 48K fast völlig frei zur Verfügung, so daß sehr große Applesoft-Programme erstellt werden konnten. Diese Möglichkeit ist unter ProDOS entfallen.

Unter DOS 3.3 gab es den bei ProDOS nicht mehr existierenden Befehl MAXFILES, mit dessen Hilfe man eine gewünschte Anzahl von Puffern reservieren konnte. Darüber hinaus konnte man HIMEM herabsetzen und größere Maschinenprogramme zwischen HIMEM und dem Beginn des untersten DOS-Puffers einladen. Unter ProDOS muß anders verfahren werden, wie später aufgezeigt wird.

1.7. Befehlstypen

Befehle an das BASIC.SYSTEM bestehen aus Befehlsnamen bzw. Befehlswörtern, die wie unter DOS 3.3 in zwei Formen erteilt werden können:

Direkte Befehle (Nicht-RUN-Modus):
Diese können über die Tastatur eingegeben werden. Vor dem Befehl muß implizit und nach dem Befehl explizit ein Return (Rtn, Ctrl-M) stehen, z.B.
(Rtn implizit)
CATALOG Rtn
Das Return vor dem Befehl gilt immer dann als implizit vorhanden, wenn der Cursor nach dem Prompt-Zeichen „Ü" am linken Bildschirmrand blinkt. Führende Leertasten wären erlaubt, z.B.
 CATALOG
Dagegen sind andere Zeichen nicht zulässig, z.B.
:CATALOG
Das Befehlswort selbst könnte Leertasten enthalten, nicht jedoch der Pfadname, z.B.
C A T A L O G/DIRECTORY
nicht jedoch
CATALOG/D I R E C T O R Y
Ferner sind Leertasten *zwischen* den einzelnen Parametern erlaubt, also
CAT /VOLDIR/SUBDIR, S6, D2
statt
CAT/VOLDIR/SUBDIR,S6,D1
nicht jedoch

CAT /VOLDIR /SUBDIR
weil „/VOLDIR/SUBDIR" eine Einheit bildet.
Mehrere BASIC.SYSTEM-Befehle innerhalb *derselben* Eingabezeile sind nicht möglich, also nicht
CATALOG: CATALOG
Schließlich sei erwähnt, daß Befehlswörter usw. auch in Kleinbuchstaben eingegeben werden können, also
catalog
statt
CATALOG
In Programmen sollte man jedoch Kleinbuchstaben in Befehlen und Pfadnamen vermeiden, zumal das MLI nur Großbuchstaben akzeptiert.

Indirekte Befehle (RUN-Modus):
Diese werden während des laufenden, nicht-compilierten Applesoft-Programms an das BASIC.SYSTEM erteilt. Man beachte also, daß im Gegensatz zu DOS 3.3 kein Integer-Basic-, kein compiliertes Applesoft- und auch kein Assembler-Programm als RUN-Modus gilt. Dem Befehl, der im RUN-Modus in Anführungszeichen gesetzt wird, geht (normalerweise) ein implizites Return sowie auf alle Fälle stets ein Ctrl-D = CHR$ (4) voraus. Ferner muß der Befehl durch ein explizites Return abgeschlossen werden, z.B.
10 PRINT CHR$ (4) "CATALOG"
Das Ctrl-D muß das erste Zeichen einer PRINT-Anweisung sein, die ihrerseits entweder unmittelbar der Zeilennummer oder unmittelbar dem Doppelpunkt (Statement-Separator) folgen muß. Mehrfach-Befehle in derselben Programmzeile sind zulässig, z.B.
10 PRINT CHR$ (4) "CAT": PRINT CHR$ (4) "CAT"
Der Befehl in Anführungszeichen kann durch eine String-Variable ersetzt werden, z.B.
10 X$ = CHR$ (4) + "CAT"
20 PRINT X$
Im Gegensatz zu DOS 3.3 befindet sich das BASIC.SYSTEM stets im sog. nicht-sichtbaren Pseudo-Trace-Modus, der nach jeder neuen Zeilennummer sowie nach jedem Doppelpunkt wirksam wird. Deshalb sind einige syntaktische Feinheiten gegenüber DOS 3.3 zu beachten:

Beispiel 1:
10 X$ = CHR$ (4) + "CATALOG"
20 PRINT "AAA";: PRINT X$
Hier wird der CATALOG-Befehl ausgeführt, obwohl nach „AAA" ein Semikolon folgt und somit kein implizites Return vorangeht. Grund: PRINT X$ steht nach einem Doppelpunkt. Bei DOS 3.3 würde demgegenüber nur „AAACATALOG" am Bildschirm angezeigt und somit der CATALOG-Befehl ignoriert.

Beispiel 2:

```
10 X$ = CHR$ (4) + "CATALOG"
20 Y$ = ",D1"
30 PRINT X$; Y$: REM richtig
40 PRINT X$;: PRINT Y$: REM falsch
```

Hier sind Befehlsname und Parameter getrennte Strings. Werden diese wie in Zeile 30 als ein einziger PRINT-Befehl ausgeführt, dann ist das BASIC.SYSTEM „zufrieden". Erfolgt indessen eine Aufsplittung wie in Zeile 40, so führt dies zu einem Syntax-Error. Unter DOS 3.3 wäre demgegenüber eine Aufteilung zulässig.

Beispiel 3:

```
10 R$ = CHR$ (13): REM Ctrl-M
20 C$ = CHR$ (4) + "CATALOG"
30 PRINT R$: PRINT C$
40 PRINT R$; C$
```

Hier wird zusätzlich ein nacktes Return als String-Variable definiert. In Zeile 30 wäre das BASIC.SYSTEM „zufrieden" (im Gegensatz zu der falschen Angabe in dem englischen ProDOS-Handbuch der Firma Apple). In Zeile 40 hingegen würde eine Fehlermeldung entstehen, weil nunmehr Ctrl-D nicht mehr das *erste* Zeichen der Zeichenkette wäre. Unter DOS 3.3 würde Zeile 40 fehlerfrei ausgeführt.

Beipiel 4:

```
10 GET X$: PRINT CHR$ (4) "CAT"
```

Wenn zwischen GET und dem CATALOG-Befehl lediglich ein Doppelpunkt steht, ist das BASIC.SYSTEM „zufrieden". Dies war unter DOS 3.3 nicht der Fall, so daß auf GET ein Return folgen mußte.

Als Fazit halten wir fest, daß ein indirekter BASIC.SYSTEM-Befehl nur dann korrekt ausgeführt wird, wenn das Ctrl-D das *erste* Zeichen der PRINT-Anweisung ist und wenn diese bis zum nächsten Statement-Separator („:" oder $00 als End of Line) *komplett* gePRINTet wird und nicht durch ein Semikolon, das das Return unterdrücken würde, abgeschlossen wird. Ferner sei darauf hingewiesen, daß das Ctrl-D wahlweise als CHR$ (4) mit Bit 7 off oder als CHR$ (132) mit Bit 7 on (4 + 128 = 132) definiert werden könnte.

Die Textdatei-Befehle OPEN, READ, WRITE, POSITION und APPEND sind übrigens reine Indirekt-Befehle, während alle anderen Befehle sowohl im RUN- wie auch im Nicht-RUN-Modus an das BASIC.SYSTEM erteilt werden können.

Im BASIC.SYSTEM sind folgende Befehle implementiert worden:

Allgemeine Befehle
CAT: Anzeige des Directory in 40 Z/Z
CATALOG: Anzeige des Directory in 80 Z/Z
PREFIX: Definition des Default-Directory
CREATE: Anlage einer neuen Datei

VERIFY: Existenz einer Datei prüfen
LOCK: Dateiname mit Sternchen versehen
UNLOCK: Sternchen entfernen
RENAME: Umbenennung einer Datei
DELETE: Datei löschen

Applesoft-Befehle
RUN: Starten eines Applesoft-Programms
LOAD: Laden eines Applesoft-Programms
SAVE: Speichern eines Applesoft-Programms

CHAIN: Starten eines Applesoft-Moduls
STORE: Speichern aller Applesoft-Variablen
RESTORE: Laden aller Applesoft-Variablen
FRE: String-Garbage-Collection

Binärdatei-Befehle
BRUN: Maschinenprogramm starten
BLOAD: (Binär)datei laden
BSAVE: (Binär)datei speichern

Start- und Exit-Befehle
–: Ersatz für RUN, BRUN, EXEC
BYE: Reboot-Befehl (nur ab Version 1.1.1)
EXEC: Textbefehlsdatei starten

I/O-Befehle
PR#: Änderung der Ausgabe-Vektoren
IN#: Änderung der Eingabe-Vektoren

Textdatei-Befehle
OPEN: Öffnen einer Textdatei
READ: Lesen von einer Textdatei
WRITE: Schreiben auf eine Textdatei
CLOSE: Schließen einer Textdatei
FLUSH: Puffer auf Textdatei übertragen

APPEND: Textdatei erweitern
POSITION: Textdatei-Feldzeiger ändern

Diejenigen, die unter DOS 3.3 programmiert haben, werden bemerken, daß die Befehle FP, INT, MON, NOMON, MAXFILES und INIT fehlen. FP und INT mußten entfallen, weil das BASIC.SYSTEM nur für Applesoft und nicht für Integer-Basic gedacht ist. MON und NOMON fehlen wahrscheinlich aus Platzgründen. (Da NOMON merkwürdigerweise bis einschließlich Version 1.1.1 in der Befehlstabelle enthalten ist, kann man diesen Befehl ausgeben. Er tut jedoch nichts und führt auch zu keiner Fehlermeldung.) Dasselbe gilt für den INIT-Befehl. MAXFILES ist nicht mehr möglich, weil mit dynamischen I/O-Puffern gearbeitet wird.
Dafür gibt es einige völlig neue Befehle, nämlich CAT, CREATE, FLUSH, PREFIX, STORE, RESTORE und FRE. Darüber hinaus wurden die meisten gleichnamigen DOS-Befehle unter ProDOS funktionell erweitert und verbessert.

1.8. Parametertypen

Die Parameter sind Zusätze zu den Befehlsnamen. Sie bestehen aus den Parameter-Buchstaben und dem Parameter-Wert und werden durch Kommas getrennt, wobei bei mehreren Befehlsparametern deren Reihenfolge beliebig ist, z.B. CATALOG, S6, D2 oder CATALOG, D2, S6.
Es gibt zwei Ausnahmen:
a) Namen werden durch Schrägstriche getrennt, z.B. CAT/DISK/DEMOS.
b) Der Parameter-Wert für die Slot-Nummer wird bei den PR#/IN#-Befehlen direkt an das Nummernkreuz angehängt, z.B. PR#1

/n: Name (max. 15 Zeichen), kompletter Pfadname (max. 64 Zeichen)
s: Slot bei PR#s, IN#s (0-7)
,Ss: Slot (1-7); Slot = Steckplatz (für Controller)
,Dd: Drive (1-2); Drive = Laufwerk
,Aa: Anfangsadresse ($0200-$BFFF)

,Ee: Endadresse ($0200-$BFFF)
,Ll: Länge ($0001-$C000)
,Bb: Byte-Offset ($000000-$FFFFFF = 0-16777215)
,Rr: Recordnummer (0-65535)
,Ff: Feldnummer (0-65535)
,§z: Zeilennummer (0-63999)
,Tt: Dateityp (3-Buchstaben-Kürzel)
,Vv: Volume-Nummer (1-254); wird ignoriert!

Beispiele

```
CATALOG/VOLUME/BRIEFE
PR#3
CAT, S3, D2
BSAVE BILDER, A$2000, E$5FFF
BSAVE BILDER, A$2000, L$4000
BLOAD BILDER, A$4000, B$2000
PRINT CHR$ (4) "READ FILE, R1000"
PRINT CHR$ (4) "READ FILE, F3"
LOAD PROGRAMM, §2000
BLOAD /VOLUME, A$1000, TDIR
```

Parameter können durch Variablen ersetzen werden. Beispiele:

```
10 S = 6: D = 1
20 D$ = CHR$ (4)
30 C$ = "CATALOG"
40 PRINT D$; C$; ",S";S; ",D";D
50 PRINT D$ C$ ",S"S ",D"D
```

Die Zeilen 40 und 50 unterscheiden sich darin, daß in Zeile 40 aus optischen Gründen zwischen den Variablen Semikola eingefügt wurden, die jedoch syntaktisch entbehrlich sind.

1.9. Dateitypen

Wie bereits erwähnt wurde, kann ein ProDOS-Datenträger (Diskette, Festplatte) im Rahmen der maximalen Speicherkapazität von 32M beliebig viele Dateien umfassen, wobei eine einzelne Datei bis zu 16M groß sein kann. Die ProDOS-Dateitypen lassen sich in drei Gruppen einteilen:

SOS-Dateitypen: Dies waren die Dateitypen des „Sophisticated Operating System" des Apple III, die inzwischen keine Rolle mehr spielen, da der Apple III nicht mehr produziert wird. Einige SOS-Dateitypen wurden inzwischen umfunktioniert, z.B. „AWP" für Appleworks-Word-Processing-File.
Selbstdefinierte Dateitypen: Ein Anwenderprogramm könnte sich seine eigenen Dateitypen mit den Typ-Nummern im Bereich $F1 bis $F8 definieren. Ein Bedarf hierfür besteht jedoch normalerweise nicht.
Standard-Dateitypen: Dies sind die normalen oder allgemeinen Dateitypen, und zwar u.a.:

SYS: Systemprogramm
TXT: Textdatei
BIN: Binärdatei
BAS: Applesoft-Programm
VAR: Applesoft-Variablendatei
DIR: Subdirectory

Der Dateityp besteht aus dem Parameter „T" und dem jeweiligen 3-Buchstaben-Kürzel, z.B. TSYS, TTXT, TBIN, TBAS, TVAR, TDIR usw. Andere Dateitypen wie „REL" für relokatives Maschinenprogramm oder „BAD" für defekte Dateien usw. werden zwar im Directory ausgewiesen, jedoch (noch) nicht durch besondere Befehle vom BASIC.SYSTEM unterstützt.

SYS-Dateien umfassen neben PRODOS und BASIC.SYSTEM weitere mögliche Systemprogramme, z.B. FILER, CONVERT u.a. SYS-Dateien werden mit
BLOAD SYSTEMDATEI, TSYS, A$2000
geladen und mit
-SYSTEMDATEI oder
BRUN SYSTEMDATEI, A$2000, TSYS
gestartet. Die Startadresse liegt praktisch immer bei $2000, obgleich dies aus dem Directory nicht ersichtlich ist (meist im Zusatzinfo-Feld enthalten, s.u.). Systemdateien können wie folgt kopiert werden:
BLOAD /DIR1/SYSTEMDATEI, TSYS, A$2000
CREATE /DIR2/SYSTEMDATEI, TSYS
BSAVE /DIR2/SYSTEMDATEI, TSYS, A$2000, L?????
Die jeweilige Länge kann man dem Directory entnehmen.
SYS-Dateien lassen sich definieren als Maschinenprogramme, die einen Teil des ProDOS-Betriebssystems beinhalten und ab $2000 gestartet werden.

TXT-Dateien sind normale Text- bzw. ASCII-Dateien. Falls diese mit den BASIC.SYSTEM-Befehlen OPEN, WRITE, READ, APPEND, POSITION, FLUSH und CLOSE bearbeitet werden, ist bei jedem ASCII-Zeichen das Bit 7 auf 0 gesetzt, also z.B. $0D statt $8D für Return. Unter DOS 3.3 war es genau umgekehrt. Man beachte ferner, daß das Mischen von Bit-7-off- und Bit-7-on-ASCII-Zeichen auf einer normalen TXT-Datei nicht ohne weiteres möglich ist.
Programmtechnisch können TXT-Dateien sequentielle oder Random-Dateien sein. Sequentielle TXT-Dateien sind aus dem Directory durch „R = 0" ersichtlich, während Random-TXT-Dateien eine Recordlänge R > 0 haben. Sequentielle TXT-Dateien haben normalerweise nicht wie bei DOS 3.3 einen Endmarker $00 = Ctrl-@. Vielmehr ergibt sich das Dateiende aus dem ENDFILE = End of File = EOF, das aus dem Directory ersichtlich ist.

BIN-Dateien sind Binärdateien aller Art, z.B. Maschinenprogramme, Hires-Bilder usw. Im Gegensatz zu DOS 3.3 steht die Startadresse und Länge einer BIN-Datei nicht im ersten Dateiblock, sondern im Directory. Auf BIN-Dateien kann ohne den TBIN-Zusatz mit BSAVE, BLOAD und BRUN zugegriffen werden. Doch sind die BLOAD/BSAVE-Befehle im BASIC.SYSTEM dergestalt generalisiert worden, daß *jede* andere Dateiart *mit* dem entsprechenden Typzusatz geladen und gespeichert werden kann, z.B.
BLOAD TEXTFILE, *TTXT*, A$1000
BLOAD BASICPROGRAMM, *TBAS*, A$0801
Directories (Volume-Directory und Subdirectories) können ebenfalls mit BLOAD geladen, jedoch (aus Sicherheitsgründen) nicht mit BSAVE zurückgespeichert werden:
BLOAD/VOLUMEDIR, A$2000, TDIR
BLOAD/VOLUMEDIR/SUBDIR, A$1000, TDIR

BAS-Dateien sind Applesoft-Programme, auf die mit den Befehlen SAVE, LOAD, RUN und CHAIN zugegriffen wird. Im Gegensatz zu DOS 3.3 steht die Länge des Applesoft-Programms nicht im ersten Dateiblock, sondern im Directory. Die Startadresse (normalerweise $0801) kann dem Directory mit dem normalen CATALOG-Befehl nicht entnommen werden, obgleich sie sich in dem Zusatzinfo-Feld (s.u.) befindet.

VAR-Dateien umfassen die Namen und Werte *aller* Variablen eines Applesoft-Programms und werden mit STORE/VARIABLENDATEI gespeichert und mit RESTORE/VARIABLENDATEI geladen (Näheres hierzu später).

DIR-Dateien sind Inhaltsverzeichnisse, und zwar entweder das Hauptinhaltsverzeichnis (Volume-Directory) mit maximal 51 Einträgen oder eines der beliebig vielen Unterinhaltsverzeichnisse (Subdirectories) mit jeweils beliebig vielen Einträgen.

1.10. CAT, CATALOG, PREFIX, CREATE

CAT /VOLUME/DIR , Ss, Dd, Tt
CATALOG /VOLUME/DIR, Ss, Dd, Tt
PREFIX /VOLUME/DIR, Ss, Dd
CREATE /VOLUME/DIR, TDIR, Ss, Dd, Tt
CREATE /VOLUME/DIR/DATEI, Tt, Ss, Dd

Beispiele:
CAT
CAT, D2 (*nicht* CAT D2!)
CAT, S6
CAT, D2, S6
CAT, S6, D2
CAT /VOLUME (*nicht* CAT VOLUME!)
CAT /VOLUME/DIR (*nicht* CAT /DIR oder CAT DIR!)
CATALOG /RAM/SUBDIR, S3, D2
PREFIX
PREFIX, S6, D1
PREFIX /VOLUME/DIR
PREFIX /
CREATE NEWDIR, S6, D1
CREATE /VOLUME/NEWDIR
CREATE /VOLUME/NEWDIR, TDIR, S6, D2
CREATE /VOLUME/DIR/SYSFILE, TSYS

s = Slot-Nummer im Bereich 1-7
d = Drive-Nummer im Bereich 1-2
t = 3stelliges Dateitypkürzel, hier „DIR".
Wenn das Präfix oder der Pfadname vollständig angegeben werden, sind die Slot-Drive-Parameter entbehrlich. Diskettenlaufwerke haben meist 2 Drives an *demselben* Slot-Controller, während Festplatten und kleinere RAM-Disks mit einem *einzigen* Drive angesprochen werden. **„Ss" ohne „Dd" bedeutet stets „Ss, D1".**

CAT, TTXT würde einen selektiven Catalog in bezug auf die TXT-Files anzeigen. Ähnliches gilt für CATALOG, TBIN usw.

ProDOS-Directory mit Zählleisten

(für EINTRAG.SUCHER und EINTRAG.ANALYSE, s.S. 107):

```
/DIRECTORY
 NAME            TYPE  BLOCKS  MODIFIED         CREATED          ENDFILE SUBTYPE
1234567890123456789012345678901234567890123456789012345678901234567890123456789
1                18    24      31       41      48        58  .  64       73
*                *     *       *        *       *         *      *        *
*PRODOS          SYS       31   1-JAN-84  0:00  <NO DATE>           15360
*BASIC.SYSTEM    SYS       21  15-NOV-83  0:00  <NO DATE>           10240
*FILER           SYS       51  <NO DATE>        <NO DATE>           25600
*CONVERT         SYS       42   1-NOV-83  0:00  <NO DATE>           20481
*STARTUP         BAS        1  23-DEC-84  0:00  23-DEC-84  0:00       176
*PRODOS.EDITOR   BIN       13   1-AUG-84  0:00   1-AUG-84  0:00      5707 A=$3FA5
 VARIABLEN       VAR        1  <NO DATE>  0:00  <NO DATE>             277
 SELBSTDEFINIERT $F1        1  <NO DATE>  0:00  <NO DATE>               0
 SUBDIRIRECTORY  DIR        2  <NO DATE>  0:00  <NO DATE>            1024
 SEQUENTIELL     TXT        1  <NO DATE>  0:00  <NO DATE>               7 R=    0
 RANDOM          TXT        1  <NO DATE>  0:00  <NO DATE>               7 R=  100
*N23456789012345 TTT    32768  TT-MMM-JJ HH:MM  TT-MMM-JJ HH:MM 16777215 A=$HHLL
1234567890123456789012345678901234567890123456789012345678901234567890123456789
            13                      35                    58
            *                       *                     *
BLOCKS FREE:   53     BLOCKS USED:  227     TOTAL BLOCKS:  280
            65536                 65536                 65536 (Maximalwerte)
```

Nachdem wir die USERS.DISK (oder eine andere ungeschützte ProDOS-Diskette) mit PR#6 o.ä. gebootet und dann ein ggf. vorhandenes Menü verlassen haben, befinden wir uns im Nicht-RUN-Modus, so daß wir jetzt über die Tastatur Befehle eingeben können.

```
Wenn man ein Directory mit
10 PRINT CHR$(4) "OPEN/DIRECTORY"
20 PRINT CHR$(4) "READ/DIRECTORY"
30 INPUT Stringvariable usw.
40 PRINT CHR$(4) "CLOSE"
einliest, so gilt:
      1. String: Directory-Name
      2. String: Kopfleiste (NAME TYPE BLOCKS usw.)
      3. String: Return (Leerzeile)
4. N-1. String: Dateieintrag-Zeilen
      N. String: Return (Leerzeile)
    N+1. String: Fußzeile (BLOCKS FREE: usw.)

Beispiel einer Eintraganalyse
(erstellt mit Programm EINTRAG.ANALYSE)

/STIEHL/DIR/EINTRAG.ANALYSE

 NAME              TYPE   BLOCKS  MODIFIED      CREATED       ENDFILE SUBTYPE
 EINTRAG.ANALYSE BAS          6  13-JAN-85 0:00 13-JAN-85 0:00    2112

 Speichertyp: $2
 Namenslänge: $F
 Dateiname:   $45 $49 $4E $54 $52 $41 $47 $2E $41 $4E $41 $4C $59 $53 $45
 Dateityp:    $FC
 Hauptzeiger: $00D5   ($HHLL)
 Blockanzahl: $0006   ($HHLL)
 Dateilänge:  $000840 ($HHMMLL)
 Create-Dat:  $2D $AA
 Create-Uhr:  $00 $00
 Version-Nr:  $00 $00
 Zugriff:     $E3
 Zusatzinfo:  $0801   ($HHLL)
 Mod-Datum:   $2D $AA
 Mod-Uhr:     $00 $00
 Kopfzeiger:  $00D3   ($HHLL)
```

Mit
CAT
oder mit
CATALOG
können wir uns nunmehr den Catalog oder das Directory (Inhaltsverzeichnis) der Diskette

in 40 Zeichen/Zeile (CAT) oder in 80 Z/Z (CATALOG) ansehen. Das Listing „PRODOS.DIRECTORY" zeigt einen solchen ProDOS-Catalog in 80 Z/Z. Betrachten wir hierzu den Dateieintrag „PRODOS":

*PRODOS SYS 31 1-JAN-84 0:00 <NO DATE> .... 15360 ....

Sternchen: Ein vorangestelltes Sternchen zeigt an, daß die Datei „PRODOS" schreibgeschützt ist. Fehlte das Sternchen, dann könnte die Datei gelöscht oder umbenannt werden.
Dateiname: Der Name „PRODOS" besteht hier aus 6 Großbuchstaben. Insgesamt kann ein Dateiname bis zu 15 Zeichen umfassen, und zwar nur Großbuchstaben, Ziffern und Punkte, aber keine Leerzeichen, Ctrl-Buchstaben, Kleinbuchstaben und sonstigen Sonderzeichen. Das erste Zeichen des Dateinamens muß ein Großbuchstabe sein. Beispiele:
SPIELPROGRAMM10 (mit Ziffern)
BASIC.SYSTEM (mit Punkt)
MEIN PROGRAMM (falsch, da Leertaste)
BASIC-PROGRAMM (falsch, da Bindestrich)
Dateityp: 3-Buchstaben-Kürzel, hier „SYS"
Blockanzahl: Anzahl der von der Datei belegten Blocks, hier „31".
Datum der Modifikation: In der Form TT-MMM-JJ, hier 1-JAN-84.
Uhrzeit der Modifikation: In der Form HH:MM, hier „ 0:00"
Datum der Anlage: Tag, an dem die Datei erstmals angelegt („kreiert") wurde, hier „<NO DATE>".
Uhrzeit der Anlage: Diese fehlt hier.
Dateilänge: Anzahl der Bytes der Datei, hier „15359". ENDFILE = End of File steht sowohl für die Gesamtanzahl der Bytes wie auch für die absolute Position des letzten Bytes der Datei, d.h. „PRODOS" umfaßt 15360 Bytes, und das letzte Byte dieser Datei ist das 15359ste Byte, da die Zählung mit 0 beginnt.
Zusatzinfo: Die Zusatzinformation wird im Directory als SUBTYPE bezeichnet, während das technische Manual von AUXTYPE (Auxiliary Type) spricht. Bei BIN-Dateien steht hier die Startadresse, bei Random-TXT-Dateien die Recordlänge. Bei SYS-, BAS-, DIR- und VAR-Dateien wird der mögliche Inhalt der Zusatzinfo nicht angezeigt.

Unter ProDOS gibt es zwei Arten von Inhaltsverzeichnissen:
Volume-Directory: Das Volume-Directory ist das Hauptinhaltsverzeichnis. Es belegt auf der Diskette die Blocks 2-5 und kann insgesamt 51 Dateieinträge aufnehmen (12 im ersten Block und je 13 in den drei weiteren Blocks = $12 + 3 \cdot 13 = 51$). Für 140K-Disketten ist dies oft ausreichend.
Subdirectory: Neben dem einzigen Volume-Directory können mit dem CREATE-Befehl beliebig viele Subdirectories angelegt werden, wobei jedes Subdirectory im übrigen beliebig viele Dateieinträge umfassen kann. Ein Subdirectory ist im Directory durch den

Dateityp „DIR" gekennzeichnet. Für Subdirectories gibt es keine festen Blocks auf der Diskette, d.h. sie können sich überall dort befinden, wo auf der Diskette gerade Platz ist. Übrigens kann auch das Subdirectory selbst wieder Sub-Subdirectories enthalten, doch sollte man diese möglichst vermeiden, denn je mehr (verschachtelte) Subdirectories, desto langsamer ist der Diskettenzugriff.

Unter ProDOS lassen sich drei Arten von Namen unterscheiden:
Volume-Name: Dies ist der beim Formatieren festgelegte Name der Diskette, wie er aus dem Volume-Directory ersichtlich ist (max. 15 Zeichen).
Subdirectory-Name: Dies ist der Name eines von beliebig vielen Subdirectories (max. 15 Zeichen).
File-Name: Dies ist der eigentliche Name einer Datei (max. 15 Zeichen).

Die Summe aus „/" + Volume-Name + „/" + Subdirectory-Name + „/" + File-Name bildet den sog. Pfadnamen (Pathname) oder vollständigen Namen, der inklusive Schrägstriche maximal 64 Zeichen umfassen kann. Die Namensbestandteile müssen durch Schrägstriche abgegrenzt werden, wobei der Schrägstrich nach dem letzten Bestandteil fakultativ ist, also
CAT/VOLUME/DIR/
oder
CAT/VOLUME/DIR

Exkurs: Hierarchische Dateistruktur

ProDOS gilt als hierarchisches Betriebssystem ähnlich wie UNIX usw. Hierzu ein Beispiel aus dem Bibliothekswesen:
ProDOS-Dateien sind wie Bücher. Jede Datei hat einen Namen, wie auch jedes Buch einen Titel hat. Der Dateiname ist nicht mit der Datei identisch, wie auch der Buchtitel nicht mit dem Buch identisch ist.
Das Volume-Directory oder Hauptinhaltsverzeichnis ist selbst eine Datei und enthält bis zu 51 Dateinamen mit sonstigen Informationen (Dateilänge usw.). Ebenso ist eine Bibliographie selbst ein Buch und enthält soundsoviel Buchtitel mit sonstigen Informationen (Seitenzahlen usw.).
Nun kann ein Volume-Directory neben oder anstelle der Namen der „normalen" Datenfiles auch Namen von Subdirectories oder Unterinhaltsverzeichnissen enthalten, wie es ja auch sog. „Bibliographien der Bibliographien" gibt, die nicht die Titel von „normalen" Büchern, sondern die Titel anderer Bibliographien enthalten. Dieses „Spielchen" kann man fortsetzen, so daß „Bibliographien der Bibliographien der Bibliographien" entstehen können. Betrachten wir hierzu folgenden Fall:
Das Volume-Directory „VOLDIR" einer ProDOS-Diskette enthalte u.a. den DIR-Ein-

trag „SUBDIR", ersichtlich mit dem Befehl CAT/VOLDIR. Nehmen wir nunmehr an, daß wir nach CAT/VOLDIR/SUBDIR in dem Subdirectory „SUBDIR" u.a. auch den DIR-Eintrag „SUBSUBDIR" sehen. Nach CAT/VOLDIR/SUBDIR/SUBSUBDIR würden wir dann das Subsubdirectory „SUBSUBDIR" untersuchen können, das jedoch nunmehr kein weiteres Subdirectory, sondern nur noch „normale" Datenfiles aufweisen möge. Dann gilt: „SUBSUBDIR" entspräche einer Bibliographie, „SUBDIR" einer Bibliographie der Bibliographien und „VOLDIR" einer Bibliogaphie der Bibliographien der Bibliographien. Da hier selbst Logiker „durcheinanderkommen" können, sei die sog. „Russellsche Paradoxie" nach dem Buch „Grundriß der Logistik" von Bochenski/Menne, 3. Aufl., S. 75, zitiert:
„Ein Katalog einer Bibliothek pflegt die Bücher dieser Bibliothek aufzuführen. Auch der Katalog selbst kann als ein Buch dieser Bibliothek betrachtet werden und so in sich selbst aufgeführt werden. Soll nun ein vollständiger Katalog aufgestellt werden, der alle, aber auch nur alle Kataloge enthält, die sich selbst nicht enthalten, so erhebt sich die Frage, ob dieser Katalog sich selbst enthalten müßte. Tut er das, so ist er selber kein Katalog, der sich nicht selbst enthält, darf sich also demgemäß nicht selber aufführen; enthält er sich aber selber nicht, so ist er einer der Kataloge, die sich nicht selbst enthalten, muß sich also selber aufführen – Aus jeder Beantwortung der Frage läßt sich ihr Gegenteil auf Grund der vorausgesetzten Bedingungen ableiten."

Mit dem PREFIX-Befehl kann man das Präfix, d.h. den Dateinamenvorspann festlegen. Danach genügt wie bei DOS 3.3 der eigentliche Dateiname (ohne Schrägstrich) als Zusatz zu den Befehlen, also statt z.B.
LOAD /VOL/SUBDIR/PROGRAMM.XYZ
zunächst
PREFIX /VOL/SUBDIR
und dann nur noch
LOAD PROGRAMM.XYZ

Beispiele:

1. NEW
2. 10 HOME
3. CREATE /VOL/DIR, TDIR
4. PREFIX /VOL/DIR
5. SAVE DEMO
6. CATALOG
7. PREFIX /VOL
8. CATALOG
9. PREFIX

1. Mit NEW löschen wird zunächst ein möglicherweise noch im Speicher befindliches Applesoft-Programm.
2. Mit 10 HOME legen wir jetzt ein Testprogramm an.
3. Mit CREATE /VOL/DIR, TDIR erstellen wir jetzt auf der Diskette mit dem Volume-Namen VOL ein Subdirectory mit dem Namen DIR.
4. Mit PREFIX /VOL/DIR legen wir das Präfix „/VOL/DIR/" fest.
5. Der Befehl SAVE DEMO speichert damit das Testprogramm genauso im Subdirectory DIR, als hätten wir SAVE /VOL/DIR/DEMO gesagt.
6. Mit CATALOG können wir automatisch das Subdirectory DIR ansehen.
7. Mit PREFIX /VOL können wir wieder zum Volume-Directory zurückschalten.
8. Der CATALOG-Befehl zeigt damit wieder das Volume-Directory an.
9. Der PREFIX-Befehl zeigt jetzt (wieder) „/VOL/" an.

Der PREFIX-Befehl bietet folgende Besonderheiten:

PREFIX/VOLUME/SUB legt das neue Präfix fest.
PREFIX, S6, D1 legt das *Volume*-Directory der Diskette in Slot 6, Drive 1 als neues Präfix fest.
PREFIX oder
10 PRINT CHR$ (4) "PREFIX": INPUT P$
zeigt im Nicht-RUN-Modus das momentane Präfix an oder weist im RUN-Modus bei einem unmittelbar folgenden INPUT der String-Variablen – der Name der Variablen ist beliebig – das momentane Präfix zu.
PREFIX/ entfernt das Präfix insgesamt. Danach verhält sich das BASIC.SYSTEM wie DOS 3.3, d.h. es können jetzt mit den Parametern Ss und Dd die jeweils aktiven Slots und Drives (als Default-Werte) bestimmt werden.

Beispiel:

1. PREFIX, S6, D1
2. CAT, D2
3. CAT
4. PREFIX/
5. CAT, D2
6. CAT

1. Zunächst wird der Volume-Name von S6, D1 als Präfix festgelegt.
2. CAT, D2 zeigt den Catalog der Diskette in Drive 2 an.
3. CAT ohne Parameter zeigt jedoch jetzt wieder Drive 1 an, da das alte Präfix noch gültig ist (Dies weicht von DOS 3.3 ab).

4. PREFIX/ entfernt das Präfix.
5. CAT, D2 zeigt den Catalog von Drive 2 an.
6. CAT zeigt jetzt erneut Drive 2 an (Dies entspricht jetzt DOS 3.3).

Default-Werte (Ersatzwerte)

a) DOS-3.3-Emulation-Default: Wenn wir mit PREFIX/ das Präfix ganz entfernen, so „weiß" ProDOS nichts mehr vom Präfix und verhält sich hinsichtlich der Slot-Drive-Default-Werte wie DOS 3.3: Der zuletzt angesprochene Slot/Drive bleibt so lange aktiv, bis explizit ein neuer Slot/Drive angegeben wird.
b) ProDOS-Default: Das mit PREFIX/VOLUME.NAME/SUBDIR.NAME definierte Default-Präfix bleibt so lange erhalten, bis es explizit neu definiert wird. Slot- und Drive-Parameter bei BASIC.SYSTEM-Befehlen ändern nicht das Präfix (außer natürlich durch den Präfix-Befehl PREFIX, Ss, Dd selbst).

Der CREATE-Befehl dient nicht nur zum Neuanlegen von Subdirectories. Im einzelnen gilt:

CREATE /VOL/CCC
oder
CREATE /VOL/CCC, TDIR
würde in beiden Fällen ein Subdirectory erzeugen, d.h. wenn CREATE *ohne* „TDIR" verwendet wird, ist für das BASIC.SYSTEM stets ein Subdirectory gemeint.
CREATE /VOL/CCC, TSYS
oder allgemein
CREATE /VOL/CCC, Tt
legt einen neuen Datenfile an, der zunächst nur 1 Block umfaßt und dessen ENDFILE bzw. EOF = 0 ist. Dies ist stets dann zwingend erforderlich, wenn ein Datenfile nicht mit den üblichen BASIC.SYSTEM-Befehlen durch SAVE (BAS-File), STORE (VAR-File) oder BSAVE (BIN-File) angelegt werden kann, z.B. bei SYS-Files und selbstdefinierten Dateitypen ($F1-$F8).

Wann Volume-Directory, wann Subdirectory?

Jeder ProDOS-Datenträger enthält ein Volume-Directory, das maximal 51 (bei der 64K-Karte-RAM-Disk nur 12) Dateinamen umfaßt. Ein Volume-Directory wird sozusagen mit dem FORMAT-Befehl automatisch angelegt und kann später zwar umbenannt, nicht jedoch gelöscht werden, selbst wenn es leer wäre. Umgekehrt kann ein Subdirectory beliebig viele Dateinamen umfassen, muß jedoch zunächst mit dem CREATE-Befehl

angelegt werden, kann aber dafür nachträglich wieder gelöscht werden, falls alle darin enthaltenen Dateinamen zuvor gelöscht wurden (mit dem DELETE-Befehl).
Mit jedem Subdirectory oder gar Subsubdirectoy erhöht sich die Zugriffszeit auf die eigentlichen Datenfiles, weil dann der „Pfad" immer länger wird. Wenn man mit normalen 140K-Disketten arbeitet und die 51 Einträge des Volume-Directory ausreichen, sollte man auf Subdirectories ganz verzichten. Wird jedoch ein Subdirectory benötigt, so sollte der Subdirectory-Eintrag im ersten Block des Volume-Directory erfolgen, d.h. einer der ersten 12 Dateinamen oder gar der allererste Dateiname des Volume-Directory sein. Dann wird das Subdirectory sofort gefunden, womit sich die Zugriffsgeschwindigkeit erhöht. Von Subsubdirectories oder gar Subsubsubdirectories sollte man selbst bei Festplatten völlig absehen.

Wann Präfix, wann Slot/Drive?

Unter ProDOS sollte möglichst slot-drive-unabhängig programmiert werden. Da das BASIC.SYSTEM – im Gegensatz zum MLI – den ONLINE-Befehl nicht kennt, der alle aktiven Slots/Drives mit den entsprechenden Volume-Präfixen ausweist, ergeben sich zwei Möglichkeiten nach einem Diskettenwechsel:

1. Bei der Programmentwicklung ermittelt man das neue Präfix mit PREFIX, S6, D1 usw. („quick and dirty").
2. In Anwenderprogrammen ermittelt man das neue Präfix über die On-Error-Routine wie folgt (Beispiel):

```
10 ONERR GOTO 20
20 PRINT "Disketten mit den Namen PROGRAMM und DATEN in beliebige Laufwerke einlegen! ": GET X$
30 PRINT CHR$ (4) "PREFIX/PROGRAMM": PRINT CHR$ (4) "PREFIX/DATEN"
40 POKE 216, 0: REM ONERR wieder abstellen
```

Schwieriger wird es, wenn ein Anwenderprogramm, z.B. eine kleine Utility, kein festes Präfix vorschreibt. Dann sollten alle von der Utility benutzten Programm- und Datenfiles in *demselben* Directory aufgeführt werden. Zu beachten ist ferner, daß nach dem Booten das BASIC.SYSTEM nicht automatisch das Präfix des Boot-Slots festlegt. Deshalb sollte das STARTUP-Programm stets mit folgender Zeile beginnen, die als Präfix das Volume-Directory festlegt.

```
10 PRINT CHR$ (4) "PREFIX": INPUT P$: PRINT CHR$ (4) "PREFIX";P$
```

Wenn sowohl Präfix- als auch Slot/Drive-Parameter benutzt werden, hat der Präfix-Parameter Vorrang. Beispiel:

/RAM sei RAM-Disk in S3, D2
Momentanes Präfix sei /VOLUME in S6, D1
Dann führt
CAT /RAM, S6, D1
trotzdem den CAT-Befehl in bezug auf /RAM aus, wobei allerdings zuvor MLI-intern auf /VOLUME zugegriffen wurde.

1.11. VERIFY, LOCK, UNLOCK, DELETE, RENAME

VERIFY /VOLUME/DIR, Ss, Dd
LOCK /VOLUME/DIR/FILE, Ss, Dd
UNLOCK /VOLUME/DIR/FILE, Ss, Dd
DELETE /VOLUME/DIR/FILE, Ss, Dd
RENAME /VOL/DIR/FILEALT, /VOL/DIR/FILENEU, Ss, Dd

Beispiele:
VERIFY FILE
VERIFY FILE, D2
VERIFY FILE, S6
VERIFY FILE, S6, D2
VERIFY /VOLUME
VERIFY /VOLUME/DIR
VERIFY /VOLUME/DIR/FILE, S6, D1
LOCK FILE
LOCK /VOLUME/DIR (*nicht* LOCK /VOLUME!)
UNLOCK /VOLUME/DIR/FILE
UNLOCK FILE, S3, D2
DELETE FILE
DELETE FILE, D1
DELETE /VOLUME/DIR/FILE
DELETE /VOLUME/DIR (*nicht* DELETE /VOLUME!)
RENAME ALTNAME, NEUNAME
RENAME /VOLUME1, /VOLUME2, S6, D1
RENAME /VOLUME/DIRALT, /VOLUME/DIRNEU

VERIFY (to verify = prüfen) prüft, ob ein Name vorhanden ist. Im Gegensatz zum DOS-3.3-Befehl liest VERIFY also nicht die Datei selbst ein, um deren Intaktheit (Bad Blocks) festzustellen. VERIFY kann auf die Namen des Volume-Directory, der Subdirec-

tories sowie der eigentlichen Datenfiles angewandt werden. VERIFY ist dann nützlich, wenn man bei einem großen Directory schnell prüfen möchte, ob eine bestimmte Datei existiert. Ferner kann man bei einem Anwenderprogramm über VERIFY ermitteln, ob die richtige Diskette eingelegt wurde:

```
10 ONERR GOTO 30
20 PRINT CHR$ (4); "VERIFY DATEI.1": GOTO 40
30 PRINT "Richtige Diskette einlegen": GOTO XYZ: REM Menü
40 REM Hier geht es weiter.
```

Probieren Sie

```
10 PRINT CHR$ (4); "VERIFY"
20 PRINT "Meldung gesehen?"
```

Sie werden jetzt eine Copyright-Meldung sehen. Dies ist ein kleiner Witz der Firma Apple. Es entsteht also keine Fehlermeldung, obwohl nach VERIFY der Dateiname fehlt!

LOCK und UNLOCK bewirken/entfernen einen Dateischreibschutz (to lock = zuschließen, to unlock = aufschließen). LOCK versieht den Namen einer Datei im Directory mit einem „*" = Sternchen. Dergestalt markierte Dateien können nicht mehr gelöscht werden. Auch kann der Name nicht mehr mit RENAME geändert werden. UNLOCK entfernt wieder das Sternchen. Es können nur Subdirectory-Namen sowie eigentliche Datenfile-Namen geLOCKt und UNLOCKt werden, d.h. das Volume-Directory kann nicht geLOCKt werden (weil es auch nicht DELETEt werden kann).
Wenn man den Namen eines Subdirectory LOCKt, dann kann es zwar selbst nicht mehr gelöscht werden, wohl aber alle darin enthaltenen Dateien, falls deren Namen nicht geLOCKt sind.
Wichtige Dateien sollte man LOCKen, wichtige Disketten mit einem Schreibschutz versehen (Kerbe zukleben).

DELETE (to delete = löschen) löscht einen Datenfile oder ein Subdirectory. Ein Volume-Directory kann nicht gelöscht werden. Ferner können nur UNLOCKte Files gelöscht werden. Im Gegensatz zum DOS-3.3-DELETE-Befehl, der im Catalog-Eintrag nur ein Byte löschte, macht der ProDOS-DELETE-Befehl – im MLI DESTROY = „zerstören" genannt – wirklich „reinen Tisch", d.h. es wird der Index-Block (s. Band 1, S. 138) mit Nullen gefüllt, womit es ohne erheblichen Aufwand kaum möglich ist, eine gelöschte Datei wieder zum Leben zu erwecken. Deshalb gilt für ProDOS: *Think first, destroy later!*

RENAME (to rename = umbenennen) ändert den Namen eines Volume-Directory, eines Subdirectory oder einer eigentlichen Datei. Es gilt immer „RENAME alter Name, neuer

Name", d.h. der neue Name steht – getrennt durch ein Komma – nach dem alten Namen. Für RENAME gilt wie für DELETE, daß der jeweilige Name nicht geLOCKt sein darf. Unter ProDOS ist es im Gegensatz zu DOS 3.3 nicht mehr möglich, daß man mit RENAME zwei *gleiche* Namen in *demselben* Directory erzeugen kann. Man beachte jedoch, daß Datenfiles in *verschiedenen* Subdirectories *gleiche* Namen haben können. Beispiel:

/VOLUME
/VOLUME/DATEI.1
/VOLUME/SUBDIRECTORY.1
/VOLUME/SUBDIRECTORY.1/DATEI.1
/VOLUME/SUBDIRECTORY.2
/VOLUME/SUBDIRECTORY.2/DATEI.1

Hier gibt es drei Dateien mit demselben Namen „DATEI.1" in drei verschiedenen Directories, nämlich im Volume-Directory und in den zwei Subdirectories. Ferner könnte ein Subdirectory den *gleichen* Namen wie das übergeordnete Volume-Directory haben, doch sollte man dies natürlich tunlichst vermeiden.
Bezüglich des Präfixes sind noch drei Dinge zu beachten:
1. Das Präfix vor Alt- und Neuname muß *identisch* sein, also nicht
RENAME/VOLUME/DATEI.1,/VOLUME/SUBDIRECTORY.1/DATEI.XYZ
2. Falls der geänderte Name momentanes Präfix ist, muß nach dem RENAME das Präfix neu definiert werden, weil sonst das BASIC.SYSTEM die Diskette bei CATALOG usw. ohne Parameter nicht mehr findet, also
PREFIX/VOLUME
RENAME/VOLUME,/VOL1
PREFIX/VOL1
und nicht (ausprobieren!)
PREFIX/VOL.ALT
RENAME/VOL.ALT,/VOL.NEU
CATALOG
Was passiert jetzt?
3. Die Parameter S und D als Ersatz für ein Präfix dürfen nur nach dem Neunamen stehen, also nicht
RENAME XXX, S6, D1, YYY, S6, D1
sondern nur
RENAME XXX, YYY, S6, D1
Im übrigen gilt für RENAME wie auch für alle anderen BASIC.SYSTEM-Befehle, daß ein Pfadname, falls er benutzt wird, stets der *erste* Parameter in der Parameter-Liste sein muß (also *nicht* „CAT S3, /RAM" usw.).

1.12 RUN, LOAD, SAVE, CHAIN, STORE, RESTORE, FRE

RUN /PROGRAMM, §z, Ss, Dd
LOAD /PROGRAMM, Ss, Dd
SAVE /PROGRAMM, Ss, Dd
CHAIN /PROGRAMM, §z, Ss, Dd
STORE /VARIABLEN, Ss, Dd
RESTORE /VARIABLEN, Ss, Dd

Beispiele:
RUN (wenn Programm bereits im Speicher ist)
RUN /VOL/DIR/PROGRAMM1
RUN DEMO, D2, S3
RUN DEMO, §1000
LOAD /VOLUME/SPIELE/MUEHLE
SAVE /VOL/DEMO
1000 PRINT CHR$ (4); "CHAIN ZWEITMODUL, §2000"
100 PRINT CHR$ (4); "STORE VARIABLEN"
200 PRINT CHR$ (4); "RESTORE VARIABLEN"

RUN ohne Programmnamen startet ein Applesoft-Programm, das sich bereits im Speicher befindet. **LOAD** lädt ein Applesoft-Programm von einem externen Datenträger (Diskette). RUN + Programmname lädt ein Programm von Diskette und startet es dann.
RUN DEMO
entspricht damit
LOAD DEMO
RUN
Im Gegensatz zu DOS 3.3 kann bei RUN jetzt zusätzlich die Zeilennummer, ab der das Programm gestartet werden soll, angegeben werden. Zu diesem Zweck muß vor die Zeilennummer das Paragraph-Zeichen § (deutscher Zeichensatz) oder das At-Zeichen @ (amerikanischer Zeichensatz) gesetzt werden, also
RUN DEMO, §1000 oder
RUN DEMO, @1000
Die angegebene Zeilennummer muß tatsächlich existieren, sonst erfolgt eine Fehlermeldung. Man beachte, daß „RUN NAME, §z" stets das *ganze* Programm einlädt, auch wenn es erst ab Zeile z gestartet wird. Der RUN-Befehl kann auch im RUN-Modus, d.h. in einem laufenden Applesoft-Programm, erteilt werden, um z.B. aus dem einen Programm heraus das andere zu starten, z.B.
1000 PRINT CHR$ (4); "RUN PROGRAMM.B"

Auch LOAD aus einem Programm heraus ist möglich, doch normalerweise nicht sinnvoll, da man danach in den Direkt-Modus gelangt.

Anfangsadresse, Endadresse und Länge eines Applesoft-Programm errechnen sich wie folgt:
Anfangsadresse (TXTTAB) ($0067-$0068): A = PEEK (103) + PEEK (104) * 256
Endadresse (PRGEND) ($00AF-$00B0): E = PEEK (175) + PEEK (176) * 256
Länge: L = E - A (als ENDFILE aus Directory ersichtlich)
Ein Applesoft-Programm beginnt normalerweise ab $0801 = 2049 im RAM-Speicher, wobei A - 1 = $0800 = 2048 eine Null enthalten muß. Führen wir dazu folgendes Experiment durch:

```
NEW
CALL -151
0800: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Ctrl-C
1 X = 1
RUN
CALL -151
0800. 080B
```

Jetzt sehen wir einen Hex-Dump, den wir untereinander schreiben:

```
2048 $0800: 00
2049 $0801: 09   $0809 Link-Byte
2050 $0802: 08
2051 $0803: 01   $0001 Zeilennummer 1
2052 $0804: 00
2053 $0805: 58   X
2054 $0806: D0    =
2055 $0807: 31   1
2056 $0808: 00
2057 $0809: 00   $0000 = End of Line
2058 $080A: 00
2059 $080B: 58   Variablenname „X"
```

Die Anfangsadresse A ist hier 2049 oder $0801, und die Endadresse ist 2059 oder $080B. Dies ergibt eine Länge L von 2059 - 2049 = 10 = $0B. Das wirkliche Ende des Programms ist bei $080A, weil ab $080B bereits die Variablen beginnen, hier die Variable X. Das

BASIC.SYSTEM speichert auf der Diskette 10 Bytes, wobei das 10. Byte nicht $58 = „X", sondern $00 ist. Anders war es bei DOS 3.3, das in diesem Fall den Inhalt von $080B, d.h. $58, mit abgespeichert hätte. Bei Applesoft-Programmen errechnet sich also die Länge durch die Formel L = E - A, weil E de facto bereits E + 1 ist. Bei BLOAD/BSAVE (s.u.) lautet die Formel L = E - A + 1.
Die Anfangsadresse steht bei Applesoft-Programmen im sog. Zusatzinfo-Feld des Directory-Eintrags, das jedoch beim CATALOG-Befehl nicht angezeigt wird. Die Länge ist der ENDFILE-Spalte zu entnehmen und steht im Directory-Eintrag und nicht – wie bei DOS 3.3 – zu Anfang des ersten Programmsektors. Normalerweise braucht man sich nicht um Anfangs- und Endadresse sowie Länge zu kümmern, weil der SAVE- und LOAD-Befehl automatisch diese Berechnungen durchführt.
Wenn man ein Applesoft-Programm durch Änderung der Zeiger (s. Kapitel 2.2) beispielsweise nach $4001 einlädt und dann wieder abspeichert, steht im Zusatzinfo-Feld tatsächlich $4001. Führt man jedoch danach zur Normalisierung der Pointer einen Kaltstart durch und lädt dieses Programm dann erneut ein, so beginnt es wieder bei $0801, d.h. der Inhalt der Zusatzinfo wird vom LOAD-Befehl ignoriert und hat damit praktisch keinen Nutzen.
Die einzelnen Datenblocks eines BAS-Files enthalten den Speicherauszug des Applesoft-Programms ab $0801 oder allgemein von TXTTAB bis PRGEND.

CHAIN (to chain = verketten usw.) entspricht dem aus einem laufenden Applesoft-Programm erteilten RUN-Befehl mit dem Unterschied, daß CHAIN die Variablen des momentanen Programms *nicht* löscht. CHAIN verwendet man insbesondere dann, wenn man ein sehr umfangreiches Gesamtprogramm in ein Hauptprogramm und mehrere Unterprogramme aufteilt. Da das von CHAIN aufgerufene Unterprogramm bereits zuvor dimensionierte Arrays nicht erneut dimensionieren darf, kann man folgende Hierarchie der Programm-Module aufstellen:
1. STARTUP-Programm, das die erforderlichen Dimensionierungen vornimmt und zu dem man später mit CHAIN *nicht* mehr zurückkehrt.
2. Hauptprogramm, von dem man zu den verschiedenen Unterprogrammen per CHAIN-Befehl verzweigt.
3. Eines oder mehrere Unterprogramme, die zum Hauptprogramm zurückverzweigen.

Da bei CHAIN auch eine Zeilennummer als Parameter zulässig ist, kann man auf das STARTUP-Programm ganz verzichten und die Dimensionierungen zu Beginn des Hauptprogramms vornehmen, die dann von „CHAIN, §z " übersprungen werden. Beispiel:

HAUPTPROGRAMM

```
100 PRINT "HAUPTPROGRAMM"
110 DIM A(1000),A%(1000),A$(1000)
120 FOR X = 1 TO 1000:A(X) = X:A%(X) = X:
    A$(X) = STR$ (X): NEXT
130 FOR X = 1 TO 1000 STEP 100:A$(X) = "": NEXT
140 FOR X = 1 TO 1000: PRINT A(X),A%(X),A$(X): NEXT
150 INPUT "UNTERPROGRAMM J/N ";X$: IF X$ = "N" THEN END
160 PRINT CHR$ (4)"CHAIN UNTERPROGRAMM"
```

UNTERPROGRAMM

```
100 PRINT "UNTERPROGRAMM"
110 FOR X = 1 TO 1000 STEP 100:A$(X) = "AAAAA": NEXT
120 FOR X = 1 TO 1000: PRINT A(X),A%(X),A$(X): NEXT
130 INPUT "HAUPTPROGRAMM J/N ";X$: IF X$ = "N" THEN END
140 PRINT CHR$ (4)"CHAIN HAUPTPROGRAMM,$130"
```

STORE bzw. **RESTORE** speichern bzw. laden *alle* Variablen. Man kann also keine Auswahl-Variablenliste als Parameter an den Befehl anhängen. Diese Befehle sind dann nützlich, wenn man später wieder *alle* Variablen braucht; beispielsweise kann man bei einem Basic-Spiel mit STORE den momentanen Spielstand abspeichern. Ansonsten belegen die durch STORE entstandenen VAR-Dateien (Variablendateien) zuviel Platz auf der Diskette. Der STORE-Befehl führt zunächst intern den FRE-Befehl aus und packt dann alle Variablennamen und Variablenwerte unterhalb von HIMEM, die dann als Speicherauszug auf der Diskette als VAR-File abgelegt werden. Der Directory-Eintrag enthält im Zusatzinfo-Feld die Anfangsadresse der gepackten Variablen (beim CATALOG-Befehl nicht ersichtlich) sowie im ENDFILE-Feld die Länge des Speicherauszugs (bis HIMEM = Endadresse) + 5 Bytes für den Kopf. Denn der VAR-File hat einen 5 Bytes umfassenden „Header", der sich zu Beginn des ersten Datenblocks befindet:
$00-$01: LLHH der Länge der einfachen und dimensionierten Variablen.
$02-$03: LLHH der Länge der einfachen Variablen allein.
$04: High Byte von HIMEM (= $0074) zu dem Zeitpunkt, als der VAR-File gespeichert wurde.
Die nachfolgenden Miniprogramme verdeutlichen die Anwendung von STORE und RESTORE.

HAUPTVARIABLEN

```
100 PRINT "HAUPTVARIABLEN"
110 DIM A(1000),A%(1000),A$(1000)
120 FOR X = 1 TO 1000:A(X) = X:A%(X) = X:A$(X) = STR$ (X): NEXT
130 FOR X = 1 TO 1000 STEP 100:A$(X) = "": NEXT
140 FOR X = 1 TO 1000: PRINT A(X),A%(X),A$(X): NEXT
150 PRINT CHR$ (4)"STORE VARIABLEN"
160 PRINT CHR$ (4)"RUN UNTERVARIABLEN"
```

UNTERVARIABLEN

```
100 PRINT "UNTERVARIABLEN"
110 PRINT CHR$ (4)"RESTORE VARIABLEN"
120 FOR X = 1 TO 1000: PRINT A(X),A%(X),A$(X): NEXT
```

CHAIN, STORE und RESTORE können auch im Direkt-Modus eingegeben werden. Man beachte jedoch, daß nach Ctrl-Reset sowie nach bestimmten BASIC.SYSTEM-Fehlermeldungen alle Variablen gelöscht werden, so daß dann weder CHAIN noch STORE möglich sind.

FRE ist ein Garbage-Collection-Befehl zur Entfernung nicht mehr benötigter Strings, der sowohl BASIC.SYSTEM-intern (u.a. vor CHAIN, STORE und OPEN) automatisch ausgeführt wird als auch im RUN- oder Direkt-Modus bewußt erteilt werden kann, z.B.
1000 PRINT CHR$ (4) "FRE": REM statt F = FRE (0)
Der BASIC.SYSTEM-FRE-Befehl ist bis zu 1000mal schneller als der Applesoft-Interpreter-FRE-Befehl, so daß letzterer möglichst nicht mehr benutzt werden sollte. Das Miniprogramm
10 FOR X = 1 TO 50000: A$ = CHR$ (7): NEXT: CALL -151
zeigt nach
9600.99FF
im Monitor, daß *automatisch* eine Garbage-Collection vorgenommen worden ist, da der I/O-Puffer jetzt „07 07..." enthält als temporärer Arbeitsspeicher für den FRE-Befehl. Der automatische FRE-Befehl findet dann statt, wenn der freie String-Pool nur noch ca. 4 Pages umfaßt.

Die nützlichen Befehle CHAIN, STORE, RESTORE und FRE gab es unter DOS 3.3 nicht. Sie haben jedoch gleichzeitig den Nachteil, daß sie sich nicht mit compilierten Applesoft-Programmen vertragen.

1.13. BRUN, BLOAD, BSAVE

BRUN /VOL/DIR/ASM.PROGRAMM, Aa, Ll, Bb, Tt, Ss, Dd
BRUN /VOL/DIR/ASM.PROGRAMM, Aa, Ee, Bb, Tt, Ss, Dd
BLOAD /VOL/DIR/FILE, Aa, Ll, Bb, Tt, Ss, Dd
BLOAD /VOL/DIR/FILE, Aa, Ee, Bb, Tt, Ss, Dd
BSAVE /VOL/DIR/FILE, Aa, Ll, Bb, Tt, Ss, Dd
BSAVE /VOL/DIR/FILE, Aa, Ee, Bb, Tt, Ss, Dd

Bei diesen Befehlen sind einige zusätzliche Parameter außer den bereits bekannten Ss und Dd möglich, die wir einzeln erläutern wollen.

BLOAD (binary load = binäres Laden) dient zum Laden eines Binärfiles oder jedes beliebigen anderen Files, falls zusätzlich der Typ-Parameter TTXT, TBAS usw. angegeben wird.
BSAVE (binary save = binäres Speichern) dient zum Speichern von Speicherauszügen als Binärfiles oder beliebiger anderer Files, falls zusätzlich der Typ-Parameter TTXT, TBAS usw. angegeben wird. Im letzteren Fall muß zunächst ein CREATE ausgeführt werden.
BRUN (binary run = binärer RUN-Befehl) dient zum Laden *und* Starten eines Assemblerprogramms, das nicht nur ein BIN-, sondern auch beispielsweise ein SYS- oder REL-File sein kann.

Ein Binärfile im engeren Sinne, also ein BIN-File, ist ein Speicherauszug. Die aufgrund der sog. System Bit Map zulässigen bzw. empfohlenen Speicherbereiche sind:
$0300-$03CF
$0800-$95FF
Damit sind folgende Bereiche ausgeklammert:
$0000-$0100 Zero-Page
$0100-$01FF Stack
$0200-$02FF Eingabe-Puffer
$0400-$07FF Bildschirmspeicher
$9600-$99FF I/O-Puffer
$9A00-$BEFF BASIC.SYSTEM
$BF00-$BFFF PRODOS Global Page
$D000-$FFFF Language Card
Zwar wäre es möglich, z.B. den Bereich $0200-$02FF mit BSAVE abzuspeichern, doch könnte man später diesen BIN-File nicht mehr mit BLOAD nach $0200 zurückladen.
Unter *Speicherauszug* verstehen wir einen zusammenhängen Speicherbereich, z.B. den Speicherraum $2000 bis $3FFF (= HGR Seite 1). Hier gilt:

A = Anfangsadresse = $2000 = 8192
E = Endadresse = $3FFF = 16383
L = Länge = $2000 = 8192
Wenn zwei Größen vorgegeben sind, errechnet sich die dritte Größe folgendermaßen:

L = E - A + 1
E = A + L - 1
A = E - L + 1

Betrachten wir zu diesem Zweck den Speicherbereich $0300-$0303:

$0300 = 768 - 0. - $41 = A
$0301 = 769 - 1. - $42 = B
$0302 = 770 - 2. - $43 = C
$0303 = 771 - 3. - $44 = D

Diesen Speicherauszug könnte man speichern mit
BSAVE BINFILE, A$0300, L$0004 oder A768, L4
BSAVE BINFILE, A$0300, E$0303 oder A768, E771
Anfangsadresse, Endadresse und Länge können als Hexadezimalzahlen (mit vorangestelltem „$") oder als Dezimalzahlen angegeben werden. Die A-, E- und L-Parameter könnten Werte im Bereich $0000-$FFFF (= 0-65535) annehmen, doch sind die o.g. Begrenzungen aufgrund der System Bit Map zu beachten.
Die Datenblocks des BIN-Files spiegeln exakt den Speicherauszug wider. Im Directory-Eintrag findet man im SUBTYPE-Feld die Anfangsadresse und im ENDFILE-Feld die Länge. Die einzelnen Bytes eines BIN-Files werden von 0 bis n numeriert, und zwar jeweils *absolut* zum nullten Byte als Bezugspunkt. Hier gibt es einen weiteren Parameter, nämlich B als Byte-Offset, der im Bereich $000000-$FFFFFF (= 0-16777215) liegen kann (= 16M). Nehmen wir zu diesem Zweck an, daß die Speicherstellen $0300-$0303 und später der BINFILE die Buchstaben „ABCD" enthalten. Dann gilt:

BLOAD BINFILE
lädt, da alle übrigen Parameter fehlen, die komplette Datei BINFILE mit dem Inhalt „ABCD" wieder in den alten Speicherbereich $0300-$0303. Wenn also die A- und L-Parameter fehlen, werden die entsprechenden Ersatzwerte aus dem Directory-Eintrag übernommen.
BLOAD BINFILE, A$1000
lädt die komplette Datei BINFILE „ABCD" in den neuen Speicherbereich $1000-$1003.
BLOAD BINFILE, A$2000, L$0002

lädt nur die ersten zwei Bytes „AB" von BINFILE in den neuen Speicherbereich $2000-$2001.
BLOAD BINFILE, A$3000, E$3002
lädt nur die ersten drei Bytes „ABC" von BINFILE in den neuen Speicherbereich $3000-$3002.
BLOAD BINFILE, A$4000, B$0002, L$0001
lädt nur das Byte „C" in die Speicherstelle $4000. Der B-Parameter $0002 steht für das 2. Byte. Da die Zählung jedoch mit dem 0. Byte beginnt, ist effektiv das dritte Byte, also der Buchstabe „C", gemeint. Da als Länge nur 1 angegeben wird, wird auch nur 1 Byte geladen.
BLOAD BINFILE, A$5000, E$5001, B$0001
lädt ab dem 1. Byte von BINFILE zwei Bytes, d.h. „BC" in den neuen Bereich $5000-$5001. Daß zwei Bytes geladen werden sollen, „weiß" das BASIC.SYSTEM aus der Differenz zwischen E - A (+ 1!). Logischerweise ergibt sich, daß *entweder* der E-Parameter *oder* der L-Parameter angegeben werden muß (also *nicht* beide gleichzeitig!).

Der BSAVE/BLOAD-Befehl kann um den T-Parameter ergänzt werden. Fehlt der T-Parameter, so ist stets eine BIN-Datei gemeint. Mit den Befehlen

CREATE FILE.XYZ, Tt
BSAVE FILE.XYZ, Tt, Aa usw.
BLOAD FILE.XYZ, Tt, Aa usw.

kann man Dateien beliebigen Typs erzeugen und darin Speicherauszüge ablegen oder von dort laden. Beispiele:

BLOAD PRODOS, TSYS, A$1000
lädt das Betriebssystem PRODOS in den Speicher ab $1000. Man beachte, daß hier sowohl der Parameter TSYS als auch eine (beliebige) Anfangsadresse erforderlich ist, denn der BLOAD-Befehl „weiß" bei Nicht-TBIN-Dateien nichts von der Anfangsadresse.
BLOAD TEXTE, TTXT, B1000, L2000, A10000
lädt von dem Textfile TEXTE ab dem 1000. Byte insgesamt 2000 Bytes in den RAM-Bereich ab der Speicherstelle 10000.
BLOAD BASIC.PROGRAMM, TBAS, A$2000
lädt ein Applesoft-Programm in den Speicher ab $2000.

CREATE BILDER, T$F1
BSAVE BILDER, T$F1, A$2000, L$4000
BLOAD BILDER, T$F1, B$0000, L$2000, A$2000
BLOAD BILDER, T$F1, B$2000, L$2000, A$2000

erzeugt zunächst eine Datei mit dem neuen T-Typ $F1, speichert dann den Inhalt der 2 HGR-Seiten ab und lädt im Anschluß daran das erste Bild in den HGR1-Bereich und danach das zweite Bild *ebenfalls* in den HGR1-Bereich. Man kann also in eine einzelne Datei mehrere Bilder packen. Der T$F1-Parameter wurde hier genommen, um zu zeigen, wie man neue Dateitypen erzeugen kann. Für Hires-Bilder wäre es ansonsten nicht erforderlich, eigens einen neuen Typ zu kreieren.

Erst bei der neuen BASIC.SYSTEM-Version ist es möglich, *über* den ENDFILE *hinaus* mit BSAVE eine Datei zu erweitern, z.B.
BSAVE XXX, A5000, L1000, B0000
BSAVE XXX, A5000, L1000, B1000
BSAVE XXX, A5000, L1000, B2000 usw.
Genauer gesagt konnte bei den Altversionen der B-Wert nie größer als der L-Wert sein. Dieser beträgt theoretisch $C000, z.B.
BSAVE XXX, A$0000, E$BFFF
Wer trotzdem monströse BSAVE-Dateien benötigt und 1.1.1 nicht besitzt, kann zu folgendem Trick-Programm namens MONSTER.MAKER greifen:

```
100  PRINT  CHR$ (4);"CAT": REM "MONSTER.MAKER"
110  INPUT "Anzahl der Blocks: "; BL
120  INPUT "Datei-Typ: "; T$
130  INPUT "Datei-Name: "; N$
140  PRINT  CHR$ (4); "CREATE"; N$; ",T" ;T$
150  PRINT  CHR$ (4); "OPEN"; N$; ",T"; T$; ",L512": REM 1 Block
160  FOR X = 0 TO BL
170  PRINT  CHR$ (4); "WRITE"; N$; ",R"; X
180  PRINT : NEXT X
190  PRINT  CHR$ (4); "CLOSE"
```

Zulässig ist jeder beliebige Dateityp, z.B. BIN, TXT, BAS usw., wobei der BIN-Typ empfohlen wird. MONSTER.MAKER schreibt in jeden Block 1 Return. Dies ist wichtig, damit alle später benötigten Blocks tatsächlich im Index-Block vermerkt werden. Die somit erzeugte Datei hat eine Größe von BL * 512 + 1, z.B. bei 100 Blocks 51201 Bytes. Das letzte Byte wird zwar später nicht gebraucht, ist jedoch erforderlich, damit der ENDFILE bis zum letzten Byte des (vor)letzten Blocks geht. Im Directory-Eintrag wird übrigens bei einer Monster-BIN-Datei A$0000 als Anfangsadresse im SUBTYPE-Feld vermerkt. Auf die BIN-Datei kann nunmehr nach Belieben mit
BLOAD DATEINAME, B$HHMMLL, A$HHLL, L$HHLL
und bei Version 1.1.1 auch mit
BSAVE DATEINAME, B$HHMMLL, A$HHLL, L$HHLL

zugegriffen werden. (Bei Version 1.1.1 wird nämlich eine BIN-Datei nicht mehr wie früher gekürzt, wenn vor dem Zugriff B + L < = ENDFILE war).

BRUN führt zunächst einen BLOAD aus und startet dann das Assemblerprogramm an der spezifizierten Anfangsadresse. Wie sich aus den vorangehenden Erörterungen ergibt, ist es möglich, *mehrere* Assemblerprogramme in eine einzige BIN-Datei zu packen (wie dies z.B. bei Appleworks geschieht). Leider kann man einer BIN-Datei nicht ansehen, ob es sich um ein BRUN-fähiges Maschinenprogramm oder beispielsweise um ein Hires-Bild handelt. Ich hätte dies durch einen gesonderten Dateityp, z.B. „ASM", kenntlich gemacht. Fremde BIN-Dateien sollte man deshalb nicht unbedacht mit BRUN starten, denn es könnte sich dahinter eine Datendatei verbergen.
Assemblerprogramme können auch aus einem Applesoft-Programm heraus gestartet werden, z.B.
1000 PRINT CHR$ (4); "BRUN UTILITY.XYZ"
1010 PRINT "Zurück vom Assemblerprogramm"
Wenn das Assemblerprogramm nicht den Stack manipuliert und mit RTS verlassen wird, so kehrt man danach zum Applesoft-Programm zurück. Alternativ kann man ein Assemblerprogramm mit
BLOAD NAME
einlesen und mit
CALL Anfangsadresse
aufrufen. Auch die BLOAD- und BSAVE-Befehle können selbstverständlich aus einem Applesoft-Programm heraus benutzt werden.

1.14. Strichbefehl, BYE

– /VOL/DIR/PROGRAMM, Ss, Dd
BYE (ohne Parameter)

Der **Strichbefehl** (im englischen ProDOS-Handbuch „Dash"- oder „Smart RUN"-Befehl genannt) dient zum Starten von
a) Applesoft-Programmen (BAS-Files)
b) Assemblerprogrammen (BIN-Files)
c) System-Programmen (SYS-Files)
d) EXEC-Programmen (TXT-Files)
f) sonstigen lauffähigen Programmen (REL-Files usw.)
Mit Strich ist der kombinierte Binde- und Gedankenstrich gemeint. Beispiele:

-PRODOS (statt BRUN PRODOS, TSYS, A$2000)
-STARTUP (statt RUN STARTUP)
-LIST.MAKER (statt EXEC LIST.MAKER)

BYE startet das in der Language Card befindliche Reboot-Programm, das im Band 1 beschrieben wurde. BYE ist nur ab der BASIC.SYSTEM-Version 1.1 (in Verbindung mit PRODOS 1.1.1) implementiert. Ersatzweise kann man bei älteren Versionen die bereits in Kapitel 1.4 genannten Pokes verwenden, also

```
1000 POKE 768, 32: POKE 769, 0: POKE 770, 191
1010 POKE 771, 101: POKE 772, 6: POKE 773, 3: POKE 774, 4
1020 CALL 768
```

1.15. EXEC

EXEC /VOL/DIR/BEFEHLSDATEI, Ff, Ss, Dd
EXEC /VOL/DIR/BEFEHLSDATEI, Rr, Ss, Dd

Im Direkt-Modus gibt man Applesoft- und BASIC.SYSTEM-Befehle über die Tastatur ein, z.B.
CATALOG
LIST
RUN PROGRAMM1
BLOAD BILD, A$2000
usw.
Man stelle sich nunmehr einen Textfile vor (TXT-Typ), der beispielsweise die o.g. Befehle als Buchstabenfolgen enthält, jeweils getrennt durch Returns, d.h. also letztlich exakt so, wie man sie normalerweise eintippen würde. Eine solcher Textfile wird EXEC-Datei (to execute = [Befehle] ausführen) genannt. Startet man eine solche Datei mit
EXEC BEFEHLSDATEI
oder aus dem Programm heraus mit
1000 PRINT CHR$ (4); "EXEC BEFEHLSDATEI"
so werden die einzelnen Befehle genau so ausgeführt, wie wenn man sie über die Tastatur eingegeben hätte. Der BASIC.SYSTEM-EXEC-Befehl unterscheidet sich von dem entsprechenden DOS-3.3-EXEC-Befehl in zwei Punkten:
1. BASIC.SYSTEM-EXEC-Programme lassen sich (meist) mit Ctrl-C unterbrechen. Dies gilt zumindest für mit EXEC gestartete Applesoft-Programme.

2. BASIC.SYSTEM-EXEC-Programme können (meist) keine Monitorbefehle wie
CALL -151
300L
usw. korrekt verarbeiten (wegen Zero-Page-Konflikten).
EXEC-Files erstelle man zweckmäßigerweise mit einem Textverarbeitungsprogramm (z.B. Applewriter o.ä.).

LIST.MAKER

Diese Utility, die sich auf der Begleitdiskette zu diesem Band befindet, wird mit
EXEC LIST.MAKER
gestartet, nachdem zuvor ein Applesoft-Programm geladen worden ist, das nunmehr mit Hilfe von LIST.MAKER in einen Textfile namens LIST verwandelt wird. Diesen Textfile kann man dann in ein Textverarbeitungsprogramm einlesen, um beispielweise Variablennamen zu ändern usw. Später kann man den dergestalt manipulierten Textfile wieder mit
EXEC LIST
in ein normales Applesoft-Programm zurückverwandeln.

LIST.MAKER

```
POKE  768,44: POKE 769,16: POKE 770,192: POKE 771,173:
      POKE 772,0: POKE 773,192: POKE 774,16: POKE 775,251:
      POKE 776,44: POKE 777,16: POKE 778,192: POKE 779,96
63999 POKE 33,33: PRINT: PRINT CHR$(4) "OPEN LIST, S6, D1":
      PRINT CHR$(4) "WRITE LIST": LIST -63998:
      PRINT CHR$(4) "CLOSE LIST": END
HOME: INVERSE: PRINT "LIST.MAKER": NORMAL: PRINT:
      PRINT "DISK IN S6, D1!": CALL 768: RUN 63999
DEL 63999, 63999
TEXT: PRINT "ERLEDIGT!"
```

DATA.MAKER

Diese Utility verwandelt ein Maschinenprogramm in eine Folge von DATA-Statements, die ihrerseits als kleines Applesoft-Programm namens DATA auf der Diskette gespeichert werden. Man braucht hierzu lediglich den Namen des zu konvertierenden Maschinenprogramms einzugeben. Anfangs- und Endadresse werden automatisch ermittelt.

Beide Utilities setzen Kenntnisse der Textfile-Befehle voraus, die im übernächsten Teilabschnitt erläutert werden.

DATA.MAKER

```
1000 TEXT : HOME : INVERSE : PRINT "DATA.MAKER": NORMAL
1010 PRINT : PRINT : INPUT "DISKETTE EINLEGEN ";X$:
     PRINT CHR$ (4);"PREFIX": INPUT P$: PRINT CHR$ (4)"PREFIX";P$:
     PRINT CHR$ (4);"CAT";P$
1020 INPUT "BINAERFILE: ";S$: GOSUB 1290
1030 REM Adresse + Länge ermitteln
1040 IF F = 0 THEN 1000
1050 A$ = MID$ (D$,75,5):L = VAL ( MID$ (D$,68,4))
1060 IF LEFT$ (A$,1) < > "$" THEN PRINT "KEIN BINAERFILE ";:
     GET X$: GOTO 1000
1070 PRINT CHR$ (4);"BLOAD";P$;S$;",A";A$;",L";L
1080 A = PEEK (48728) + PEEK (48729) * 256: REM Startadresse A
1090 E = A + L - 1
1100 HOME : PRINT "BITTE WARTEN..."
1110 REM EXEC-File DATA anlegen
1120 F$ = P$ + "DATA"
1130 PRINT CHR$ (4);"OPEN";F$: PRINT CHR$ (4);"CLOSE";F$:
     PRINT CHR$ (4);"DELETE";F$: PRINT CHR$ (4);"OPEN";F$:
     PRINT CHR$ (4);"WRITE";F$
1140 REM 10 DATA pro Zeile ab Zeile 10
1150 N = 10:C = 0: PRINT N;"DATA";
1160 FOR X = A TO E
1170 PRINT PEEK (X);:C = C + 1: IF X = E THEN 1200
1180 IF C < 10 THEN PRINT ",";: GOTO 1200
1190 C = 0: PRINT :N = N + 1: PRINT N;"DATA";
1200 NEXT X
1210 N = N + 1: PRINT : PRINT N;"RESTORE: FOR X =";A;"TO";E;":
     READ Y:POKE X,Y:NEXT"
1220 PRINT "DEL 1000, 2000"
1230 PRINT "SAVE";P$;"POKES"
1240 PRINT "LIST"
1250 PRINT CHR$ (4);"CLOSE";F$
1260 PRINT CHR$ (4);"EXEC";F$
1270 END
1280 REM Binärfile suchen
1290 PRINT CHR$ (4);"OPEN";P$;",TDIR"
1300 PRINT CHR$ (4);"READ";P$
1310 LL = LEN (S$)
1320 INPUT D$: INPUT D$: INPUT D$: REM Kopf ignorieren!
1330 F = 0: INPUT D$: IF MID$ (D$,2,LL) = S$ AND
     MID$ (D$,LL + 2,1) = " " THEN F = 1: GOTO 1350: REM F=Such-Flag
1340 IF D$ < > "" THEN 1330
1350 PRINT CHR$ (4);"CLOSE";P$
1360 IF F = 0 THEN PRINT P$;S$" FEHLT ";: GET X$
1370 RETURN
```

1.16. PR# und IN#

PR#s, Aa
PR# Aa
IN#s
IN# Aa

Diese beiden Befehle dienen zum Anschluß peripherer Ein/Ausgabe-Geräte. Beispiele:

PR#1
schaltet den Drucker in Slot 1 ein.
PR#0
schaltet den Drucker wieder ab bzw. den 40-Zeichen-Bildschirm ein.
PR#3
schaltet die 80-Zeichenkarte in Slot 3 ein.
IN#2
macht ein Modem in Slot 2 empfangsbereit.
PR#6
bootet von dem Laufwerk im Slot 6.

In jedem Fall ist es erforderlich, daß die entsprechenden Peripherie-Geräte an den entsprechenden Slots angeschlossen sind, weil sonst ProDOS „abstürzen" kann. Man beachte nämlich, daß ProDOS während des Boot- und Installationsvorgangs nicht alle Peripherie-Geräte korrekt erkennt. Beispielsweise „weiß" ProDOS nicht, ob an einem Controller nur 1 Drive oder 2 Drives angeschlossen sind. Ferner „denkt" ProDOS, daß eine de facto vorhandene Accelerator-Karte gar nicht existiert usw.

s in PR#s steht für eine Slot-Nummer im Bereich 1-7 (0 ist ebenfalls möglich und bedeutet bei PR#0 Bildschirm und bei IN#0 Tastatur). Da PR# und IN# ebenfalls Applesoft-Befehle sind, muß man in einem laufenden Applesoft-Programm stets Ctrl-D voranstellen, damit das BASIC.SYSTEM nicht „abgehängt" wird, also richtig
100 PRINT CHR$ (4); "PR#1"
und falsch
100 PR#1
Man kann diese Tatsache jedoch ausnutzen, um das BASIC.SYSTEM vorübergehend vollständig „abzuhängen", z.B. durch die Zeile
100 POKE 242, 0: PR#0: IN#0
Nach Ctrl-Reset oder CALL 39447 (Version 1.0.1 bis 1.1.1) ist das BASIC.SYSTEM dann „wieder da".

Ein Peripherie-Gerät hat normalerweise einen Treiber in der Interface-Karte, die in dem jeweiligen Slot steckt. Im Gegensatz zu DOS 3.3 ist es jetzt unter dem BASIC.SYSTEM

möglich, die Ein/Ausgabe auf eigene Treiberprogramme umzulenken. Damit sie als solche korrekt erkannt werden, müssen sie mit dem CLD-Befehl ($D8) beginnen. Beispielsweise würde

PR# $0300

die *momentane* Ausgabe (z.B. Bildschirm, Drucker usw.) auf einen eigenen Treiber umlenken, der bei $0300 mit CLD anfangen muß. Man beachte im übrigen die abweichende Parameter-Syntax bei PR# und IN#.

Auf die Darstellung spezieller Treiberprogramme muß an dieser Stelle verzichtet werden, da sie nicht zum eigentlichen Thema gehören.

1.17. OPEN, READ, WRITE, CLOSE, FLUSH, APPEND, POSITION

OPEN /VOL/DIR/FILE, Tt, Ll, Ss, Dd
READ /VOL/DIR/FILE, Rr, Ff, Bb
WRITE /VOL/DIR/FILE, Rr, Ff, Bb
CLOSE /VOL/DIR/FILE
FLUSH /VOL/DIR/FILE
APPEND /VOL/DIR/FILE, Tt, Ll, Ss, Dd
POSITION /VOL/DIR/FILE, Ff oder
POSITION /VOL/DIR/FILE, Rr

Diese Befehle dienen unter dem BASIC.SYSTEM im wesentlichen zum Bearbeiten von Textfiles = Textdateien = TXT-Files = ASCII-Files, weil bei anderen Dateitypen keine geeigneten PRINT-INPUT-Befehle existieren (Ausnahme: DIR-Files). Wenn der T-Typ fehlt, sind automatisch TXT-Files gemeint. Bevor wir auf die Befehle im einzelnen eingehen, sind einige allgemeine Erörterungen erforderlich. Ferner sei für DOS-3.3-Kenner angemerkt, daß wir nicht auf durch Kommas abgegrenzte *Elemente* = Feldteile eingehen werden, da letztere zu einer uneffizienten Programmiertechnik führen. Näheres hierzu findet man in meinem Buch „Apple DOS 3.3".

Ein *Textfile* ist für das BASIC.SYSTEM eine Folge von ASCII-Zeichen (mit Bit 7 off, siehe Tabelle S. 202), z.B. Strings (= Zeichenketten, Zeichenfolgen) oder Zahlen (im ASCII-Format als Zahlen-Strings), z.B.

A $41
B $42
C $43
r $0D = Buchstaben-String „ABC" + Return

1 $31
2 $32
3 $33
r $0D = Zahlen-String „123" + Return

Eingelesene Textfiles werden im Monitor als eine Folge von Bytes (= 2stellige Hexzahlen) dargestellt und vom Applesoft-Programm als Buchstaben- oder Ziffernfolgen interpretiert. Zu beachten ist, daß „normale" Strings im Variablenspeicher des Applesoft-Programms tatsächlich als ASCII-Zeichen = Hexzahlen im Bereich $01-$7F bzw. 1-127 abgelegt werden, während Fließkommazahlen als gepackte 5-Byte-Folgen im Variablenspeicher verschlüsselt sind. Für Textfiles sind jedoch Zahlen stets „spezielle" Strings, die nur Zahlzeichen enthalten („0" ... „9", „-", „+", „." und „E"). Man beachte ferner, daß unter DOS 3.3 Textfiles mit Bit 7 on (Hexzahlenbereich $81-$FF) gespeichert wurden, während für ProDOS Bit 7 off gilt ($01-$7F). Das Mischen von Bit-7-on- und Bit-7-off-ASCII-Zeichen ist nicht ohne weiteres möglich.

Sequentieller Textfile

Eine Folge von ASCII-Zeichen ist zunächst eine unstrukturierte „Soße". Deshalb lassen sich des näheren (a) Felder und (b) Records unterscheiden:
Ein *Feld* ist eine Zeichenfolge, die durch Return = Ctrl-M = $0D (symbolisiert durch „r") abgeschlossen wird. Wenn in einer Textdatei Felder mit *unterschiedlicher* Länge aufeinanderfolgen, so spricht man von *sequentieller* Datei mit indirektem Zugriff. Beispiel:

```
Feld 0                Feld 1          Feld 2              Feld-Nummer
0  1  2  3  4  5      0  1  2  3      0  1  2  3  4       Feld-Byte-Nummer
W  A  L  T  E  R  r   K  A  R  L  r   A  D  O  L  F  r   ASCII-Folge (TXT-file)
0  1  2  3  4  5  6  7  8  9  10 11  12 13 14 15 16 17  Byte-Nr. (ENDFILE = 18)
```

„WALTER" umfaßt 6 Zeichen + r
„KARL" umfaßt 4 Zeichen + r
„ADOLF" umfaßt 5 Zeichen + r

Insgesamt umfaßt die Datei 18 Zeichen (ENDFILE = 18), numeriert von 0 bis 17. Das Return dient als Feldbegrenzer. Man spricht von sequentiellem oder indirektem Zugriff, weil wegen der unterschiedlichen Länge der Felder kein (mathematisch) berechneter direkter Zugriff auf ein beliebiges Feld f erfolgen kann. Dem obigen Diagramm ist zwar zu

entnehmen, daß Feld 2 ab der absoluten Byte-Position 12 beginnt, doch weiß man dies nur, weil wir hier die einzelnen Feldlängen bereits ermittelt haben.
Betrachten wir nunmehr die beiden nachfolgenden Miniprogramme:

WRITE

```
10 PRINT CHR$ (4); "OPEN DATEI"
20 PRINT CHR$ (4); "WRITE DATEI"
30 W$ = "WALTER": PRINT W$
40 K$ = "KARL": PRINT K$
50 A$ = "ADOLF": PRINT A$
60 PRINT CHR$ (4); "CLOSE DATEI"
```

READ

```
10 PRINT CHR$ (4); "OPEN DATEI"
20 PRINT CHR$ (4); "READ DATEI"
30 INPUT W$
40 INPUT K$
50 INPUT A$
60 PRINT CHR$ (4); "CLOSE DATEI"
```

Auf eine Textdatei wird geschrieben durch die Befehlsfolge
OPEN (Textdatei eröffnen)
WRITE (Schreibvorgang einleiten)
PRINT (tatsächlich schreiben)
und es wird von einer Textdatei gelesen durch die Befehlsfolge
OPEN (Textdatei eröffnen)
READ (Lesevorgang einleiten)
INPUT (tatsächlich lesen)

Bevor wir diese Miniprogramme näher analysieren, zunächst einige Worte zum Begriff **I/O-Puffer**. Wie bereits erwähnt, befindet sich im Bereich $9600-$99FF, d.h. unmittelbar nach HIMEM (= $9600), ein allgemeiner I/O-Puffer. Dieser wird benutzt für Befehle wie CATALOG, BLOAD, BSAVE, LOAD, SAVE usw. Der wesentliche Unterschied zwischen z.B. einem BLOAD und einem OPEN-READ ist der, daß BLOAD stets „in einem Zug" erfolgt, d.h. der I/O-Puffer wird nur während des LOAD-Vorgangs benötigt und ist danach sofort wieder für andere Zwecke wie CATALOG usw. frei. Anders beim OPEN-READ-Befehl: Solange eine mit OPEN eröffnete Textdatei noch nicht mit CLOSE geschlossen worden ist, bleibt der hierfür reservierte I/O-Puffer belegt, denn ein Textfile wird (normalerweise) nicht „in einem Zuge" gelesen. Im Gegensatz zu DOS 3.3

können unter ProDOS nur maximal 8 I/O-Puffer angelegt werden. Jeder I/O-Puffer umfaßt 1024 Bytes (= $0400 Bytes = 2 Blocks). Die ersten 512 Bytes enthalten den momentanen Daten-Block und die zweiten 512 Bytes den momentanen Index-Block (= Verzeichnis der von der Datei belegten Blocks). Nach der Folge OPEN-WRITE bewirkt der PRINT-Befehl nicht eine Ausgabe des Datenfeldes an den Bildschirm, sondern das jeweilige Feld wird in den Textfile-I/O-Puffer übertragen. Wenn sich 512 Bytes angesammelt haben, werden die momentanen Ausgabefelder als Daten-Block auf die Diskette übertragen. Analog bewirkt nach der Folge OPEN-READ der INPUT-Befehl nicht eine Eingabe eines Datenfeldes über die Tastatur, sondern das Eingabefeld wird aus dem Textfile-I/O-Puffer in den Variablenspeicher übertragen, nachdem zuvor der benötigte Daten-Block eingelesen worden ist. Das **Diagramm** (s. S. 63) veranschaulicht in vereinfachter Form diesen Vorgang.

Neben dem I/O-Puffer spielt der sog. **(relative) Positionszeiger** bei Textfiles eine wichtige Rolle. Nach OPEN wird der Zeiger zunächst auf das nullte Byte (bzw. den Anfang des nullten Feldes) der Textdatei gesetzt, d.h. auf das „W" von „WALTER". Folgt nun ein READ-INPUT-W$, so wird „WALTER" eingelesen und der Zeiger auf das „K" von „KARL" gesetzt usw. Beim READ-Vorgang zeigt also der Zeiger immer auf die Position des *als nächstes* einzulesenden Feldes: zunächst Feld 0, dann Feld 1 usw. bis zum Feld n. Wenn das letzte Feld eingelesen worden ist, weist der Zeiger auf das Byte, das dem Ende der Datei folgen würde, wenn es dort stände. Im Gegensatz zu DOS 3.3 werden jedoch unter ProDOS Textfiles nicht durch einen sog. Endmarker = Hexzahl $00 abgeschlossen. Vielmehr ergibt sich das Dateiende hier stets aus ENDFILE, der bei unserer Minidatei den Wert 18 hat. Da die Numerierung der Bytes eines Textfiles mit 0 beginnt, aber andererseits das 0. Byte bei „normaler" Zählweise eigentlich das 1. Byte darstellt, hat **ENDFILE** zwei Bedeutungen:
– Erstens bedeutet ENDFILE bei einer Zählung von 1 aufwärts die Gesamtzahl aller Bytes (bzw. Zeichen) eines Textfiles.
– Zweitens bedeutet ENDFILE bei einer Zählung von 0 aufwärts die Position des Bytes, das dem physisch letzten Byte folgt bzw. folgen würde.
Dies wird deutlicher, wenn wir nunmehr den Schreibvorgang betrachten. Nach OPEN wird der Zeiger zunächst auf das nullte Byte bzw. den Anfang des nullten Feldes der Textdatei gesetzt. Der erste PRINT (von „WALTER") beginnt damit beim nullten Byte. Danach weist der Zeiger auf den Anfang des potentiellen nächsten Feldes (Feld 1 für „KARL") usw. Wäre also unsere Testdatei noch niemals beschrieben worden, so würde vor jedem PRINT der Zeiger jeweils auf dasjenige Byte zeigen, das dem momentanen ENDFILE (= End of File = Ende der Datei) entspricht.

Mit den Parametern F (für Feld-Offset) und B (für Byte-Offset) kann man den momentanen Positionszeiger *vor*rücken (d.h. nur vorwärts, *niemals* rückwärts!).

F0 ist das momentane Feld, F1 das nächste, F2 das übernächste usw.
B0 ist das momentane Byte, d.h. Byte 0 des momentanen Feldes, B1 ist das nächste Byte des momentanen Feldes, d.h. Byte 1 usw. Betrachten wir hierzu unser „WALTERr-KARLrADOLFr"-Beispiel:

```
10 PRINT CHR$ (4); "OPEN DATEI"
20 PRINT CHR$ (4); "READ DATEI,F1,B2"
30 INPUT KR$
```

KR$ würde Feld 1 (= „KARL") ab dem Byte 2 bis zum Return, d.h. „RL" einlesen.
Man beachte, daß F und B als Parameter von READ und WRITE *relative* Positionszeiger sind, d.h. F und B errechnen sich immer ab der *momentanen* Position. Im Gegensatz dazu ist der R-Parameter (s.u.) ein *absoluter* Positionszeiger, der immer den absoluten Dateianfang (tatsächliches Byte 0 von Feld 0 der Gesamtdatei) als Bezugspunkt hat, also in unserem Beispiel die tatsächliche Position von Buchstabe „W" des ersten Wortes „WALTER". Unmittelbar nach dem OPEN einer Datei sind absolute und relative Position identisch, danach nicht mehr. Man merke sich ferner, daß der B-Paramter bei READ und WRITE stets die relative Byte-Position meint, während umgekehrt der B-Parameter bei BLOAD und BSAVE stets die absolute Byte-Position meint.

Random-Textfile

Während sequentielle Textfiles durch Felder unterschiedlicher Länge charakterisiert sind, haben Random-Textfiles (Random-Access-Textfiles = Textdateien mit direktem Zugriff) entweder Felder oder Feldgruppen (= Datensätze, Records) mit *gleicher Länge*. Solche Random-Textfiles werden deshalb mit einem L-Parameter eröffnet. Beispiel:

```
Record 0            Record 1            Record 2
W  A  L  T  E  R  r  K  A  R  L  r  .  .  A  D  O  L  F  r  .  ASCII-Folge
0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 Byte-Nr.
Feld 0              Feld 1              Feld 2                 ENDFILE = 21
```

WRITE

```
10 PRINT CHR$ (4); "OPEN DATEI, L7"
20 PRINT CHR$ (4); "WRITE DATEI, R0": PRINT "WALTER"
30 PRINT CHR$ (4); "WRITE DATEI, R1": PRINT "KARL"
40 PRINT CHR$ (4); "WRITE DATEI, R2": PRINT "ADOLF"
50 PRINT CHR$ (4); "CLOSE DATEI"
```

READ

```
10 PRINT CHR$ (4); "OPEN DATEI, L7"
20 PRINT CHR$ (4); "READ DATEI, R2": INPUT A$
30 PRINT CHR$ (4); "READ DATEI, R1": INPUT K$
40 PRINT CHR$ (4); "READ DATEI, R0": INPUT W$
50 PRINT CHR$ (4); "CLOSE DATEI"
```

Beim OPEN-Befehl muß der L-Parameter (L = Länge) auch das Return berücksichtigen. Jeder nachfolgende READ oder WRITE muß mit einem R-Parameter (R = Record) versehen sein. Da die Records alle gleich lang sind, kann das BASIC.SYSTEM leicht den absoluten Positionszeiger berechnen. Deshalb muß die Random-Datei nicht Record für Record bzw. Feld für Feld aufsteigend gelesen werden. Vielmehr kann man wahlweise (= at random) beliebige Record-Nummern vorgeben. Wenn z.B. L = 7 und R = 2 vorgegeben sind, so errechnet sich die absolute Byte-Position durch 7 * 2 = 14. D.h. der absolute Positionszeiger wird auf das 14. Byte gesetzt und ab dieser Stelle wird entweder gelesen oder geschrieben. Theoretisch könnte man deshalb auf den R-Parameter verzichten, indem man sich den absoluten Byte-Offset selbst durch Multiplikation errechnet. Dies wäre jedoch nur bei dem etwas komplizierten BLOAD-BSAVE-Zugriff auf Textfiles möglich, weil nur hier der B-Parameter den absoluten Byte-Offset impliziert. Außerdem besteht ein typischer Record (= Datensatz = Summe der Felder) normalerweise nicht nur aus einem einzigen Feld, sondern aus mehreren Feldern, z.B. bei der Adreßverwaltung aus den Feldern Vorname, Zuname, Straße, PLZ, Ort usw. Nehmen wir hierzu der Einfachheit halber an, daß eine Datei angelegt werden soll, die jeweils 3 Vornamen in einen einzigen Record packt. Dann wäre bei unserem Beispiel die Zeichenkette „WALTERrKARLr..ADOLFr." ein einziger Record, nämlich Record 0, und „WALTERr", „KARLr.." und „ADOLFr." wären die Felder 0, 1 und 2 innerhalb dieses Records. Man hätte dann folgende allgemeine Dateistruktur:

R0 mit F0, F1, F2; R1 mit F0, F1, F2; ....Rn mit F0, F1, F2

Zusammenfassung

Nach dieser vereinfachten Einführung folgt nunmehr eine systematische Darstellung:

OPEN: Ein Textfile muß mit OPEN eröffnet werden, wodurch bis zum nächsten CLOSE ein I/O-Puffer bereitgestellt wird. Random-Textfiles werden mit dem L-Parameter eröffnet. Der L-Parameter ist später aus dem SUBTYPE-Feld des Dateieintrages im Directory als R=????? ersichtlich. (R = Recordlänge und *nicht* Recordnummer. Der Buchstabe „R" ist unglücklich gewählt.). Es gilt:
Wenn R = 0, dann sequentieller Textfile.
Wenn R > 0, dann Random-Textfile.

Es besteht auch die Möglichkeit, generell *Nicht*-TXT-Files (mit Typ-Zusatz) sowie speziell Directories (mit DIR-Typ-Zusatz) zu eröffnen. Doch nur für DIR-Dateien (Volume-Directory oder Subdirectory) wurde der normale READ-INPUT-Zugriff im BASIC.SYSTEM implementiert (siehe Kapitel 3.1). Directories können im übrigen nur gelesen und niemals beschrieben werden. Der bei OPEN verwendete Pfadname, z.B. „VOL/DIR/DATEI.1" muß in exakt derselben Form bei allen nachfolgenden READ-WRITE-Befehlen benutzt werden, selbst wenn zwischenzeitlich das Präfix geändert würde. Lediglich bei CLOSE und FLUSH kann der Pfadname gänzlich entfallen.

CLOSE und FLUSH: CLOSE ohne Pfadname schließt *alle* offenen Dateien, während CLOSE mit Pfadname nur diese Datei *allein* schließt und alle anderen Dateien weiterhin offenläßt. FLUSH überträgt nach (mehreren) WRITEs den Inhalt des I/O-Puffers auch dann auf die Diskette, wenn die Summe der Felder noch keinen kompletten, weiteren Daten-Block ausmacht. Ferner aktualisiert FLUSH den Index-Block und die Dateieintrag-Daten (wie Dateilänge usw.) FLUSH ist damit ein Vorsichtsmaßnahme gegen eventuellen Stromausfall und unvorhergesehenen Programmabbruch. Da FLUSH jedoch mehrere Sekunden dauern kann, muß man zwischen Datensicherheit und Geschwindigkeit abwägen. Nach FLUSH muß ein erneuter WRITE oder READ erfolgen. CLOSE und FLUSH können auch nach einem Programmabbruch im Direktmodus eingegeben werden, während OPEN/READ/WRITE/APPEND/POSITION nur im RUN-Modus zugelassen sind.

WRITE: Bei sequentiellen und Random-Dateien *kann* WRITE durch die Parameter F und B ergänzt werden, die dann den momentanen Positionszeiger um die Anzahl der Felder oder Bytes vorrücken. F und B sind relative Positionszeiger (außer unmittelbar nach OPEN und vor dem ersten READ/WRITE). Bei Random-Dateien *muß* der R-Parameter mit angegeben werden. R ist ein absoluter Positionszeiger, der intern (SET MARK) die absolute Byte-Position durch das Produkt R * L ermittelt. Der R-Parameter kann den absoluten Positionszeiger vorwärts und rückwärts rücken. Nach WRITE kann ein Feld mit PRINT Variablenname oder PRINT „Konstante" beschrieben werden, z.B.

```
100 PRINT 123: REM Zahlenkonstante
110 PRINT X: REM Zahlenvariable
120 PRINT "ABC": REM Stringkonstante
130 PRINT X$: REM Stringvariable
```

Ein Feld muß immer durch Return abgegrenzt werden. Andere Begrenzer werden vom BASIC.SYSTEM nicht anerkannt (wohl aber vom MLI). Bei Random-Files muß vor jedem PRINT ein WRITE, Rr erfolgen. Bei sequentiellen Files muß der WRITE nur dann erfolgen, wenn zwischenzeitlich der WRITE-Befehl durch PRINT-Ctrl-D allein desaktiviert wurde (siehe Kapitel 2.7). (Assemblerprogrammierer können in der Regel auch mit COUT auf die Datei schreiben.)

READ: Für READ gilt das gleiche wie für WRITE. Nach READ kann ein Feld mit INPUT Stringvariable oder INPUT Zahlenvariable eingelesen werden, z.B.
100 INPUT X: REM Zahlenvariable
110 INPUT X$: REM Stringvariable
Darüber hinaus können auch einzelne Zeichen eines Feldes oder gar der gesamten (sequentiellen) Datei mit GET eingelesen werden, doch ist dies extrem langsam. Felder, die Kommas oder Doppelpunkte enthalten, können nicht mit INPUT, aber dafür mit GET eingelesen werden. Eine bessere Lösung dieses Problems bietet Kapitel 2.5. (Assemblerprogrammierer können normalerweise nicht mit RDKEY von der Datei lesen, wohl aber in der Regel mit GETLN. Man beachte nämlich, daß im Gegensatz zu DOS 3.3 die Möglichkeit des Schreibens mit COUT und des Lesens mit RDKEY und GETLN im BASIC.SYSTEM nicht eigens vorgesehen wurde.)

APPEND: Mit APPEND wird erstens eine (sequentielle) Textdatei eröffnet, zweitens der Positionszeiger auf ENDFILE gesetzt und drittens WRITE vorbereitet, so daß nachfolgende PRINTs Daten an die Datei anhängen (to append = anhängen). Bei Random-Files sollte man Append durch OPEN RANDOM, Ll ... WRITE RANDOM, Rn+1 ersetzen, wobei n+1 den nächsten, noch nicht beschriebenen Record meint.

POSITION: Dieser Befehl kann wahlweise mit dem R- oder F-Parameter benutzt werden. In beidem Fällen, d.h. auch bei R, ist F = relativer Feld-Offset gemeint. POSITION ist wegen READ FILE, Ff bzw. WRITE FILE, Ff entbehrlich und nur aus Gründen der DOS-3.3-Kompatibilität implementiert worden.

EXEC: Dieser Befehl wurde bereits im Kapitel 1.15 besprochen. Ergänzend sei erwähnt, daß der wahlweise benutzbare R- oder F-Parameter im Gegensatz zum obigen POSITION-Befehl jeweils den *absoluten* Feld-Offset meint, denn praktisch kann der F-Parameter nur einmal erteilt werden.

Abschließend sei bemerkt, daß die Parameter Ss und Dd nur bei OPEN und APPEND zulässig sind (vgl Kapitel 2.4). Das nachfolgende TEXTDEMO veranschaulicht die Benutzung der wichtigsten Textfile-Befehle. Weitere Beispiele findet man im nächsten Kapitel.

TEXTDEMO

```
100 REM TEXTDEMO
110 REM Textfile mit Feldern
120 REM Feld-  0 aaaaaaaaaaR bis
130 REM Feld-100 aaaaaaaaaaR erzeugen
140 REM 12345678901234567890 = 20 Bytes
150 T$ = "TEXTFILE":N = 99
```

```
160 HOME
170 PRINT "Textfile anlegen"
180 PRINT CHR$ (4);"OPEN";T$
190 PRINT CHR$ (4);"WRITE";T$
200 FOR X = 0 TO N
210 PRINT "Feld-";
220 IF X < 010 THEN PRINT " ";
230 IF X < 100 THEN PRINT " ";
240 PRINT X;" ";
250 PRINT "aaaaaaaaaa"
260 NEXT X
270 PRINT CHR$ (4);"CLOSE"
280 PRINT
290 PRINT "Textfile normal sequentiell einlesen"
300 PRINT
310 PRINT CHR$ (4);"OPEN";T$
320 FOR X = 0 TO N
330 PRINT CHR$ (4);"READ";T$
340 INPUT X$: PRINT CHR$ (4): PRINT X$
350 NEXT
360 PRINT CHR$ (4);"CLOSE"
370 PRINT
380 PRINT "Textfile sequentiell mit F = 1,":
    PRINT "d.h. nur jedes zweite Feld lesen":
    PRINT "F1, F3, F5 ... Fn-1"
390 PRINT :F = 1
400 PRINT CHR$ (4);"OPEN";T$
410 PRINT CHR$ (4);"READ";T$;",F"0:
    REM Zeiger auf 0 setzen wegen Version-1.0.1/1.0.2-Bug!
420 FOR X = 0 TO INT (N / 2)
430 PRINT CHR$ (4);"READ";T$;",F"F
440 INPUT X$: PRINT CHR$ (4): PRINT X$
450 NEXT
460 PRINT CHR$ (4);"CLOSE"
470 PRINT
480 PRINT "Textfile sequentiell mit B = 4,":
    PRINT "d.h. 'Feld'-Wort überspringen"
490 PRINT :B = 4
500 PRINT CHR$ (4);"OPEN";T$
510 FOR X = 0 TO N
520 PRINT CHR$ (4);"READ";T$;",B"B
530 INPUT X$: PRINT CHR$ (4): PRINT X$
540 NEXT
550 PRINT CHR$ (4);"CLOSE"
560 PRINT
570 PRINT "Textfile als Random mit L = 20":
    PRINT "rückwärts von n bis 0 einlesen"
580 PRINT :L = 20
590 PRINT CHR$ (4);"OPEN";T$;",L"L
600 FOR X = N TO 0 STEP - 1
610 PRINT CHR$ (4);"READ";T$;",R"X
620 INPUT X$: PRINT CHR$ (4): PRINT X$
630 NEXT
640 PRINT CHR$ (4);"CLOSE"
```

```
650 PRINT
660 PRINT "Textfile als Random mit L = 40,":
    PRINT "d.h. 2 Feldern pro Record und":
    PRINT "mit F = 1 und B = 4 rückwärts lesen"
670 PRINT :L = 40:F = 1:B = 4
680 PRINT CHR$ (4);"OPEN";T$;",L"L
690 FOR X = INT (N / 2) TO 0 STEP - 1
700 PRINT CHR$ (4);"READ";T$;",R"X;",F"F;",B"B
710 INPUT X$: PRINT CHR$ (4): PRINT X$
720 NEXT
730 PRINT CHR$ (4);"CLOSE"
```

„Loch"-Dateien

Random-Dateien können drei Arten von „Löchern" (= Ctrl-@ = $00) enthalten:

1. Wenn wir eine Random-Datei anlegen, bei der jeder Record nur ein einziges Feld enthält, z.B.

R0: WALTERr
R1: KARLr..
R2: ADOLFr.

werden die kürzeren Feldinhalte hinten mit Ctrl-@ aufgefüllt (hier symbolisiert durch Punkte).

2. Wenn wir eine Random-Datei anlegen, bei der jeder Record mehrere Felder enthält, z.B.

R0: WALTERrKARLrADOLFr...

werden die kürzeren Recordinhalte mit Ctrl-@ aufgefüllt.

3. Wenn wir bei einer Random-Datei *neue* Records nicht numerisch lückenlos aufsteigend beschreiben, d.h. wenn wir eine Record-Nummer r vorgeben, obwohl es noch Record-Nummern < r gibt, die bislang noch niemals beschrieben wurden, so entstehen Löcher *zwischen* den Records (= echte „Loch"-Dateien).

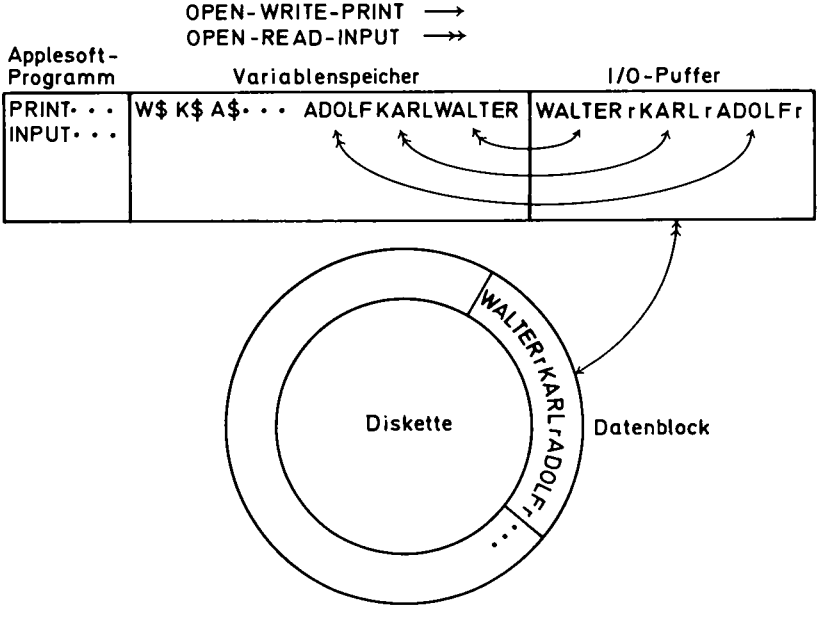
Wenn es uns gelingt, erstens „record-transzendente" „Loch"-Dateien zu vermeiden (Fall 3) und zweitens „record-immanente" Löcher durch Leertasten o.ä. aufzufüllen (Fall 1 und 2), dann sind wir in der Lage, solche Textfiles sowohl im direkten wie auch im indirekten Zugriff, d.h. sequentiell, zu lesen und zu beschreiben. Es ist evident, daß solche Dateien erheblich flexibler bearbeitet werden können, da meistens sowohl eine sequentieller als auch eine Random-Zugriff gewünscht ist. Für den Fall, daß ein Record 3 Felder enthalten soll, ergibt sich beispielsweise folgender Lösungsansatz:

```
100 PRINT CHR$ (4); "OPEN DATEI,L19"
110 PRINT CHR$ (4); "WRITE DATEI, R0"
120 PRINT "WALTER";: REM Semikolon!
```

```
130 PRINT "KARL--";: REM Semikolon!
140 PRINT "ADOLF-": REM nur hier Return!
150 PRINT CHR$ (4); "CLOSE"
160 PRINT CHR$ (4); "OPEN DATEI, L19"
170 PRINT CHR$ (4); "READ DATEI, R0"
180 INPUT G$: PRINT CHR$ (4); "CLOSE"
190 FOR F = 0 TO 2
200 PRINT MID$ (G$, F * 6 + 1, 6)
210 NEXT
```

```
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18
W  A  L  T  E  R  K  A  R  L  -   -   A   D   0   L   F   -   r
```

Ein Record hat bei dieser Datei die obige Struktur, wobei nur die Records und nicht die Felder selbst durch Returns abgegrenzt sind. Die Leertasten oder sonstigen Füllzeichen sind durch „-" symbolisiert. Unter Applesoft-Basic kann ein solcher „gepackter" Record eine Länge von ca. 235 Zeichen haben (vgl. Kapitel 2.5).



2. Vermischte Tips und Tricks

Bis Ende 1984 gab es nur 2 BASIC.SYSTEM-Versionen:
a) 1.0 in Verbindung mit ProDOS 1.0, 1.0.1, 1.0.2 und
b) 1.1 in Verbindung mit ProDOS 1.1 und 1.1.1.
Alt- und Neuversionen haben eine Reihe von Bugs, die wohl bei zukünftigen Versionen (hoffentlich!) nach und nach beseitigt werden. Einige Beispiele:

Trace-Bug

```
10 FOR X = 1 TO 60
20 IF X = 20 THEN FLASH
30 IF X = 40 THEN NORMAL
40 NEXT
```

Das BASIC.SYSTEM, das sich normalerweise in einem nicht sichtbaren Pseudo-Trace-Modus befindet (s. Band 1, S 191), gerät in mehreren Fällen in einen sichtbaren Pseudo-Trace-Modus, der oft nur mit POKE 242, 0 (Trace-Flag-Speicherstelle) abgestellt werden kann. Das obige Bug-Demo legt auch noch Version 1.1 „aufs Kreuz".

HHMMLL-Bug

Wenn man z.B. eine Textdatei mit einer Länge von mehr als $00FFFF bearbeitet hat, wird danach in der BASIC.SYSTEM Global Page bei Version 1.0 das höchstwertige Byte nicht mehr zurückgesetzt, so daß beispielsweise ein nachfolgender BSAVE mit einer Länge von weniger als $00MMLL eine effektive Länge von $HH0000 + $00MMLL erhält.

Field-Bug

Bis zur Version 1.1 konnte man eine Textdatei nicht mit Ff > 1 eröffnen, weil sonst das Applesoft-Programm zerstört wurde (s. Demo FIELD.BUG auf der Begleitdiskette zu diesem Band).

Darüber hinaus hat das BASIC.SYSTEM eine Reihe von Unzulänglichkeiten, auf die in den Kapiteln 2.2 bis 2.7 eingegangen wird. Im Gegensatz zu DOS 3.3 kann man Bugs und Unzulänglichkeiten nicht ohne weiteres durch „Patches" beseitigen, da vermutlich auch in diesem Jahr wieder weitere Versionen erscheinen werden, die jeweils völlig neu assembliert sind, so daß Patches in absoluten Speicherbereichen bei nachfolgenden Versionen nicht mehr funktionieren.

2.1. BASIC.SYSTEM Global Page

Nachfolgend werden alle wichtigen Adressen aus der BASIC.SYSTEM Global Page ($BE00), die sich für „globale" Peeks, Pokes und Calls anbieten, zusammengefaßt. Die PRODOS.SYSTEM Global Page wurde bereits im Band 1 dargestellt. Auf den halbfett gedruckten Labelnamen folgt jeweils im Klammern die Adresse in hexadezimal und dezimal sowie entsprechende Erläuterungen:

BIENTRY ($BE00 = 48640) – Basic Interpreter Entry: Dies ist die Warmstart-Adresse, auf die der Reset-Vektor ($03F2: 00 BE 1B) sowie der Warmstart-Vektor ($03D0: 4C 00 BE) zeigen. Ein Applesoft-Programm könnte anstelle von END mit CALL 48640 beendet werden. Analog wird man am Ende von Maschinenprogrammen oft nach $BE00 springen.

DOSCMD ($BE03 = 48643) – DOS Command: Dies ist die externe Einsprungstelle für im BASIC.SYSTEM bereits implementierte Befehle. Zu diesem Zweck legt man den gewünschten Befehl, z.B. „CATALOG, S3, D2", *mit Bit 7 on* im Eingabepuffer ab $0200 ab (mit $00 als Endmarker) und ruft DOSCMD mit JSR $BE03 auf, worauf der Befehl so ausgeführt wird, als hätte man ihn entweder über die Tastatur eingegeben oder als wäre er von einem laufenden Applesoft-Programm in der üblichen PRINT-Anweisung mit Ctrl-D ausgegeben worden. Wenn kein Fehler auftrat, kehrt das BASIC.SYSTEM mit zurückgesetztem Carry-Flag zum aufrufenden Maschinenprogramm zurück. Andernfalls wird das Carry-Flag gesetzt und der Akkumulator enthält die Fehlernummer. Beispiele für die Anwendung von DOSCMD findet man im Band 1, Seite 43 (BASIC.SYSTEM-Befehle) sowie in diesem Band 2 in den Programmen PRODOS.FID.O und PROFID.

EXTRNCMD ($BE06 = 48646) – External Command: Diese Adresse enthält normalerweise einen Sprung auf ein RTS, ist also zunächst nicht aktiv. Man kann nun in EXTRNCMD+1 und EXTRNCMD+2 das Low- und High-Byte derjenigen Adresse eintragen, wo ein selbst-entwickelter, externer Befehl steht. Ein Beispiel hierfür ist das Programm FP.COMMAND.

Exkurs: Implementierung eines externen Befehls

Nehmen wir als Beispiel an, wir wollten einen eigenen CATALOG-Befehl implementieren, der andere Informationen als der normale CATALOG-Befehl bietet:
1. Zunächst müssen wir uns ein Befehlswort ausdenken, das nicht bereits vom BASIC.SYSTEM benutzt wird, z.B. KATALOG.
2. Dann müssen wir ein Maschinenprogramm schreiben, das später den neuen KATALOG-Befehl ausführt. Dies ist die Hauptarbeit, soll aber hier nicht erörtert werden.
3. Das KATALOG-Programm muß über eine vorgeschaltete Move-Routine verfügen, die erstens über GETBUFR die Anzahl der benötigten Pages anfordert und zweitens das eigentliche KATALOG-Programm in den Bereich oberhalb von HIMEN schiebt. Ferner müssen wir die alten Werte von EXTRNCMD+1 und EXTRNCMD+2 „retten" (= EXTRNCMD.ALT) und die Startadresse unseres KATALOG-Programms in EXTRNCMD+1 und EXTRNCMD+2 eintragen (= EXTRNCMD.NEU).
Danach ist der KATALOG-Befehl implementiert. Gibt man nun im Direkt-Modus über die Tastatur den Befehl
KATALOG, S6, D2
ein oder erteilt aus einem Applesoft-Programm die PRINT-Anweisung
1000 PRINT CHR$ (4) "KATALOG, S6, D2",
so passiert im einzelnen folgendes:
4. Das BASIC.SYSTEM legt den Befehl „KATALOG, S6, D2" intern als Zeichenkette mit einem vorangestellten Länge-Byte ab der Adresse, die in VPATH1 steht (s.u.), ab und stellt fest, daß das Wort „KATALOG" nicht in der eigenen Befehlsworttabelle enthalten ist.
5. Nun springt das BASIC.SYSTEM zur in EXTRNCMD+1 und EXTRNCMD+2 angegebenen externen Befehlsadresse, also zu unserem KATALOG-Programm (= EXTRNCMD.NEU).
6. Wir übernehmen nun aus VPATH1 ($BE6C-$BE6D) das Low- und High-Byte der Adresse des Befehlsstrings $HHLL und erhöhen diese Adresse um 1 auf $HHLL+1, um das Länge-Byte zunächst zu ignorieren.
7. Jetzt prüfen wir, ob die ersten 7 Bytes ab $HHLL+1 dem ASCII-String „KATALOG" (*mit Bit 7 off!*) entsprechen.
8. Wenn nein, verlassen wir unsere KATALOG-Routine, indem wir zunächst das Carry-Flag setzen (SEC sagt dem BASIC.SYSTEM = nicht unser KATALOG-Befehl) und dann zum „geretteten" EXTRNCMD springen (= EXTRNCMD.ALT). (Das BASIC.SYSTEM würde jetzt noch bei EXTRNCMD.ALT den alten Befehl auszuführen versuchen.)
9. Wenn ja, muß nun der „Rest" (= die Parameter-Liste) von „KATALOG, S6, D2", also „,S6, D2", ausgewertet werden. Grundsätzlich sind drei Fälle denkbar:

9a. Ein externer Befehl hat überhaupt keine Parameter, z.B. der „FP"-Befehl in unserem Beispielprogramm. Wir fahren bei 10 fort.
9b. Der externe Befehl hat Parameter, die wir jedoch selbst anhand des Befehlsstrings auswerten, der ab VPATH1 mit dem Länge-Byte beginnt. Zur Verdeutlichung sei wiederholt, daß der Befehlsstring nicht bei VPATH1, sondern bei der in VPATH1 enthaltenen Adresse beginnt. Wir werten nun die Parameter selbst aus und fahren bei 10 fort.
9c. Der externe Befehl hat Parameter, die wir jedoch vom BASIC.SYSTEM selbst auswerten lassen. Wir fahren hierzu bei 11 fort.
10. Im Fall 9a sowie im Fall 9b führen wir jetzt die KATALOG-Routine aus und verlassen sie am Ende, indem wir PBITS und PBITS+1 auf 0 setzen, das Carry-Flag löschen und ein RTS ausführen, also

```
LDA #0
STA PBITS
STA PBITS+1
CLC
RTS
```

Wenn die Parameter-Bits auf 0 gesetzt werden, „weiß" das BASIC.SYSTEM, daß es keine Parameter mehr auswerten muß, womit der externe Befehl für das BASIC.SYSTEM „erledigt" ist. Damit sind die Fälle 9a und 9b hier beendet.

11. Im Fall 9c muß nun im einzelnen folgendes geschehen:
11a. Die Parameter-Bits PBITS und PBITS+1 müssen so gesetzt werden, wie es für den KATALOG-Befehl erforderlich ist. Die PBITS werden weiter unten sowie bei dem Programm MON.COMMANDS näher erläutert. Für den KATALOG-Befehl wären analog zum CATALOG-Befehl %10010101 in PBITS und %00000100 in PBITS + 1 zu speichern.
11b. Nun wäre die Länge (minus 1) des Strings „KATALOG", also 7 - 1 = 6, in XLEN (externe Länge) zu speichern. Ferner muß XCNUM (externe Befehlsnummer) auf 0 gesetzt werden, da unter dem BASIC.SYSTEM alle externen Befehle dieselbe Nummer 0 haben. Schließlich muß in XTRNADR und XTRNADR+1 die externe Rücksprungadresse in den zweiten Teil unserer KATALOG-Routine eingetragen werden.
11d. Danach verlassen wir *vorübergehend* über RTS den ersten Teil unseres KATALOG-Handlers, indem wir zuvor noch das Carry-Flag zurückgesetzt haben. Ein Beispielprogramm für die Positionen 11a – 11d sieht so aus:

```
LDA #%10010101
STA PBITS
LDA #%00000100
STA PBITS+1
LDA #6
```

```
STA XLEN
LDA #0
STA XCNUM
LDA #<RÜCKSPRUNG
STA XTRNADR
LDA #>RÜCKSPRUNG
STA XTRNADR+1
CLC
RTS (Sprung zum BASIC.SYSTEM)
RÜCKSPRUNG (ab hier zweiter Teil der KATALOG-Routine)
```

12. Wenn das BASIC.SYSTEM illegale Parameter vorfand, z.B. „KATALOG, S0, D3", so kehrt es nicht mehr zurück, sondern gibt eine entsprechende Fehlermeldung aus.
13. Wenn die Parameter in Ordnung waren, kehrt das BASIC.SYSTEM zu unserem zweiten Teil namens „RÜCKSPRUNG" zurück, nachdem es zuvor die ausgewerteten Parameter ab $BE58 (= FVALS) abgelegt hat.
14. Wir übernehmen nun diese Parameter, z.B. die Slot- und Drive-Nummer in VSLOT = $BE61 und VDRIV = $BE62, und führen dann den Hauptteil der KATALOG-Routine durch.
15. Danach verlassen wir unser KATALOG-Programm endgültig über

```
CLC
RTS
```

Auf das Zurücksetzen des Carry-Flags kann man auch verzichten, da der externe Befehl nach dem Rücksprung für das BASIC.SYSTEM bereits als erledigt gilt.

ERROUT ($BE09 = 48649): – Error Output: Hier steht ein Sprung zum BASIC.SYSTEM-internen Error-Handler, der entweder die Fehlermeldung ausgibt oder im Falle von ONERR zur ONERR-Zeile springt.

PRINTERR ($BE0C = 48652): – Print Error: Diese Routine gibt zur Fehlernummer, die sich im Akkumulator befinden muß, den Fehlertext aus, z.B. ergibt „LDA #9 JMP PRINTERR" die Meldung „DISK FULL". Dies *muß* im RUN-Modus eines *Applesoft*-Programms geschehen, z.B.:

```
10 REM ONERR.DEMO
20 ONERR GOTO 70
30 INPUT "BEFEHL: ";B$
40 IF B$ = "ENDE" THEN POKE 216,0: END
50 PRINT CHR$ (4);B$: GOTO 20
60 REM Fehlermeldung anzeigen und dann Stack normalisieren
```

```
70 F =  PEEK (222): POKE 640,169: POKE 641,F: POKE 642,76: POKE 643,12:
   POKE 644,190: POKE 645,104: POKE 646,168: POKE 647,104: POKE 648,166:
   POKE 649, 223: POKE 650,154: POKE 651,72: POKE 652,152: POKE 653,72:
   POKE 654,96: CALL 640: CALL 645
80 GOTO 30
```

Man beachte, daß bei *reinen Maschinenprogrammen* PRINTERR selbst dann nicht zum Programm zurückkehrt, wenn der Aufruf mit JSR PRINTERR erfolgt. Hierzu wäre ein Patch erforderlich, der im Band 1, Seite 43, steht. Nach JSR DOSCMD (s.o.) befindet sich die Fehlernummer im Akkumulator, falls das Carry-Flag gesetzt ist. Nunmehr ergeben sich folgende Möglichkeiten:

a) Man gibt die Fehlernummer über JSR PRBYTE (= $FDDA) aus und behält die Kontrolle.

b) Man gibt die Fehlermeldung über JSR PRINTERR aus und verliert die Kontrolle.

c) Man patcht PRINTERR+1 und PRINTERR+2, gibt die Fehlermeldung über JSR PRINTERR aus und behält die Kontrolle. Die letztere Variante wird von mir heute nicht mehr befürwortet, da sie nicht bei allen ProDOS-Versionen funktionieren dürfte.

ERRCODE ($BE0F = 48655): – Error Code: Hier befindet sich zusätzlich die Fehlernummer nach dem zuletzt ausgeführten, fehlerhaften BASIC.SYSTEM-Befehl. Diese Nummer kann man mit PRINT PEEK (48655) abrufen.

OUTVECT0-OUTVECT7: – Output-Vektoren: Hier stehen die Ausgabevektoren (als Slotadressen) in der Form Low Byte – High Byte für die Slots 0-7. Bei nichtbelegten Slots ist der entsprechende Vektor auf die Adresse der Ausgabe von „No device connected" umgelegt. Mit Hilfe des „PR#s,A$HHLL"-Befehls ist es möglich, für den Slot s eine beliebige andere Startadresse zu definieren.

OUTVECT0 ($BE10 = 48656): Slot 0, z.B. $FDF0 (= COUT1)
OUTVECT1 ($BE12 = 48658): Slot 1, z.B. $C100
OUTVECT2 ($BE14 = 48660): Slot 2, z.B. $C200
OUTVECT3 ($BE16 = 48662): Slot 3, z.B. $C300
OUTVECT4 ($BE18 = 48664): Slot 4, z.B. $C400
OUTVECT5 ($BE1A = 48666): Slot 5, z.B. $C500
OUTVECT6 ($BE1C = 48668): Slot 6, z.B. $C600
OUTVECT7 ($BE1E = 48670): Slot 7, z.B. $C700

INVECT0-INVECT7: – Input-Vektoren: Hier stehen die Eingabevektoren (als Slotadressen) in der Form Low Byte – High Byte für die Slots 0-7.

INVECT0 ($BE20 = 48672): Slot 0, z.B. $FD1B (= KEYIN)
INVECT1 ($BE22 = 48674): Slot 1, z.B. $C100
INVECT2 ($BE24 = 48676): Slot 2, z.B. $C200
INVECT3 ($BE26 = 48678): Slot 3, z.B. $C300
INVECT4 ($BE28 = 48680): Slot 4, z.B. $C400
INVECT5 ($BE2A = 48682): Slot 5, z.B. $C500
INVECT6 ($BE2C = 48684): Slot 6, z.B. $C600
INVECT7 ($BE2E = 48686): Slot 7, z.B. $C700

VECTOUT ($BE30 = 48688): – Momentaner Output-Vektor in der Form Low Byte – High Byte; bei 40-Z/Z-Darstellung F0 FD, d.h. $FDF0 = COUT1.

VECTIN ($BE32 = 48690): – Momentaner Input-Vektor in der Form Low Byte – High Byte; bei 40-Z/Z-Darstellung 1B FD, d.h. $FD1B = KEYIN.

Der momentane Output-Vektor kann mit PR#A$HHLL (z.B. PR#A$0300) und der momentane Input-Vektor mit IN#A$HHLL (z.B. IN#A$0350) umgelenkt werden. Assemblerprogrammierer können statt dessen folgendes Verfahren anwenden (Beispiel für Output-Vektoränderung):

```
 LDA VECTOUT
 STA OUTPUT.EXIT+1
 LDA VECTOUT+1
 STA OUTPUT.EXIT+2
 LDA #<OUTPUT.ENTRY
 STA VECTOUT
 LDA #>OUTPUT.ENTRY
 STA VECTOUT+1
 RTS
```

Hier Beginn neuer Output. Akkumulator enthält beim Entry auszugebendes Zeichen. Das Demo vertauscht zwei Zeichen, hier „ß" mit „↑", wie dies gelegentlich bei Typenraddruckern mit Nicht-ASCII-Typenrädern in ähnlicher Form erforderlich ist.

```
OUTPUT.ENTRY ORA #$80
ESZETT CMP #"ß"
 BNE PFEIL
 LDA #"↑"
 BNE OUTPUT.EXIT
PFEIL CMP #"↑"
 BNE OUTPUT.EXIT
 LDA #"ß"
```

Gepokte Adresse bei OUTPUT.EXIT

```
OUTPUT.EXIT JMP $FFFF
```

Das BASIC.SYSTEM läßt sich mit „POKE 242, 0: PR#0: IN#0" völlig abhängen. Mit CALL 39424 oder JSR $9A00 läßt es sich wieder anhängen. Die Adresse $9A00 ist jedoch möglicherweise nicht für alle Versionen gültig. Ersatzweise kann man nach $BE00 springen (CALL 48640) oder Ctrl-Reset drücken. Ctrl-Reset hebt übrigens auch stets über PR#A$HHLL oder in der obigen Art „verbogene" Vektoren auf und schaltet auf diejenigen Werte um, die sich in OUTVECT0 und INVECT0 finden.

VDOSIO ($BE34-$BE37 = 48692-48695)
VSYSIO ($BE38-$BE3B = 48696-48699)

Zu diesen internen DOS- und System-Vektoren sind einige allgemeine Anmerkungen erforderlich (siehe auch Band 1, S. 40ff.):
a) Die Standard-ROM-Output-Adresse ist $FDED = COUT (= Character Output). Die Standard-ROM-Input-Adresse ist $FD0C = RDKEY (= Read Key).
b) COUT springt indirekt nach $0036 = CSWL = Character Output Switch Low Byte. RDKEY springt indirekt nach $0038 = KSWL = Keyboard Input Switch Low Byte.
c) Das BASIC.SYSTEM trägt in CSWL/CSWH eine Output-Abfangadresse (aus $BE34-$BE35 VDOSIO) und in KSWL/KSWH eine Input-Abfangadresse (aus $BE36-$BE37 VDOSIO = Vector DOS Input Output) ein. Daneben gibt es noch einen Umlenkungsvektor für Output ($BE38-$BE39 VSYSIO) und Input ($BE3A-$BE3B VSYSIO = Vector System Input Output). Je nachdem, ob man sich im Direkt-Modus oder im RUN-Modus befindet, kommen andere interne Vektoren zum Tragen. Beispielsweise ist bei ProDOS 1.0.1 die „echte" Vektoradresse für Direkt-Modus $9A2F und für RUN-Modus $9E5B.
Welchen praktischen Nutzen kann man aus diesen Erläuterungen ziehen? Nehmen wir an, wir wollten in einem laufenden Applesoft-Programm einen Text an den Bildschirm, den Drucker oder ein Modem ausgeben, ohne daß das BASIC.SYSTEM etwas „merkt". In diesem Fall
a) „retten" wir CSWL/CSWH,
b) ersetzen CSWL/CSWH durch den Inhalt von VECTOUT/VECTOUT+1,
c) geben den Text aus und
d) ersetzen wieder CSWL/CSWH durch die ursprünglichen Werte.
(Vgl. hierzu das Programm MON.COMMANDS).

DEFSLT ($BE3C = 48700): – Default Slot.
DEFDRV ($BE3D = 48701): – Default Drive.

DEFSLT enthält den Ersatz-Slot ($01-$07) und DEFDRV den Ersatz-Drive ($01-$02). Unmittelbar nach dem Booten mit PR#6 o.ä. bzw. nach -PRODOS werden bei den bisherigen ProDOS-Versionen 1.0.1, 1.0.2 und 1.1.1 die Boot-Slot/Drive-Werte hier eingetragen. Später werden Sie nur dann aktualisiert, wenn ein Slot oder Drive mit z.B. CAT, S6, D2 usw. angesprochen wird. Das BASIC.SYSTEM übernimmt die Ersatzwerte nur dann, wenn ein Null-Präfix („PREFIX/") definiert wurde. BASIC.SYSTEM-Befehle mit Präfix, z.B. CAT/RAM, ändern nicht automatisch die Ersatz-Werte, was man beweisen kann, indem mal falsche Ersatzwerte in $BE3C-$BE3D und $BF30 pokt. Im Falle eines Null-Präfixes könnte man neue Ersatz-Slot/Drive-Werte poken. Ansonsten hat das definierte Präfix Vorrang. Die definitiv zuletzt angesprochene Unit-Nummer läßt sich nur DEVNUM bzw. LASTDEV ($BF30 = 48944) entnehmen, wie das folgende Mini-Programm zeigt:

```
100 REM LASTDEVICE
110 U = PEEK (48944): D = 1
120 IF U > 127 THEN D = 2: U = U - 128
130 S = U / 16
140 PRINT "Slot ";S; " Drive ";D
```

Die nachfolgenden Adressen sind weniger wichtig und sollen deshalb nur der Vollständigkeit halber erwähnt werden:

PREGA ($BE3E = 48702): – A-Register-Zwischenspeicher.
PREGX ($BE3F = 48703): – X-Register-Zwischenspeicher.
PREGY ($BE40 = 48704): – Y-Register-Zwischenspeicher.
DTRACE ($BE41 = 48705): – Wenn Bit 7 on, d.h. Wert größer als 127, dann Applesoft-Trace aktiv.
STATE ($BE42 = 48706): – Wenn 0, dann Direkt-Modus. Wenn größer als 0, dann RUN-Modus.
EXACTV ($BE43 = 48707): – Wenn Bit 7 on, dann EXEC-File aktiv.
IFILACTV ($BE44 = 48708): – Wenn Bit 7 on, dann findet gerade ein (Textfile-)Input bzw. READ statt.
OFILACTV ($BE45 = 48709): – Wenn Bit 7 on, dann findet gerade ein (Textfile-)Output bzw. Write statt.
PFXACTV ($BE46 = 48710): – Wenn Bit 7 on, dann wird gerade ein Präfix gelesen.
DIRFLG ($BE47 = 48711): – Wenn Bit 7 on, dann wird gerade ein Directory gelesen.

EDIRFLG ($BE48 = 48712): – Flag für Directory-Ende (bei älteren ProDOS-Versionen).
STRINGS ($BE49 = 48713): – String-Pool-Zähler für Garbage Collection.
TBUFPTR ($BE4A = 48714): – Byteanzahl beim gepufferten WRITE.
INPTR ($BE4B = 48715): – Länge des Tastatur-Eingabestrings (bei $0200).

CHRLAST ($BE4C = 48716): – Zuletzt ausgegebenes Zeichen (unterdrückt Mehrfach-Returns). Zu den skurrilen Besonderheiten des BASIC.SYSTEMs gehört es, daß Mehrfach-Returns in einem Maschinenprogramm, das z.B. mit 300G gestartet wird, unterdrückt werden. (Dies gilt nicht für Applesoft-Programme.) Wenn man trotzdem z.B. 2 aufeinanderfolgende Returns ausgeben will, kann man entweder zwischendurch ein nicht-störendes Zeichen senden oder CHRLAST dekrementieren.

Mit Leertaste dazwischen:
```
LDA #$8D ;1. Return
JSR COUT
LDA #$A0 ;Space
JSR COUT
LDA #$8D ;2. Return
JSR COUT
```

Mit Decrement CHRLAST dazwischen:
```
LDA #$8D ;Return
DEC CHRLAST ;Scramble CHRLAST
JSR COUT
DEC CHRLAST
JSR COUT
```

OPENCNT ($BE4D = 48717): – Anzahl der OPEN-Files (ohne möglichen EXEC-File).
EXFILE ($BE4E = 48718): – Wenn Bit 7 on, dann EXEC-File gerade geschlossen.
CATFLAG ($BE4F = 48719): – Zu formatierender Zeilentyp beim Lesen eines Directory (CATALOG-Befehl).

Die nachfolgenden Adressen wurden bereits oben unter EXTNCMD besprochen:

XTRNADR ($BE50-$BE51 = 48720-48721): – Externe Rücksprungadresse (Low Byte – High Byte).
XLEN ($BE52 = 48722): – Länge des externen Befehlswortes minus 1, z.B. „DUMP" = 4 - 1 = 3.

XCNUM ($BE53 = 48723): – Externe Befehlsnummer immer auf 0 setzen; enthält sonst meist den Wert $FF.

PBITS ($BE54-$BE55 = 48724-48725)
FBITS ($BE56-$BE57 = 48726-48727)

Die PBITS = Permitted Parameter Bits = Soll-Parameter und FBITS = Found Parameter Bits = Ist-Parameter sind eine Wissenschaft für sich. Deshalb sollte man sich das ausführlich kommentierten Programm MON.COMMANDS genauer ansehen. Die nachstehende Tabelle umfaßt die PBITS aller zur Zeit definierten Befehle. Die Buchstaben in der Kopfleiste bedeuten im einzelnen:

p = Präfix suchen (bei CAT und PREFIX)
s = nur Slot-Parameter (bei PR#s und IN#s)
i = Immediate Mode (Direkt-Modus) unzulässig (bei WRITE usw.)
n = Name kann ganz entfallen (bei CAT, RUN usw.)
c = Datei CREATEn, falls noch nicht existent
T = Typ der Datei zulässig (TTXT usw.)
2 = 2. Name erforderlich (nur bei RENAME)
1 = 1. Name erforderlich (bei fast allen Befehlen, vgl. n)

A = A-Parameter (Anfangsadresse) zulässig
B = B-Paramter (Byte-Offset) zulässig
E = E-Parameter (Endadresse) zulässig
L = L-Parameter (Länge) zulässig
§ = §-Parameter (Zeilennummer) zulässig
S = S- *und* D-Parameter (Slot/Drive) zulässig
F = F-Parameter (Feldnummer) zulässig
R = R-Parameter (Recordnummer) zulässig

Wenn man z.B. den Befehl „VERIFY" ohne Parameter (also *nicht* „VERIFY DATEI.XYZ") eingibt, erscheint die Meldung „COPYRIGHT APPLE COMPUTER". Dies ist offenbar ein kleiner Spaß des ProDOS-Programmierers. Daß keine Fehlermeldung („Syntax Error" o.ä.) erscheint, erklärt sich nunmehr aus der Tatsache, daß bei VERIFY n = 1 ist, d.h. der Name kann gänzlich entfallen.
Man beachte, daß es Befehle ohne irgendwelche Parameter gibt, z.B. FRE. Ferner sei darauf hingewiesen, daß bei späteren ProDOS-Versionen die Befehle umdefiniert werden könnten, so daß die Tabelle entsprechend abzuändern ist.

p	s	i	n	c	T	2	1	A	B	E	L	§	S	F	R	
1	0	0	1	0	1	0	1	0	0	0	0	0	1	0	0	CAT
1	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	PREFIX
0	0	0	0	1	1	0	1	0	0	0	0	0	1	0	0	CREATE
0	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0	VERIFY
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	LOCK
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	UNLOCK
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	DELETE
0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	RENAME
0	0	0	1	0	0	0	1	0	0	0	0	1	1	0	0	RUN
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	LOAD
0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	SAVE
0	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	CHAIN
0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	0	STORE
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	RESTORE
0	0	0	0	0	0	0	1	1	1	1	1	0	1	0	0	BRUN
0	0	0	0	0	1	0	1	1	1	1	1	0	1	0	0	BLOAD
0	0	0	0	1	1	0	1	1	1	1	1	0	1	0	0	BSAVE
0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	1	EXEC
0	0	1	0	1	1	0	1	0	0	0	1	0	1	0	0	OPEN
0	0	1	0	1	1	0	1	0	0	0	1	0	1	0	0	APPEND
0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	1	WRITE
0	0	1	0	0	0	0	1	0	1	0	1	0	0	1	1	READ
0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1	POSITION
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	FLUSH
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	CLOSE
0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	-Befehl
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	PR#
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	IN#
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	FRE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	BYE
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	
PBITS								PBITS+1								
FBITS								FBITS+1								

FVALS ($BE58-$BE6B = 48728-48747): Found Values – Hier werden die ausgewerteten Parameter eines Befehls abgelegt. Wenn man z.B. den Befehl „BSAVE XXX, A$2000, L$1000, S6, D1" erteilt, werden in den entsprechenden FVALS die Werte von A, L, S und D als Hex-Zahlen angelegt. Beachte die Logik: Wenn keine Parameter eingegeben werden, stehen sie auch nachher nicht bei den FVALS. Nach dem Booten enthalten die FVALS so lange „Schrott", d.h. Zufallswerte, bis die jeweiligen Parameter in Form eines entsprechenden Befehls definiert worden sind. So findet man nach „BLOAD XXX" nicht die Startadresse und Länge der Datei „XXX" in den FVALS. Im übrigen sind die FVALS nur unmittelbar nach einem Parameter-Befehl als gültig anzusehen und lassen sich dann als 1-Byte-Zahl (LL), 2-Byte-Zahl (LLHH) oder 3-Byte-Zahl (LLMMHH) peeken (vgl. auch das Programm MON.COMMANDS):
1 Byte: PRINT PEEK (FVAL)
2 Bytes: PRINT PEEK (FVAL) + PEEK (FVAL+1) * 256
3 Bytes: PRINT PEEK (FVAL) + PEEK (FVAL+1) * 256 + PEEK (FVAL+2) * 65536

VADDR ($BE58-$BE59 = 48728-48729): Value of Address = A-Wert (LLHH).
VBYTE ($BE5A-$BE5C = 48730-48732): Value of Byte Offset = B-Wert (LLMMHH).
VENDA ($BE5D-$BE5E = 48733-48734): Value of End Address = E-Wert (LLHH).
VLNTH ($BE5F-$BE60 = 48735-48736): Value of Length = L-Wert (LLHH).
VSLOT ($BE61 = 48737): Value of Slot (bei CAT, S6 usw.) = S-Wert (LL); vgl. unten VIOSLT = s-Wert.
VDRIV ($BE62 = 48738): Value of Drive = D-Wert (LL).
VFELD ($BE63-$BE64 = 48739-48740): Value of Field = F-Wert (LLHH).
VRECD ($BE65-$BE66 = 48741-48742): Value of Record = R-Wert (LLHH). Der R-Wert geht also nur theoretisch bis $FFFFFF = 16777215 und ist praktisch auf $FFFF = 65535 begrenzt.
VVOLM ($BE67 = 48743): Value of Volume Number (früher DOS 3.3) = V-Wert (1-254); wird ignoriert, da es kein PBIT für V gibt.
VLINE ($BE68-$BE69 = 48744-48745): Value of Line = §-Wert (LLHH).
VTYPE ($BE6A = 48746): Value of Type = T-Wert (LL). Hier wird also nicht die ASCII-Folge für „TXT" usw., sondern die Typ-Nummer, also $04 usw., abgelegt.
VIOSLT ($BE6B = 48747): Value of I/O-Slot (bei PR#s) = s-Wert (LL); vgl. oben VSLOT = S-Wert.

Dez-Hex-Konverter:
Als „Abfallprodukt" der obigen Peeks verfügt man über einen „Quick-and-Dirty"-Dezimal-Hexadezimal-Konverter für 24-Bit-Zahlen, wenn man zu folgendem Trick greift (Beipiel):
BLOAD BBB, B16000000
(Fehlermeldung)

CALL -151
BE5A.BE5C
(Zeigt LL MM HH an)
Ctrl-C
Anmerkungen: „BBB" muß eine nicht-existente Binärdatei sein. Bevor das BASIC.SYSTEM die Fehlermeldung ausgibt, hat es bereits die Dezimalzahl „B16000000" oder allgemein „Bzzzzzzzz" ausgewertet und in VBYTE abgelegt, so daß man sie im Monitor mit BE5A.BE5C ansehen kann.

VPATH1 ($BE6C-$BE6D = 48748-48749)
VPATH2 ($BE6E-$BE6F = 48750-48751)

Hier befinden sich die Adressen des 1. Pfadnamens sowie des 2. Pfadnamens (nur bei RENAME benötigt) im üblichen Format (Low Byte – High Byte). VPATH1 weist auf eine Speicherstelle innerhalb des BASIC.SYSTEMs, während VPATH2 bei den bisherigen ProDOS-Versionen auf $0280, d.h. die Mitte des Eingabepuffers, zeigt. VPATH1 (und evtl. VPATH2) sind wichtig für externe Befehle (siehe Anfang dieses Kapitels).

MLI-Aufrufe ($BE70-$BEDE = 48752-48862)

Hier befinden sich die Parameter-Listen für die vom BASIC.SYSTEM vorgenommenen MLI-Aufrufe. Eine detaillierte Analyse erübrigt sich, da Assemblerprogrammierer tunlichst eigene Parameter-Listen für MLI-Aufrufe anlegen sollten, weil sich die einzelnen Adressen innerhalb der BASIC.SYSTEM Global Page bei späteren Versionen ändern könnten. Der Vollständigkeit halber seien die Adressen trotzdem kurz beschrieben.

GOSYSTEM ($BE70-$BE84): Allgemeine und zugleich einzige Einsprungadresse für MLI-Aufrufe durch das BASIC.SYSTEM. GOSYSTEM wird intern mit A-Register = MLI-Befehlsnummer aufgerufen. Die Adresse der Parameter-Liste ergibt sich als Offset aus der umgewandelten Befehlsnummer.
SYSCALL ($BE85): Befehlsnummer des MLI-Aufrufs. Bei den gegenwärtigen ProDOS-Versionen kommen nur $C0-$D3 (CREATE usw.) als Befehlsnummern vor, d.h. Blockread- und Interrupt-Routinen entfallen.
SYSPARM ($BE86-$BE87): Adresse der Parameter-Liste (wird jeweils gepokt).
SYSRTS ($BE88-$BE8A): Rückkehr vom MLI.
BADCALL ($BE8B-$BE9E): Umwandlung der MLI-Fehlernummer in eine BASIC.SYSTEM-Fehlernummer.
BISPARE1 ($BE9F): zur Zeit nicht benutzt.

SCREATE ($BEA0-$BEAB): Parameter-Liste für CREATE. Das „S" vor dem Labelnamen steht für „System".
SSGPRFX ($BEAC-$BEAE): Kombinierte Parameter-Liste für GET PREFIX, SET PREFIX und DESTROY.
SRENAME ($BEAF-$BEB3): Parameter-Liste für RENAME.
SSGINFO ($BEB4-$BEC5): Kombinierte Parameter-Liste für SET FILE INFO und GET FILE INFO.
SONLINE ($BEC6-$BECA): Kombinierte Parameter-Liste für ONLINE, SET MARK, GET MARK, SET EOF, GET EOF, SET BUF und GET BUF. (Laut „Beneath Apple ProDOS" auch für REBOOT = BYE, doch würde dann die Befehlsnummer-Umrechnung von $65 bei GOSYSTEM „durchdrehen". In der Version 1.1.1 wird REBOOT bei $B02E *direkt* über das MLI aufgerufen.)
SOPEN ($BECB-$BED0): Parameter-Liste für OPEN.
SNEWLN ($BED1-$BED4): Parameter-Liste für NEWLINE. Bei Textfiles gilt stets $8D/$0D als Newline-Character.
SREAD ($BED5-$BEDC): Kombinierte Parameter-Liste für READ und WRITE.
SCLOSE ($BEDD-$BEDE): Kombinierte Parameter-Liste für CLOSE und FLUSH.

CCCSPARE ($BEDF = 48863): zur Zeit nicht benutzt.
COPYRIGHT ($BEF0-$BEF4 = 48864-48884): Copyright-Vermerk. Man beachte, daß der Copyright-Vermerk, der nach „VERIFY" erscheint, nicht hier, sondern innerhalb des BASIC.SYSTEMs verschlüsselt abgelegt ist.

GETBUFR ($BEF5-$BEF7 = 48885-48887): Get Buffer.
FREEBUFR ($BEF8-$BEFA = 48888-48890): Free Buffer.
BHIMEM ($BEFB = 48891): BASIC.SYSTEM HIMEM (High Byte).

Nach dem Booten von ProDOS liegt HIMEM zunächst bei $9600, d.h. BHIMEM enthält High Byte $96. Der allgemeine I/O-Puffer liegt dann im Bereich $9600-$99FF und das BASIC.SYSTEM selbst beginnt bei $9A00. Mit

```
LDA PAGES
JSR GETBUFR
BCS ERROR
STA PUFBEGH
RTS
```

kann man sich einen Freiraum (= Maschinenprogramm-Puffer) zwischen dem Ende des I/O-Puffers und dem Anfang des BASIC.SYSTEMs schaffen. Im Falle von PAGES = 1 = 256 Bytes würde nach erfolgreichem Rücksprung das A-Register $99 für $9900 (PUF-

BEGH = Pufferbeginn High Byte) enthalten und HIMEM würde nunmehr bei $9500 liegen. GETBUFR kann wiederholt aufgerufen werden, wobei sich der momentane Maschinenprogramm-Puffer jeweils um die Anzahl der angeforderten Pages weiter nach unten verschiebt.
MIT JSR FREEBUFR werden *alle* Maschinenprogramm-Puffer wieder entfernt und HIMEM erneut auf $9600 gesetzt. Die System Bit Map interessiert das BASIC.SYSTEM dabei nicht! Setzt man jedoch BHIMEM auf einen Wert, der niedriger als $96 ist, so bleibt der Bereich des neuen BHIMEM bis zum alten BHIMEM von FREEBUFR ausgespart. Dieser Trick ist von der Firma Apple nicht dokumentiert und funktioniert möglicherweise nicht in zukünftigen ProDOS-Versionen.

NOTUSED ($BEFC-$BEFF = 48892-48895): zur Zeit nicht benutzt.

2.2. LOMEM, FRE und HGR

Wenn das BASIC.SYSTEM eine Textdatei mit OPEN eröffnet, wird HIMEM bis zum CLOSE dieser Datei vorübergehend um $0400 Bytes nach unten verschoben (z.B. von $9600 nach $9200). Dazu ist es erforderlich, intern eine Garbage-Collection vorzunehmen, um die unterhalb von HIMEM (normal $9600) befindlichen Strings zu „retten". Es findet dabei eine „wüste Speicherschieberei" statt, die beispielsweise den Bereich nach LOMEM in den Bereich ab $9800 schiebt usw. Das traurige Resultat ist dann oft eine Zerstörung des HGR-Bereichs ($2000-$3FFF für HGR1 und $4000-$5FFF für HGR2). Die nachfolgenden zwei Demos FRE.HGR1 und FRE.HGR2, die einen ursprünglichen HIMEM-Wert von $9600 voraussetzen, veranschaulichen dies auf eindrucksvolle Weise.

FRE.HGR1

```
100 HOME : TEXT
110 PRINT "FRE.HGR1"
120 HGR
130 A = 1150
140 DIM A(A)
150 FOR X = 1 TO A:A(X) = 1
160 NEXT
170 GET X$
180 PRINT CHR$ (4)"OPEN XXX"
190 PRINT CHR$ (4)"WRITE XXX"
200 PRINT 1
210 PRINT CHR$ (4)"CLOSE"
220 GET X$
230 TEXT
```

FRE.HGR2

```
100 HOME : TEXT
110 PRINT "FRE.HGR2"
120 HGR2
130 A = 2750
140 DIM A(A)
150 FOR X = 1 TO A:A(X) = 1
160 NEXT
170 GET X$
180 PRINT CHR$ (4)"OPEN XXX"
190 PRINT CHR$ (4)"WRITE XXX"
200 PRINT 1
210 PRINT CHR$ (4)"CLOSE"
220 GET X$
230 TEXT
```

Wenn man bei diesen Programmen jeweils die Zeile
170 GET X$
entfernt, findet keine Garbage-Collection statt und die HGR-Seiten werden nicht angetastet. Dies zeigt jedoch gleichzeitig, daß bereits ein einziger String-Buchstabe das BASIC.SYSTEM veranlaßt, eine „Speicherschieberei" vorzunehmen. Ein Grafik-Programm dürfte deshalb überhaupt keine Strings enthalten, was mehr als unrealistisch ist. Das Problem läßt sich nur dadurch umgehen, daß man das Applesoft-Programm in den Bereich *nach* der benötigten HGR-Seite legt. Hierzu muß man den sog. TXTTAB-Zeiger durch ein kleines Programm ändern und dann das eigentliche Programm starten.

TXTTAB

```
10 REM TXTTAB-Änderung
20 REM $0800= 2048 normal
30 REM $4000=16384 nach HGR1
40 REM $6000=24576 nach HGR2
50 P = 16384: REM Programmanfang
60 POKE P,0:P = P + 1:PH = INT (P / 256):
   PL = P - PH * 256: POKE 103,PL: POKE 104,PH: CLEAR
70 REM Jetzt eigenes Programm laden mit
   PRINT CHR$ (4); "RUN XXX" o.ä.
```

Der allgemeine Puffer (für CAT, BLOAD usw.) ist immer der adreßmäßig niedrigste, also CAT-Puffer *vor* Open $9600, *nach* Open $9200, nach Close wieder $9600 usw.

2.3. HIMEM und Maschinenprogramme

Abgesehen von dem Page-3-Bereich ($0300-$03CF), der für kleine Maschinenprogramme immer frei ist, erhebt sich die Frage, wo man unter dem BASIC.SYSTEM Maschinenprogramme unterbringen kann, die von einem Applesoft-Programm aufgerufen werden sollen. (Das Problem stellt sich nicht für Stand-alone-Assemblerprogramme, da bei diesen sogar auf das BASIC.SYSTEM insgesamt verzichtet werden könnte.)

1. Man kann den TXTTAB-Zeiger, der ursprünglich auf $0801 zeigt, mit dem obigen TXTTAB-Programm erhöhen und dann das Maschinenprogramm ab $0801 (besser $0803) in den Speicher einladen. Es gilt somit folgende Speicherverteilung:

- Anfang der Maschinenprogramm (z.B. $0803)
- Anfang des Applesoft-Programms (z.B. $4001)
- HIMEM (z.B. $9600)

2. Man kann LOMEM (= Applesoft-Variablenanfang) heraufsetzen und dann Maschinenprogramme zwischen Applesoft-Programmende und neuem LOMEM einladen. Dann ergibt sich folgende Speicherverteilung:

- Anfang des Applesoft-Programms ($0801)
- Ende des Applesoft-Programms (z.B. $1E20)
- Anfang des Maschinenprogramms (z.B. $2000)
- Anfang des Variablenspeichers = LOMEM (z.B. $2500)
- HIMEM (z.B. $9600)

3. Man kann über die GETBUFFER-Routine und/oder durch Änderung der System Bit Map HIMEM um die erforderliche Anzahl der Pages herabsetzen und dann das Maschinenprogramm oberhalb von HIMEM + $0400 einladen. Dann ergibt sich folgende Speicherverteilung:

- Anfang des Applesoft-Programms ($0801)
- Anfang des Variablenspeichers (z.B. $2134)
- HIMEM = Anfang des I/O-Puffers (z.B. $9200)
- Anfang des Maschinenprogramms (z.B. $9600)
- Anfang des BASIC.SYSTEMs ($9A00)

System-Bit-Map-Methode

1. Man achtet darauf, daß momentan kein Textfile offen ist, weil sonst HIMEM nicht den Originalwert hat.
2. Man lädt das Maschinenprogramm in einen mittleren Bereich, z.B. ab $2000.
3. Eine dem eigentlichen Maschinenprogramm vorgelagerte Routine
3.1. ändert die System Bit Map
3.2. setzt den HIMEM-Zeiger ($0073-$0074) herab
3.3. verschiebt das eigentliche Maschinenprogramm in den vorgesehenen Bereich oberhalb vom neuen HIMEM + $0400.
(Vgl. hierzu Kapitel 2.10 sowie Band 1, Seite 118.)

GETBUFFER-Methode

1. Man achtet darauf, daß momentan kein Textfile offen ist.
2. Man fordert über LDA Page-Anzahl JSR GETBUFR die erforderliche Anzahl der Pages an.
3. Man BLOADet das Maschinenprogramm in den Bereich oberhalb vom neuen HIMEM + $0400.
(Vgl. hierzu Seite 78f. sowie Kapitel 2.8 und 2.9.)

Man beachte, daß sich HIMEM stets auf einer Seitengrenze befinden muß, d.h. das Low Byte muß immer 0 sein ($0073: 00 HH). Andernfalls dreht das BASIC.SYSTEM völlig durch.
Die GETBUFFER-Methode ändert nicht die System Bit Map, so daß nachträglich andere Programme *direkt* in den Bereich des alten Maschinenprogramms geBLOADet werden können. Dies ist bei der System-Bit-Map-Methode nicht möglich.

2.4. Open-Dateien bei 1-Drive-Besitzern

Stellen Sie sich vor, Sie hätten nur ein einziges Drive 1 und 2 Disketten A und B. Nun wollen Sie einen großen Textfile von A in Drive 1 auf B in Drive 1 kopieren, indem Sie zwischendurch Disketten wechseln. Dies war unter DOS 3.3 möglich. Unter ProDOS funktioniert dies nicht mehr, weil eine „Unit" (Slot + Drive) nach einem OPEN bis zum nächsten CLOSE unlöslich mit einem einzigen Volume (= Diskette) verknüpft sein muß. Die nachfolgenden 3 Open-Tests veranschaulichen dies. Es gibt hier nur eine Lösung: Vor dem Diskettenwechsel muß die Datei auf der momentanen Diskette geschlossen und später wieder geöffnet werden. Dies ist indes mit erheblichem Zeitverlust verbunden.

OPEN.TEST1

```
100 REM OPEN.TEST1
110 PRINT CHR$ (4)"OPEN/A/XXX": PRINT CHR$ (4)"WRITE/A/XXX":
    FOR X = 1 TO 100: PRINT "1234567890": NEXT : PRINT CHR$ (4)"CLOSE"
120 PRINT CHR$ (4)"OPEN/A/XXX"
130 PRINT CHR$ (4)"READ/A/XXX"
140 FOR X = 1 TO 50: INPUT X$: NEXT
150 PRINT CHR$ (4): INPUT "B:";Y$
160 PRINT CHR$ (4)"OPEN/B/YYY"
170 PRINT CHR$ (4)"WRITE/B/YYY"
180 FOR X = 1 TO 50: PRINT X$: NEXT
190 PRINT CHR$ (4): INPUT "A:";Y$
200 PRINT CHR$ (4)"READ/A/XXX"
210 FOR X = 1 TO 50: INPUT X$: NEXT
220 PRINT CHR$ (4): INPUT "B:";Y$
230 PRINT CHR$ (4)"WRITE/B/YYY"
240 FOR X = 1 TO 50: PRINT X$: NEXT
250 PRINT CHR$ (4)"CLOSE"
260 REM Funktioniert nur bei 2 verschiedenen Disketten
    in 2 verschiedenen Drives
```

OPEN.TEST2

```
100 REM OPEN.TEST2
110 PRINT CHR$ (4)"OPEN/A/XXX,S6,D1": PRINT CHR$ (4)"WRITE/A/XXX":
    FOR X = 1 TO 100: PRINT "1234567890": NEXT : PRINT CHR$ (4)"CLOSE"
120 PRINT CHR$ (4)"OPEN/A/XXX,S6,D1"
130 PRINT CHR$ (4)"READ/A/XXX"
140 FOR X = 1 TO 50: INPUT X$: NEXT
150 PRINT CHR$ (4): INPUT "B:";Y$
160 PRINT CHR$ (4)"OPEN/B/YYY,S6,D1"
170 PRINT CHR$ (4)"WRITE/B/YYY"
180 FOR X = 1 TO 50: PRINT X$: NEXT
190 PRINT CHR$ (4): INPUT "A:";Y$
200 PRINT CHR$ (4)"READ/A/XXX"
210 FOR X = 1 TO 50: INPUT X$: NEXT
220 PRINT CHR$ (4): INPUT "B:";Y$
230 PRINT CHR$ (4)"WRITE/B/YYY"
240 FOR X = 1 TO 50: PRINT X$: NEXT
250 PRINT CHR$ (4)"CLOSE"
260 REM Funktioniert bei 2-Drive-Besitzern, weil 'S6,D1'
    ignoriert wird, wenn 'B'-Volume gefunden wurde!
```

OPEN.TEST3

```
100 REM OPEN.TEST3
110 PRINT CHR$ (4)"OPEN XXX,S6,D1": PRINT CHR$ (4)"WRITE XXX":
    FOR X = 1 TO 100: PRINT "1234567890": NEXT : PRINT CHR$ (4)"CLOSE"
```

```
120 PRINT CHR$ (4)"OPEN XXX,S6,D1"
130 PRINT CHR$ (4)"READ XXX"
140 FOR X = 1 TO 50: INPUT X$: NEXT
150 PRINT CHR$ (4): INPUT "B:";Y$
160 PRINT CHR$ (4)"OPEN YYY,S6,D1"
170 PRINT CHR$ (4)"WRITE YYY"
180 FOR X = 1 TO 50: PRINT X$: NEXT
190 PRINT CHR$ (4): INPUT "A:";Y$
200 PRINT CHR$ (4)"READ XXX"
210 FOR X = 1 TO 50: INPUT X$: NEXT
220 PRINT CHR$ (4): INPUT "B:";Y$
230 PRINT CHR$ (4)"WRITE YYY"
240 FOR X = 1 TO 50: PRINT X$: NEXT
250 PRINT CHR$ (4)"CLOSE"
260 REM Funktioniert nur, wenn keine Diskette gewechselt wird!
```

2.5. Input von Strings mit Komma und Doppelpunkt

Für den INPUT-Befehl beim Textfile-READ gelten dieselben Einschränkungen wie beim normalen Applesoft-INPUT. Tippen Sie
NEW
10 INPUT X$: PRINT X$: GOTO 10
RUN
und führen Sie dann folgende Experimente durch:
1. Geben Sie einen String ein, der Kommas und/oder Doppelpunkte enthält. Sie werden sehen, daß der String-Rest nach dem ersten Komma oder Doppelpunkt abgehackt wird. Dies gilt auch für das BASIC.SYSTEM, führt aber im Gegensatz zu DOS 3.3 nicht zu der EXTRA-IGNORED-Meldung.
2. Geben Sie einen String ein, dessen erstes Zeichen ein Anführungszeichen ist und der Kommas und/oder Doppelpunkte enthält. Sie werden sehen, daß zwar das Anführungszeichen nicht der Variablen X$ zugewiesen wird, daß jedoch dafür die Kommas, Doppelpunkte und führenden Leerzeichen erhalten bleiben und somit der String-Rest nicht abgehackt wird.
3. Geben Sie einen String mit mehreren Anführungszeichen *nach* der ersten Stelle ein. Sie werden sehen, daß die Anführungszeichen inmitten des Strings erhalten bleiben.

Um das Problem von Komma/Doppelpunkt/Anführungszeichen zu lösen, bieten sich für den Textfile-READ drei Möglichkeiten an:

1. Man verzichtet auf Kommas und Doppelpunkte und kann dann alle anderen Zeichen einschließlich des Anführungszeichens einlesen, falls das erste Zeichen kein Anführungs-

zeichen ist. Die maximale Stringlänge beträgt beim BASIC.SYSTEM hier 238 Bytes, d.h. 1 Byte weniger als unter DOS 3.3 (Bug!). Vgl. STRING.SAVER.1 in Verbindung mit INPUT.LOADER (und INALL.LOADER).
Ein String darf übrigens fast alle Ctrl-Zeichen enthalten. Ausgenommen sind
Ctrl-@ = CHR$ (0) – Endmarker
Ctrl-D = CHR$ (4) – DOS-Erkennungszeichen
Ctrl-M = CHR$ (13) – Return als Feldbegrenzer

2. Man verzichtet auf das Anführungszeichen und leitet jeden WRITE-PRINT mit einem CHR$ (34) = Anführungszeichen ein. Die maximale Stringlänge beträgt beim BASIC.SYSTEM hier 237 Zeichen + Anführungszeichen (unter DOS 3.3 1 Zeichen mehr!). Beim späteren READ-INPUT wird das führende Anführungszeichen nicht dem String zugewiesen. Vgl. STRING.SAVER.2 in Verbindung mit INPUT.LOADER.

3. Man verzichtet auf gar nichts und liest später mit der INALL-Routine anstelle von INPUT ein. Die eingelesenen Strings können gleichzeitig Anführungszeichen (auch am Anfang) sowie Kommas und Doppelpunkte enthalten. Auch führende Leerzeichen bleiben erhalten. Die maximale Stringlänge beträgt hier 238 Bytes, also ebenfalls 1 Byte weniger als unter DOS 3.3.

Die INALL-Routine ist völlig relokativ, kann sich also überall im Speicher befinden. Der Aufruf erfolgt in der Form
CALL A, Stringvariable
A steht für die Anfangsadresse, z.B. 768, und für die Stringvariable kann eine einfache Variable, z.B. X$, oder eine Array-Variable, z.B. X$ (100), eingesetzt werden. Man beachte im übrigen das Komma zwischen A und Stringvariable. INALL kann außerdem anstelle des normalen Nicht-Textfile-INPUT verwendet werden. Die INALL-Routine stammt von H. Grumser („Peeker", Heft 5/1985, S. 50).

STRING.SAVER.1

```
100 REM STRING.SAVER.1
110 PRINT CHR$ (4)"OPEN XXX": PRINT CHR$ (4)"WRITE XXX"
120 FOR X = 100 TO 103
130 PRINT X;: REM 3
140 FOR Y = 4 TO 230: PRINT "-";: NEXT Y
150 PRINT "12345678"
160 NEXT X
170 PRINT CHR$ (4)"CLOSE"
180 REM Hier funktioniert INPUT.LOADER und INALL.LOADER
190 REM 238 Bytes, d.h. 1 Byte weniger als unter DOS 3.3!
```

STRING.SAVER.2

```
100 REM STRING.SAVER.2
110 PRINT CHR$ (4)"OPEN XXX": PRINT CHR$ (4)"WRITE XXX"
120 FOR X = 100 TO 103
130 PRINT CHR$ (34);X;: REM 4
140 FOR Y = 5 TO 230: PRINT "-";: NEXT Y
150 PRINT ",:345678"
160 NEXT X
170 PRINT CHR$ (4)"CLOSE"
180 REM Hier funktioniert nur INPUT.LOADER
190 REM Anführungszeichen + 237 Bytes,
        d.h. 1 Byte weniger als unter DOS 3.3!
```

STRING.SAVER.3

```
100 REM STRING.SAVER.3
110 PRINT CHR$ (4)"OPEN XXX": PRINT CHR$ (4)"WRITE XXX"
120 FOR X = 100 TO 103
130 PRINT X;: REM 3
140 FOR Y = 4 TO 230: PRINT "-";: NEXT Y
150 PRINT ",:345678"
160 NEXT X
170 PRINT CHR$ (4)"CLOSE"
180 REM Hier funktioniert nur INALL.LOADER
190 REM 238 Bytes, d.h. 1 Byte weniger als unter DOS 3.3!
```

INPUT.LOADER

```
100 REM INPUT.LOADER
110 DIM S$(3)
120 PRINT CHR$ (4)"OPEN XXX": PRINT CHR$ (4)"READ XXX"
130 FOR X = 1 TO 3
140 INPUT S$(X)
150 NEXT
160 PRINT CHR$ (4)"CLOSE"
170 FOR X = 1 TO 3: PRINT S$(X): NEXT
```

INALL.LOADER

```
100 REM INALL.LOADER
110 DATA 32,6,227,32,190,222,32,227,223,32
120 DATA 108,221,133,133,132,134,32,44,213,200
130 DATA 32,233,227,76,154,218
140 RESTORE : FOR X = 768 TO 793: READ Y: POKE X,Y: NEXT
150 DIM S$(3)
160 PRINT CHR$ (4)"OPEN XXX": PRINT CHR$ (4)"READ XXX"
170 FOR X = 1 TO 3
180 CALL 768,S$(X)
190 NEXT
200 PRINT CHR$ (4)"CLOSE"
210 FOR X = 1 TO 3: PRINT S$(X): NEXT
```

2.6. Zahlen-Arrays als Binärfiles

Bei OPEN, WRITE, PRINT Z wird die Fließkommazahl Z zunächst in einen Zahlen-String umgewandelt, der dann auf die Diskette übertragen wird. Bei OPEN, READ, INPUT Z wird zuerst der Zahlen-String eingelesen und dann in die Fließkommazahl Z umgewandelt. Diese Konvertierungen beanspruchen Zeit, weshalb das Speichern bzw. Einlesen von Zahlen-Arrays relativ langsam ist. Das nachfolgende Programm ZAHLEN.ARRAY speichert und lädt deshalb Zahlen-Arrays (Fließkommazahlen oder Integerzahlen) als Speicherauszug-Binärfiles. Die DATA-Statements bilden ein Maschinenprogramm, das eine gekürzte und zugleich relokative Version der GETARYPT-Routine aus meinem „Apple Assembler", S. 170, darstellt, die in die zweite Hälfte des Eingabepuffers ab $0280 (= 640) gelegt wird. Der Aufruf erfolgt bei Fließkommazahlen-Arrays mit

CALL 640, A: REM A für A (n)

und bei Integerzahlen-Arrays mit

CALL 640, A%: REM A% für A% (n)

A bzw. A% müssen durch die tatsächlich verwendeten Array-Namen ersetzt werden, etwa Z1, XY% usw. Die Utility ist *nicht* für String-Arrays gedacht! Man beachte, daß der 7 Bytes umfassende Array-Kopf ebenfalls gespeichert bzw. eingeladen wird. Ein Überschreiben nachfolgender Arrays ist nicht möglich, wohl aber könnte man einen falschen Array einlesen. Deshalb beachte man, daß erstens für den eingelesenen Array dieselbe Dimensionierung gilt, daß nicht etwa ein Integer- mit einem Floating-Point-Number-Array verwechselt wird und daß die Routine nicht auf String-Arrays angewandt wird. Ferner beachte man, daß vor dem eigentlichen BSAVE/BLOAD keine *neue* Variable definiert werden darf, weil sich dadurch der Array im Speicher verschieben würde.

Der Geschwindigkeitsvorteil gegenüber dem konventionellen Zugriff auf Zahlen-Textfiles läßt sich den Demos ARRAY.ALT und ARRAY.NEU entnehmen.

ZAHLEN.ARRAY

```
100 HOME : PRINT "ZAHLEN.ARRAY-Demo": DIM A(1000)
110 FOR X = 1 TO 1000:A(X) = X + X: NEXT
120 F$ = "ARRAY1": GOSUB 1010
130 FOR X = 1 TO 1000:A(X) = X * X: NEXT
140 F$ = "ARRAY2": GOSUB 1010
150 PRINT "ARRAY1: X + X"
160 F$ = "ARRAY1": GOSUB 1040
170 FOR X = 1 TO 1000: PRINT X;" - ";A(X): NEXT
180 PRINT : PRINT "ARRAY1: X * X"
190 F$ = "ARRAY2": GOSUB 1040
200 FOR X = 1 TO 1000: PRINT X;" - ";A(X): NEXT
210 END
```

```
1000 REM *** GETARYPT.BSAVE ***
1010 GOSUB 1120: CALL 640,A
1020 GOSUB 1130: PRINT CHR$ (4)"BSAVE";F$;",A"G1;",L"G2: RETURN
1030 REM *** GETARYPT.BLOAD ***
1040 GOSUB 1120: CALL 640,A
1050 GOSUB 1130: PRINT CHR$ (4)"BLOAD";F$;",A"G1;",L"G2: RETURN
1060 DATA 32,190,222,32,217,247,165,155,141,250
1070 DATA 2,165,156,141,251,2,160,2,177,155
1080 DATA 141,252,2,200,177,155,141,253,2,24
1090 DATA 165,155,160,2,113,155,141,254,2,165
1100 DATA 156,200,113,155,141,255,2,206,254,2
1110 DATA 208,3,206,255,2,76,149,217
1120 RESTORE : FOR G1 = 640 TO 697: READ G2: POKE G1,G2: NEXT :
     RETURN : REM Ab $0280
1130 G1 = PEEK (762) + PEEK (763) * 256:G2 = PEEK (764) +
     PEEK (765) * 256: RETURN
1140 REM $02FA-$02FB=762-763=Anfangsadresse-LLHH
1150 REM $02FC-$02FD=764-765=Länge-LLHH
1160 REM $02FE-$02FF=766-767=Endadresse-LLHH
```

ARRAY.ALT

```
100 HOME : PRINT "ARRAY.ALT-Geschwindigkeitsdemo": DIM A(1000):
    FOR X = 1 TO 1000:A(X) = 12.34: NEXT :
    REM Hier 5 Bytes pro Zahl + Return
110 PRINT CHR$ (7)
120 F$ = "ARRAY1": GOSUB 190
130 F$ = "ARRAY2": GOSUB 190
140 F$ = "ARRAY1": GOSUB 210
150 F$ = "ARRAY2": GOSUB 210
160 PRINT CHR$ (7)
170 END
180 REM *** WRITE ***
190 PRINT CHR$ (4)"OPEN";F$: PRINT CHR$ (4)"WRITE";F$:
    FOR X = 1 TO 1000: PRINT A(X): NEXT :
    PRINT CHR$ (4)"CLOSE": RETURN
200 REM *** READ ***
210 PRINT CHR$ (4)"OPEN";F$: PRINT CHR$ (4)"READ";F$:
    FOR X = 1 TO 1000: INPUT A(X): NEXT :
    PRINT CHR$ (4)"CLOSE": RETURN
220 REM Bei Disk-II:  38 Sekunden
230 REM Bei RAM-Disk: 18 Sekunden
240 REM Geschwindigkeit hängt von durchschnittlicher
    Länge des Zahlenstrings ab!
```

Man beachte, daß ZAHLEN.ARRAY auch bei mehrdimensionalen Arrays funktioniert.

ARRAY.NEU

```
100 HOME : PRINT "ARRAY.NEU-Geschwindigkeitsdemo":
    DIM A(1000): FOR X = 1 TO 1000:A(X) = 12.34: NEXT :
    REM 5 Bytes pro Zahl
110 PRINT CHR$ (7)
120 F$ = "ARRAY1": GOSUB 190
130 F$ = "ARRAY2": GOSUB 190
140 F$ = "ARRAY1": GOSUB 220
150 F$ = "ARRAY2": GOSUB 220
160 PRINT CHR$ (7)
170 END
180 REM *** GETARYPT.BSAVE ***
190 GOSUB 300: CALL 640,A
200 GOSUB 310: PRINT CHR$ (4)"BSAVE";F$;",A"G1;",L"G2: RETURN
210 REM *** GETARYPT.BLOAD ***
220 GOSUB 300: CALL 640,A
230 GOSUB 310: PRINT CHR$ (4)"BLOAD";F$;",A"G1;",L"G2: RETURN
240 DATA 32,190,222,32,217,247,165,155,141,250
250 DATA 2,165,156,141,251,2,160,2,177,155
260 DATA 141,252,2,200,177,155,141,253,2,24
270 DATA 165,155,160,2,113,155,141,254,2,165
280 DATA 156,200,113,155,141,255,2,206,254,2
290 DATA 208,3,206,255,2,76,149,217
300 RESTORE : FOR G1 = 640 TO 697: READ G2: POKE G1,G2: NEXT : RETURN :
    REM Ab $0280
310 G1 = PEEK (762) + PEEK (763) * 256:G2 = PEEK (764) +
    PEEK (765) * 256: RETURN
320 REM Bei Disk-II:  17 Sekunden
330 REM Bei RAM-Disk:  1 Sekunde
```

GETARYPT

```
Durch DATA-Statements poken!
Daher Quellcode nicht auf Begleitdiskette zu diesem Buch

                1             ORG  $0280
                2     *
                3     * GETARYPT für ProDOS/US/11.05.85
                4     * ========
                5     *
                6     * GETARYPT sucht Anfangsadresse
                7     * des Array-Kopfes, dessen Name
                8     * bei TXTPTR steht, und legt
                9     * Adresse in LOWTR und LOWTR+1 ab.
                10    * Die danach berechneten Werte
                11    * können benutzt werden, um
                12    * den Array mit
```

```
                        13  * BSAVE NAME, A ANFADR, L LAENGE
                        14  * zu speichern sowie mit
                        15  * BLOAD NAME, A ANFADR, L LAENGE
                        16  * später zu laden.
                        17  * Beachte, daß nur Integer- und
                        18  * Real-Arrays, NICHT String-
                        19  * Arrays gespeichert werden
                        20  * können.
                        21  *
                        22  * Nach Getarypt zeigt Lowtr auf
                        23  * Byte 0 des Array-Kopfes,
                        24  * der bei eindimensionalen
                        25  * Arrays, z.B. DIM A (100),
                        26  * aus 7 Bytes 0-6 besteht:
                        27  *
                        28  * Byte 0-1: 1. und 2. Buchstabe
                        29  *           des Variable-Namens
                        30  * Byte 2-3: LL-HH-Offset zur
                        31  *           Anfangsadresse des
                        32  *           nächsten Arrays
                        33  * Byte 4:   01 für 1. Dimension
                        34  * Byte 5-6: HH-LL-Anzahl der
                        35  *           Elemente der 1.
                        36  *           Dimension.
                        37  *
                        38  LOWTR    EQU  $009B       ;-$009C
                        39  ANFADR   EQU  $02FA       ;762
                        40  LAENGE   EQU  $02FC       ;764
                        41  ENDADR   EQU  $02FE       ;766
                        42  DATA     EQU  $D995       ;zum ":"
                        43  CHKCOM   EQU  $DEBE       ;Komma
                        44  GETARYPT EQU  $F7D9
                        45  *
                        46  * Lowtr = Anfangsadresse
                        47  *
0280: 20 BE DE          48  ENTRY    JSR  CHKCOM      ;640
0283: 20 D9 F7          49           JSR  GETARYPT
0286: A5 9B             50           LDA  LOWTR
0288: 8D FA 02          51           STA  ANFADR
028B: A5 9C             52           LDA  LOWTR+1
028D: 8D FB 02          53           STA  ANFADR+1
                        54  *
                        55  * Offset = Länge
                        56  *
0290: A0 02             57           LDY  #2          ;Byte 2
0292: B1 9B             58           LDA  (LOWTR),Y
0294: 8D FC 02          59           STA  LAENGE
0297: C8                60           INY              ;Byte 3
0298: B1 9B             61           LDA  (LOWTR),Y
029A: 8D FD 02          62           STA  LAENGE+1
                        63  *
                        64  * Endadresse = Lowtr + Offset
                        65  *
029D: 18                66           CLC
```

```
029E: A5 9B     67            LDA  LOWTR
02A0: A0 02     68            LDY  #2          ;Byte 2
02A2: 71 9B     69            ADC  (LOWTR),Y
02A4: 8D FE 02  70            STA  ENDADR
02A7: A5 9C     71            LDA  LOWTR+1
02A9: C8        72            INY              ;Byte 3
02AA: 71 9B     73            ADC  (LOWTR),Y
02AC: 8D FF 02  74            STA  ENDADR+1
                75   *
                76   * Endadresse = Endadresse - 1
                77   *
02AF: CE FE 02  78            DEC  ENDADR
02B2: D0 03     79            BNE  EXIT
02B4: CE FF 02  80            DEC  ENDADR+1
                81   *
                82   * Txtptr auf ":" oder Eol
                83   *
02B7: 4C 95 D9  84   EXIT     JMP  DATA

58 Bytes
```

2.7. Simulierter MON-Befehl

Der MON-Befehl, der unter DOS 3.3 zur Überwachung von Input, Output und Commands (MON I, O, C) implementiert war, fehlt im BASIC.SYSTEM. Man kann ihn in bezug auf Input/Output ohne tiefgreifende, versionsabhängige Patches nur simulieren, indem man den READ- oder WRITE-Zustand vorübergehend mit einem nackten
PRINT CHR$ (4)
desaktiviert. Danach muß der READ- oder WRITE-Zustand durch erneutes READ bzw. WRITE reaktiviert werden. Das nachfolgende MON.DEMO zeigt, wie es gemacht wird.

MON.DEMO

```
100 REM MON.DEMO
110 REM Write + Print
120 HOME : PRINT "MON.WRITE.PRINT"
130 PRINT CHR$ (4)"OPEN XXX"
140 FOR X = 100 TO 200
150 PRINT CHR$ (4): PRINT X;"AAAAAAAAAA": REM Bildschirm
160 PRINT CHR$ (4)"WRITE XXX": PRINT X;"AAAAAAA": REM Diskette
170 NEXT
180 PRINT CHR$ (4);"CLOSE"
190 REM Read + Print
200 HOME : PRINT "MON.READ.PRINT"
210 PRINT CHR$ (4)"OPEN XXX"
220 FOR X = 100 TO 200
230 PRINT CHR$ (4)"READ XXX"
240 INPUT X$: REM Diskette
```

```
250 PRINT CHR$ (4): PRINT X$: REM Bildschirm
260 NEXT
270 PRINT CHR$ (4)"CLOSE"
280 HOME : PRINT "FINGER AUF RETURN-TASTE LASSEN"
290 FOR X = 1 TO 1000: NEXT
300 REM Read + Get
310 PRINT : PRINT "MON.READ.GET"
320 PRINT CHR$ (4)"OPEN XXX"
330 FOR X = 100 TO 200
340 PRINT CHR$ (4)"READ XXX"
350 INPUT X$: REM Diskette
360 PRINT CHR$ (4): PRINT X$: GET X$: REM Bildschirm
370 NEXT
380 PRINT CHR$ (4)"CLOSE"
390 REM Read + Input
400 HOME : PRINT "MON.READ.INPUT"
410 PRINT CHR$ (4)"OPEN XXX"
420 FOR X = 100 TO 200
430 PRINT CHR$ (4)"READ XXX"
440 INPUT X$: REM Diskette
450 PRINT CHR$ (4): PRINT X$: INPUT "";X$: REM Bildschirm
460 NEXT
470 PRINT CHR$ (4)"CLOSE"
480 REM Read + Print + Get ohne Return!
490 HOME : PRINT "MON.READ.PRINT.GET"
500 PRINT CHR$ (4)"OPEN XXX"
510 FOR X = 100 TO 200
520 PRINT CHR$ (4)"READ XXX"
530 INPUT X$: REM Diskette
540 PRINT CHR$ (4): PRINT X$;: GET X$: REM Bildschirm
550 NEXT
560 PRINT CHR$ (4)"CLOSE"
570 END
```

2.8. MON.COMMANDS

Der „MON C"-Befehl läßt sich mit dem nachfolgenden Maschinenprogramm simulieren, das mit

BRUN MON.COMMANDS

gestartet wird und voraussetzt, daß sich HIMEM *vor* dem Start bei $9600 befand. Um „MON C" aufzurufen, muß man in dem Applesoft-Programm an den entsprechenden Stellen das &-Zeichen einfügen, z.B.

1000 PRINT CHR$ (4); "OPEN XXX": &

Danach werden die PBITS- und FBITS-Werte sowie alle momentanen Parameter des letzten BASIC.SYSTEM-Befehls angezeigt. Man kann „&" im übrigen auch über die Tastatur eingeben.

MON.COMMANDS

```
BSAVE MON.COMMANDS, A$20B6, L366
```

```
1                ORG  $20B6
2     *
3     * MON.COMMANDS US/01.05.85
4     * ═══════════
5     *
6     * zeigt PBITS/FBITS-Werte an.
7     *
8     * Mit BRUN MON.COMMANDS starten
9     * Mit & oder CALL 38912 aufrufen
10    *
11    IND1     EQU  $FC
12    IND2     EQU  $FE
13    CSWL     EQU  $36
14    CSWH     EQU  $37
15    HIMEM    EQU  $73
16    AMPER    EQU  $03F5
17    PRBYTE   EQU  $FDDA
18    PRNTAX   EQU  $F941
19    COUT     EQU  $FDED
20    CROUT    EQU  $FD8E
21    VECTOUT  EQU  $BE30
22    GETBUFR  EQU  $BEF5
23    *
24    PUFFALT  EQU  $9600
25    PUFFNEU  EQU  $9400
26    ORIGIN   EQU  $9800
27    BASSYS   EQU  $9A00
28    *
29    PBITS    EQU  $BE54      ;-$BE55
30    FBITS    EQU  $BE56      ;-$BE57
31    *
32    * A = Anfangsadresse
33    * B = Byte-Offset
34    * E = Endadresse
35    * L = Länge
36    * S = Slot bei CAT, S6 usw.
37    * D = Drive
38    * F = Feld
39    * R = Record
40    * V = Volume-Nummer (ignoriert)
41    * § = At-Zeilennummer
42    * T = Type (TXT usw.)
43    * s = Slot bei PR#s
44    *
45    A_WERT   EQU  $BE58      ;-$BE59
46    B_WERT   EQU  $BE5A      ;-$BE5C
47    E_WERT   EQU  $BE5D      ;-$BE5E
48    L_WERT   EQU  $BE5F      ;-$BE60
49    S_WERT   EQU  $BE61
50    D_WERT   EQU  $BE62
```

```
                        51  F_WERT   EQU  $BE63       ;-$BE64
                        52  R_WERT   EQU  $BE65       ;-$BE66
                        53  V_WERT   EQU  $BE67
                        54  §_WERT   EQU  $BE68       ;-$BE69
                        55  T_WERT   EQU  $BE6A
                        56  s_WERT   EQU  $BE6B
                        57  *
                        58  *-----------------------------
                        59  *
20B6: A5 74             60          LDA  HIMEM+1
20B8: C9 96             61          CMP  #>PUFFALT  ;$9600
20BA: F0 05             62          BEQ  GETPAGE
20BC: A9 87             63  ERROR   LDA  #$87       ;Bell
20BE: 4C ED FD          64          JMP  COUT
                        65  *
                        66  * 2 Pages über GETPUFFER verlangen
                        67  *
20C1: A9 02             68  GETPAGE LDA  #2
20C3: 20 F5 BE          69          JSR  GETBUFR
20C6: B0 F4             70          BCS  ERROR
20C8: A5 74             71          LDA  HIMEM+1
20CA: C9 94             72          CMP  #>PUFFNEU  ;$9400
20CC: D0 EE             73          BNE  ERROR
                        74  *
                        75  * Programm nach $9800-$99FF
                        76  * verschieben.
                        77  * Zuvor Ampersand-Vektor setzen
                        78  *
20CE: A0 00             79          LDY  #0
20D0: A9 00             80          LDA  #$00       ;$2100
20D2: 85 FC             81          STA  IND1
20D4: A9 21             82          LDA  #$21
20D6: 85 FD             83          STA  IND1+1
20D8: A9 4C             84          LDA  #$4C       ;JMP
20DA: 8D F5 03          85          STA  AMPER
20DD: A9 00             86          LDA  #<ORIGIN
20DF: 85 FE             87          STA  IND2
20E1: 8D F6 03          88          STA  AMPER+1
20E4: A9 98             89          LDA  #>ORIGIN
20E6: 85 FF             90          STA  IND2+1
20E8: 8D F7 03          91          STA  AMPER+2
20EB: B1 FC             92  MOVE1   LDA  (IND1),Y
20ED: 91 FE             93          STA  (IND2),Y
20EF: C8                94          INY
20F0: D0 F9             95          BNE  MOVE1
20F2: E6 FD             96          INC  IND1+1
20F4: E6 FF             97          INC  IND2+1
20F6: B1 FC             98  MOVE2   LDA  (IND1),Y
20F8: 91 FE             99          STA  (IND2),Y
20FA: C8               100          INY
20FB: C0 23            101          CPY  #<ENDE-ANFANG-256
20FD: D0 F7            102          BNE  MOVE2
20FF: 60               103          RTS
                       104  *
```

```
                    105  *-----------------------------
                    106  *
                    107  *
                    108           ORG  ORIGIN
                    109  *
                    110  * COUT-Vektoren wegen
                    111  * BASIC.SYSTEM abhängen
                    112  *
9800: D8            113  ANFANG   CLD
9801: A5 36         114           LDA  CSWL
9803: 8D 21 99      115           STA  CSWLSAVE
9806: AD 30 BE      116           LDA  VECTOUT
9809: 85 36         117           STA  CSWL
980B: A5 37         118           LDA  CSWH
980D: 8D 22 99      119           STA  CSWHSAVE
9810: AD 31 BE      120           LDA  VECTOUT+1
9813: 85 37         121           STA  CSWH
                    122  *
                    123  * PBITS und FBITS ausgeben
                    124  *
9815: 20 8E FD      125           JSR  CROUT
9818: A2 00         126           LDX  #0
981A: BD D2 98      127  KOPF1    LDA  KOPF,X
981D: F0 06         128           BEQ  KOPF2
981F: 20 ED FD      129           JSR  COUT
9822: E8            130           INX
9823: D0 F5         131           BNE  KOPF1
9825: A9 D0         132  KOPF2    LDA  #"P"
9827: 20 ED FD      133           JSR  COUT
982A: AD 54 BE      134           LDA  PBITS
982D: 20 0A 99      135           JSR  BITS
9830: AD 55 BE      136           LDA  PBITS+1
9833: 20 0A 99      137           JSR  BITS
9836: 20 8E FD      138           JSR  CROUT
9839: A9 C6         139  KOPF3    LDA  #"F"
983B: 20 ED FD      140           JSR  COUT
983E: AD 56 BE      141           LDA  FBITS
9841: 20 0A 99      142           JSR  BITS
9844: AD 57 BE      143           LDA  FBITS+1
9847: 20 0A 99      144           JSR  BITS
984A: 20 8E FD      145           JSR  CROUT
984D: 20 8E FD      146           JSR  CROUT
                    147  *
                    148  * A, B usw. ausgeben
                    149  *
9850: A0 00         150           LDY  #$00
9852: 20 F1 98      151           JSR  LETTER1
9855: AD 59 BE      152           LDA  A_WERT+1
9858: AE 58 BE      153           LDX  A_WERT
985B: 20 01 99      154           JSR  AXKOMLET
985E: AD 5C BE      155           LDA  B_WERT+2
9861: 20 DA FD      156           JSR  PRBYTE
9864: AD 5B BE      157           LDA  B_WERT+1
9867: AE 5A BE      158           LDX  B_WERT
```

```
986A: 20 01 99  159          JSR  AXKOMLET
986D: AD 5E BE  160          LDA  E_WERT+1
9870: AE 5D BE  161          LDX  E_WERT
9873: 20 01 99  162          JSR  AXKOMLET
9876: AD 60 BE  163          LDA  L_WERT+1
9879: AE 5F BE  164          LDX  L_WERT
987C: 20 01 99  165          JSR  AXKOMLET
987F: AD 61 BE  166          LDA  S_WERT
9882: 20 DA FD  167          JSR  PRBYTE
9885: 20 04 99  168          JSR  KOMLET
9888: AD 62 BE  169          LDA  D_WERT
988B: 20 DA FD  170          JSR  PRBYTE
988E: 20 04 99  171          JSR  KOMLET
9891: AD 64 BE  172          LDA  F_WERT+1
9894: AE 63 BE  173          LDX  F_WERT
9897: 20 01 99  174          JSR  AXKOMLET
989A: AD 66 BE  175          LDA  R_WERT+1
989D: AE 65 BE  176          LDX  R_WERT
98A0: 20 01 99  177          JSR  AXKOMLET
98A3: AD 67 BE  178          LDA  V_WERT
98A6: 20 DA FD  179          JSR  PRBYTE
98A9: 20 04 99  180          JSR  KOMLET
98AC: AD 69 BE  181          LDA  §_WERT+1
98AF: AE 68 BE  182          LDX  §_WERT
98B2: 20 01 99  183          JSR  AXKOMLET
98B5: AD 6A BE  184          LDA  T_WERT
98B8: 20 DA FD  185          JSR  PRBYTE
98BB: 20 04 99  186          JSR  KOMLET
98BE: AD 6B BE  187          LDA  s_WERT
98C1: 20 DA FD  188          JSR  PRBYTE
98C4: 20 8E FD  189          JSR  CROUT
                190  *
                191  * COUT-Vektoren wieder herstellen
                192  *
98C7: AD 21 99  193          LDA  CSWLSAVE
98CA: 85 36     194          STA  CSWL
98CC: AD 22 99  195          LDA  CSWHSAVE
98CF: 85 37     196          STA  CSWH
98D1: 60        197          RTS              ;Exit
                198  *
                199  *------------------------------
                200  *
                201  * PBITS- Beispiel
                202  * ------------
                203  *
                204  * FEDCBA98 76543210
                205  * 00001101 11110100 BSAVE
                206  * psincT21 ABEL§SFR
                207  * PBITS    PBITS+1  HHLL-Format
                208  *
                209  * PBITS
                210  * p = Präfix suchen (CAT/PREFIX)
                211  * s = nur Slot (PR#/IN#)
                212  * i = Immediate Mode unzulässig
```

```
                        213  * n = Name kann ganz entfallen
                        214  * c = CREATE, falls inexistent
                        215  * T = Typ (TXT usw.) zulässig
                        216  * 2 = n2 erforderlich (RENAME)
                        217  * 1 = n1 erforderlich
                        218  * PBITS+1
                        219  * A = Anfangsadresse zulässig
                        220  * B = Byte-Offset zulässig
                        221  * E = Endadresse zulässig
                        222  * L = Länge zulässig
                        223  * § = Zeilennunmmer zulässig
                        224  * S = Slot/Drive zulässig
                        225  * F = Feld zulässig
                        226  * R = Record zulässig
                        227  *
98D2: A0 F0 F3          228  KOPF     ASC  " psincT21ABEL§SFR"
98D5: E9 EE E3 D4 B2 B1 C1 C2
98DD: C5 CC C0 D3 C6 D2
98E3: 8D 00             229           HEX  8D00
98E5: C1 C2 C5          230  LETTER   ASC  "ABELSDFRVXTs"
98E8: CC D3 C4 C6 D2 D6 C0 D4
98F0: F3
                        231  *
                        232  * Buchstabe, Komma und Hexzahl
                        233  *
98F1: B9 E5 98          234  LETTER1  LDA  LETTER,Y
98F4: C8                235           INY
98F5: 4C ED FD          236  LETTER2  JMP  COUT
                        237  *
98F8: A9 AC             238  KOMMA    LDA  #","
98FA: 20 ED FD          239           JSR  COUT
98FD: A9 A0             240           LDA  #$A0
98FF: D0 F4             241           BNE  LETTER2
                        242  *
9901: 20 41 F9          243  AXKOMLET JSR  PRNTAX
9904: 20 F8 98          244  KOMLET   JSR  KOMMA
9907: 4C F1 98          245           JMP  LETTER1
                        246  *
                        247  * 8 Bits ausgeben
                        248  *
990A: A2 08             249  BITS     LDX  #8
990C: 0A                250  BITS1    ASL
990D: 8D 20 99          251           STA  BITMASKE
9910: A9 B1             252           LDA  #"1"
9912: B0 02             253           BCS  BITS2
9914: A9 B0             254           LDA  #"0"
9916: 20 ED FD          255  BITS2    JSR  COUT
9919: AD 20 99          256           LDA  BITMASKE
991C: CA                257           DEX
991D: D0 ED             258           BNE  BITS1
991F: 60                259           RTS
9920: 00                260  BITMASKE HEX  00
                        261  *
9921: 00                262  CSWLSAVE HEX  00
```

```
9922: 00        263  CSWHSAVE HEX  00
                264  *
9923: EA        265  ENDE     NOP

366 Bytes
```

2.9. FP.COMMAND

Der FP-Befehl unter DOS 3.3 schaltete nicht nur von Integer-Basic nach Applesoft-Basic um, sondern bewirkte auch eine Normalisierung gewisser Zeiger. Unser FP.COMMAND, der zugleich ein Beispiel für eine relokative BASIC.SYSTEM-Befehlserweiterung darstellt, wird mit

BRUN FP.COMMAND

gestartet, wobei es im Gegensatz zu MON.COMMANDS gleichgültig ist, ob sich bereits weitere Programme oberhalb von HIMEM befinden oder nicht (Wäre man sowohl an MON.COMMANDS als auch an FP.COMMAND interessiert, so könnte man erst MON.COMMANDS und dann FP.COMMAND BRUNnen.)

Danach gibt man bei Bedarf

FP

über die Tastatur ein. Auch

PRINT CHR$ (4) "FP"

aus einem Programm heraus würde akzeptiert.

Im einzelnen bewirkt der FP-Befehl folgendes:

1. Der Speicher $0300 bis $03CF und $0800 bis HIMEM - 1 wird gelöscht, d.h. auf 0 gesetzt.
2. Der TXTTAB-Zeiger (vgl. Kapitel 2.2) wird wieder auf $0801 gesetzt und es wird ein Applesoft-NEW ausgeführt.

Dagegen wird im Gegensatz zum DOS-3.3-FP-Befehl HIMEM *nicht* normalisiert, weil sich sonst der FP-Befehl selbst zerstören würde.

FP.COMMAND

```
BSAVE FP.COMMAND, A$2000, L160

                1            ORG  $2000
                2    *
                3    * FP.COMMAND/US/01.05.85
                4    * ═══════════
                5    *
                6    * Simulierter FP-Befehl mit
                7    * zusätzlichem Löschen des
                8    * freien RAM-Bereichs als
                9    * externer BASIC.SYSTEM-Befehl.
```

```
                        10  *
                        11  CHRGET   EQU  $B1
                        12  TXTTAB   EQU  $67        ;-$68
                        13  HIMEM    EQU  $73        ;-$74
                        14  IND      EQU  $CE        ;-$CF
                        15  DOSWARM  EQU  $03D0
                        16  *
                        17  DOSCMD   EQU  $BE03      ;JMP LLHH
                        18  EXTRNCMD EQU  $BE06      ;JMP LLHH
                        19  *
                        20  * Die nachfolgenden 3 Labels
                        21  * werden bei diesem Befehl
                        22  * nicht benutzt, da keine
                        23  * zusätzlichen Parameter nach
                        24  * "FP" vorkommen.
                        25  *
                        26  XTRNADR  EQU  $BE50      ;LLHH
                        27  XLEN     EQU  $BE52      ;LEN-1
                        28  XCNUM    EQU  $BE53      ;0=extern
                        29  *
                        30  VPATH1   EQU  $BE6C      ;LLHH
                        31  PBITS    EQU  $BE54      ;-$BE55
                        32  GETBUFR  EQU  $BEF5
                        33  *
                        34  NEW      EQU  $D64B
                        35  ZPSTUFF  EQU  $F10B
                        36  CROUT    EQU  $FD8E
                        37  BELL     EQU  $FF3A
                        38  *
                        39  *-------------------------------
                        40  *
                        41  * Beispiel:
                        42  * ------
                        43  *
                        44  * 1. Vor  JSR GETBUFR
                        45  *    HIMEM:          $9600
                        46  *    I/O-Puffer:     $9600-$99FF
                        47  *    BACIS.SYSTEM:   $9A00-$BEFF
                        48  *    A-Register      #$01
                        49  * 2. Nach JSR GETBUFR
                        50  *    HIMEM:          $9500
                        51  *    I/O-Puffer:     $9500-$98FF
                        52  *    A-Register:     #$99
                        53  *    FP-Befehl:      $9900-$99FF
                        54  *    BASIC.SYSTEM:   $9A00-$BEFF
                        55  *
                        56  * 1 Page über GETBUFR anfordern.
                        57  * Wenn BCC, dann ist alles okay
                        58  * und A-Register enthält High-Byte
                        59  * der Pufferadresse.
                        60  *
2000: A9 01             61          LDA  #$01       ;1 Page
2002: 20 F5 BE          62          JSR  GETBUFR
2005: 90 03             63          BCC  MOVE
2007: 4C 3A FF          64          JMP  BELL       ;Fehler!
```

```
                        65  *
                        66  * FP-Programm moven (z.B. nach
                        67  * $9900 ff); volle Page moven!
                        68  *
200A: 85 CF             69  MOVE     STA  IND+1       ;HH
200C: A0 00             70           LDY  #0
200E: 84 CE             71           STY  IND
2010: B9 30 20          72  MOVE1    LDA  FP,Y
2013: 91 CE             73           STA  (IND),Y
2015: C8                74           INY
2016: D0 F8             75           BNE  MOVE1
                        76  *
                        77  * Alte EXTRNCMD-Adresse
                        78  * in FP-Programm retten (NICHTFP)
                        79  * Neue EXTRNCMD-Adresse auf
                        80  * Beginn von FP-Programm setzen
                        81  *
2018: AD 07 BE          82           LDA  EXTRNCMD+1 ;LL
201B: A0 05             83           LDY  #5
201D: 91 CE             84           STA  (IND),Y    ;LL
201F: AD 08 BE          85           LDA  EXTRNCMD+2 ;HH
2022: C8                86           INY
2023: 91 CE             87           STA  (IND),Y    ;HH
2025: A5 CE             88           LDA  IND
2027: 8D 07 BE          89           STA  EXTRNCMD+1 ;LL
202A: A5 CF             90           LDA  IND+1
202C: 8D 08 BE          91           STA  EXTRNCMD+2 ;HH
202F: 60                92           RTS
                        93  *
                        94  *-------------------------------
                        95  *
                        96  * Mit CLD anfangen, damit
                        97  * BASIC.SYSTEM "zufrieden" ist.
                        98  *
2030: D8                99  FP       CLD
2031: 18                100          CLC              ;$9900
2032: 90 06             101          BCC  FP?
                        102 *
                        103 * NICHTFP = FP + 4, z.B. $9904
                        104 *
2034: 4C FF FF          105 NICHTFP  JMP  $FFFF       ;Dummy
                        106 *
                        107 * Wenn "FP" nicht gefunden wurde,
                        108 * mit Carry set zum alten
                        109 * EXTRNCMD springen, so daß ein
                        110 * ggf. bereits installierter
                        111 * anderer EXTRNCMD abgearbeitet
                        112 * werden kann. Falls kein anderer
                        113 * EXTRNCMD "ge-daisy-chained" ist,
                        114 * erfolgt durch BASIC.SYSTEM
                        115 * Syntax-Error-Meldung.
                        116 *
2037: 38                117 NICHTFP1 SEC            ;nein!
2038: B0 FA             118          BCS  NICHTFP
```

```
                    119  *
                    120  * VPATH1 ist Zeiger (LLHH) zur
                    121  * momentanen Befehlszeile, die
                    122  * mit Länge-Byte beginnt und
                    123  * dann "FP" enthalten sollte.
                    124  *
203A: AD 6C BE      125  FP?       LDA  VPATH1
203D: 85 CE         126          STA  IND
203F: AD 6D BE      127          LDA  VPATH1+1
2042: 85 CF         128          STA  IND+1
2044: A0 01         129          LDY  #1          ;nach Länge
2046: B1 CE         130          LDA  (IND),Y
2048: C9 46         131          CMP  #'F'        ;Bit 7 off!
204A: D0 EB         132          BNE  NICHTFP1
204C: C8            133          INY
204D: B1 CE         134          LDA  (IND),Y
204F: C9 50         135          CMP  #'P'
2051: D0 E4         136          BNE  NICHTFP1
                    137  *
                    138  * Wenn unser externer Befehl
                    139  * zusätzliche Parameter hätte,
                    140  * so würde man diese jetzt
                    141  * auswerten.
                    142  *
                    143  * TXTTAB auf $0801 setzen
                    144  *
2053: A9 01         145  CLEAR1   LDA  #$01
2055: 85 67         146          STA  TXTTAB
2057: A9 08         147          LDA  #$08        ;$0800
2059: 85 68         148          STA  TXTTAB+1
                    149  *
                    150  * CHRGET + RANDOM kopieren
                    151  *
205B: A2 1C         152          LDX  #$1C
205D: BD 0B F1      153  CLEAR2   LDA  ZPSTUFF,X
2060: 95 B1         154          STA  CHRGET,X
2062: CA            155          DEX
2063: 10 F8         156          BPL  CLEAR2
                    157  *
                    158  * NEW-Befehl aufrufen:
                    159  * Initialisiert VARTAB-Pointer
                    160  * und führt CLEAR aus. Ferner
                    161  * wird Stackpointer auf $F8
                    162  * gesetzt: LDX #$F8 TXS
                    163  *
2065: 18            164          CLC              ;wegen ADC
2066: 20 4B D6      165          JSR  NEW
                    166  *
                    167  * Von HIMEM abwärts bis $0800
                    168  * Speicher löschen
                    169  *
2069: A0 00         170          LDY  #$00
206B: 84 CE         171          STY  IND
206D: A5 74         172          LDA  HIMEM+1     ;HH
206F: 85 CF         173          STA  IND+1
```

```
2071: C6 CF      174  CLEAR3   DEC  IND+1
2073: A5 CF      175           LDA  IND+1
2075: C5 68      176           CMP  TXTTAB+1
2077: 90 08      177           BCC  CLEAR5
2079: 98         178           TYA              ;A=0
207A: 91 CE      179  CLEAR4   STA  (IND),Y
207C: C8         180           INY
207D: D0 FB      181           BNE  CLEAR4
207F: F0 F0      182           BEQ  CLEAR3
                 183  *
                 184  * $0300-$03CF löschen: Page 3
                 185  *
2081: 98         186  CLEAR5   TYA
2082: A0 CF      187           LDY  #$CF
2084: 99 00 03   188  CLEAR6   STA  $0300,Y
2087: 88         189           DEY
2088: 10 FA      190           BPL  CLEAR6
                 191  *
                 192  * $0200-$02FF löschen: Puffer
                 193  *
208A: A0 00      194           LDY  #0
208C: 99 00 02   195  CLEAR7   STA  $0200,Y
208F: C8         196           INY
2090: D0 FA      197           BNE  CLEAR7
                 198  *
                 199  * $0100-$01F0 löschen: Stack
                 200  *
2092: 99 00 01   201  CLEAR8   STA  $0100,Y
2095: C8         202           INY
2096: C0 F1      203           CPY  #$F1
2098: D0 F8      204           BNE  CLEAR8
                 205  *
                 206  * Normalerweise würde man jetzt
                 207  * PBITS und PBITS+1 auf 0 setzen
                 208  * und mit CLC RTS den externen
                 209  * Befehl abschließen. Da FP
                 210  * jedoch den Speicher löscht,
                 211  * erfolgt hier der Exit über
                 212  * DOSWARM.
                 213  *
                 214  * LDA #0
                 215  * STA PBITS
                 216  * STA PBITS+1
                 217  * CLC
                 218  * RTS
                 219  *
209A: 20 8E FD   220           JSR  CROUT       ;Return
209D: 4C D0 03   221           JMP  DOSWARM

160 Bytes
```

2.10. BITMAP-Anzeige

Das Programm BITMAP, das mit
BLOAD BITMAP
CALL 768
gestartet wird, bewirkt einen Dump der System-Bit-Map-Werte, so daß man sich über die momentane Belegung des Speichers Klarheit verschaffen kann.
Bit 1 bedeutet belegte Page,
Bit 0 bedeutet freie Page.
In eine belegte Page kann man keine Programme BLOADen. Näheres s. Band 1, S. 54.

BITMAP-Dump
(mit Programm BITMAP erzeugt)

```
0000 (BF58) CF=11001111
0800 (BF59) 00=00000000
1000 (BF5A) 00=00000000
1800 (BF5B) 00=00000000
2000 (BF5C) 00=00000000
2800 (BF5D) 00=00000000
3000 (BF5E) 00=00000000
3800 (BF5F) 00=00000000
4000 (BF60) 00=00000000
4800 (BF61) 00=00000000
5000 (BF62) 00=00000000
5800 (BF63) 00=00000000
6000 (BF64) 00=00000000
6800 (BF65) 00=00000000
7000 (BF66) 00=00000000
7800 (BF67) 00=00000000
8000 (BF68) 00=00000000
8800 (BF69) 00=00000000
9000 (BF6A) 03=00000011
9800 (BF6B) FF=11111111
A000 (BF6C) FF=11111111
A800 (BF6D) FF=11111111
B000 (BF6E) FF=11111111
B800 (BF6F) C3=11000011
```

BITMAP

BSAVE BITMAP, A768, L93

Mit BLOAD BITMAP einladen.
Mit CALL 768 aufrufen.

```
                        1               ORG  $300
                        2    *
                        3    * System-Bit-Map anzeigen
                        4    * ========================
                        5    *
                        6    * von U. Stiehl/April 85
                        7    *
                        8    MLI        EQU  $BF00
                        9    COUT       EQU  $FDED
                       10    PRBYTE     EQU  $FDDA
                       11    *
0300: A0 58             12              LDY  #$58        ;$BF58
0302: A9 00             13              LDA  #$00        ;$0000
0304: 48                14              PHA              ;->
0305: F0 05             15              BEQ  MAP2        ;stets
                       16    *
                       17    * 2K-Bereich $0000, $0800, $1000
                       18    * $1800, $2000 usw. anzeigen.
                       19    *
0307: 68                20    MAP1       PLA
0308: 18                21              CLC
0309: 69 08             22              ADC  #$08
030B: 48                23              PHA
030C: 20 DA FD          24    MAP2       JSR  PRBYTE      ;HH
030F: A9 00             25              LDA  #$00
0311: 20 DA FD          26              JSR  PRBYTE      ;LL
0314: A9 A0             27              LDA  #$A0
0316: 20 ED FD          28              JSR  COUT
                       29    *
                       30    * MLI-Adresse anzeigen $BF58 usw.
                       31    *
0319: A9 A8             32              LDA  #"("
031B: 20 ED FD          33              JSR  COUT
031E: A9 BF             34              LDA  #>MLI
0320: 20 DA FD          35              JSR  PRBYTE      ;HH
0323: 98                36              TYA
0324: 20 DA FD          37              JSR  PRBYTE      ;LL
0327: A9 A9             38              LDA  #")"
0329: 20 ED FD          39              JSR  COUT
032C: A9 A0             40              LDA  #$A0
032E: 20 ED FD          41              JSR  COUT
                       42    *
```

```
                        43   * Hexbyte und Binärzahl anzeigen
                        44   * Bit 1 = Page belegt
                        45   * Bit 0 = Page frei
                        46   *
0331: B9 00 BF          47            LDA  MLI,Y
0334: 48                48            PHA
0335: 20 DA FD          49            JSR  PRBYTE
0338: A9 BD             50            LDA  #"="
033A: 20 ED FD          51            JSR  COUT
033D: 68                52            PLA
033E: A2 07             53            LDX  #7
0340: 48                54            PHA              ;-->
0341: 68                55   BIN1     PLA
0342: 0A                56            ASL
0343: 48                57            PHA
0344: A9 B0             58            LDA  #"0"
0346: 90 02             59            BCC  BIN2
0348: A9 B1             60            LDA  #"1"
034A: 20 ED FD          61   BIN2     JSR  COUT
034D: CA                62            DEX
034E: 10 F1             63            BPL  BIN1
0350: 68                64            PLA              ;<--
0351: A9 8D             65            LDA  #$8D
0353: 20 ED FD          66            JSR  COUT
                        67   *
                        68   * Erneuter Durchlauf bis $BF70
                        69   *
0356: C8                70            INY
0357: C0 70             71            CPY  #$70        ;$BF70
0359: 90 AC             72            BCC  MAP1
035B: 68                73            PLA              ;<--
035C: 60                74            RTS

93 Bytes
```

3. ProDOS-Utilities

Die nachfolgenden Utilities – meist Assemblerprogramme – decken fast alles ab, was im täglichen Umgang mit ProDOS an Hilfsprogrammen benötigt wird. Sie ersetzen im wesentlichen die Funktionen der ProDOS-Utilities FILER und CONVERT der Firma Apple, die beide erschreckend aufgebläht sind und damit unverhältnismäßig viel Speicherraum auf einer 140K-Diskette belegen (zusammen über 90 von 280 Blocks). So nimmt z.B. unser PROFID nur 1/15 des Speicherraums des FILERs ein und ist trotzdem noch ca. 40% schneller!
Alle Utilities sind ohne Tricks geschrieben und laufen deshalb auf *allen* bisher bekannten ProDOS-Versionen 1.0, 1.0.1, 1.0.2, 1.1 und 1.1.1 und BASIC.SYSTEM-Versionen 1.0 und 1.1.
Assemblerprogrammierer, die meine Utilities umgestalten oder erweitern wollen, mögen bitte den kommentierten Quellcode durcharbeiten. Nachfolgend beschränken wir uns auf das „praktische Handling" dieser Programme, deren Anwendung keinerlei Assemblerkenntnisse erfordert.

3.1. Catalogleseprogramme

3.1.1. EINTRAG.SUCHER

Dieses Programm zeigt in Form einer Unterroutine, wie der Dateieintrag bzw. die ganze Eintragzeile in einem Directory einer Stringvariablen zugewiesen werden kann. Das Programm wird mit RUN EINTRAG.SUCHER gestartet und erwartet dann die Eingabe des gewünschten bzw. zu suchenden Dateinamens. Es ist als Unterroutine für eigene Applesoft-Programme gedacht.

```
  100  INPUT "Dateieintrag:";S$
  110  GOSUB 61000: END
61000  REM Dateieintrag per Textfile suchen
61010  PRINT  CHR$ (4)"PREFIX": INPUT P$
61020  PRINT  CHR$ (4);"OPEN";P$;",TDIR"
61030  PRINT  CHR$ (4);"READ";P$
61040  LL =  LEN (S$)
61050  INPUT D$: INPUT K$: INPUT D$: REM Ignorieren!
61060  F = 0: INPUT D$: IF  MID$ (D$,2,LL) = S$ AND  MID$
       (D$,LL + 2,1) = " " THEN F = 1: GOTO 61080: REM
       F=Such-Flag
61070  IF D$ <  > "" THEN 61060
61080  PRINT  CHR$ (4);"CLOSE";P$
61090  PRINT P$;S$
61100  IF F = 0 THEN  PRINT "FEHLT"
61110  PRINT : RETURN
```

3.1.2. EINTRAG.ANALYSE

Dieses Programm, das mit RUN EINTRAG.ANALYSE gestartet wird, erwartet die Eingabe des Namens einer Datei, deren kompletter Directory-Eintrag dann in der internen Form angezeigt wird (s.S. 29). Es werden auch gelöschte und umbenannte Einträge angezeigt. (Wenn ein längerer Altname durch einen kürzeren Neunamen ersetzt wird, bleibt der Rest des Altnamens in Directory-Eintrag erhalten.) Zum Verständnis dieses Programms ziehe man Kapitel 1.4.3 aus Band 1 zu Rate, das die internen Bestandteile eines Dateieintrags detailliert erläutert.

```
100  REM EINTRAG.ANALYSE
110  REM U.Stiehl/1985
120  PRINT  CHR$ (4)"PR#3"
130  PRINT  CHR$ (4);"CATALOG"
140  INPUT "Dateieintrag: ";S$: IF S$ = "" THEN 140
150  HOME : GOSUB 530
160  REM 768 = Byte
170  REM CALL 769 = Hexbyte
180  REM CALL 776 = Low Nibble
190  REM CALL 783 = High Nibble
200  DATA 0,173,0,3,32,218,253,96,173,0
210  DATA 3,32,227,253,96,173,0,3,74,74
220  DATA  74,74,32,227,253,96
230  RESTORE
240  FOR X = 768 TO 793: READ Y: POKE X,Y: NEXT
250  REM Dateieintrag per Binärfile suchen
260  A = 8192:B = 0:L = 512: REM Address, Byte, Length
270  O = 4 + 39: REM Offset für nächsten Eintrag
280  PRINT  CHR$ (4)"BLOAD";P$;",TDIR,A"A;",B";B;",L"L
290  X = 1
```

```
300 ON  PEEK (A + 0 + X) < > ASC ( MID$ (S$,X,1)) GOTO
    310:X = X + 1: ON X < = LL GOTO 300: GOTO 340
310 0 = 0 + 39: IF 0 < 511 THEN 290
320 IF  PEEK (A + 2) < > 0 OR  PEEK (A + 3) < > 0 THEN
    B = B + L:0 = 4: GOTO 280: REM Nächster Block
330 PRINT "Nicht gefunden!": END
340 PRINT "Gefunden!": IF F = 0 THEN  PRINT "Jedoch
    gelöschte oder umbenannte Datei!"
350 REM Komponenten des Dateieintrags
360 PRINT : PRINT "Speichertyp: $";: POKE 768, PEEK (A +
    0 + 0): CALL 783: REM High Nibble
370 PRINT : PRINT "Namenslänge: $";: POKE 768, PEEK (A +
    0 + 0): CALL 776: REM Low Nibble
380 PRINT : PRINT "Dateiname:  ";: FOR X = (A + 0 + 1) TO
    (A + 0 + 15): PRINT " $";: POKE 768, PEEK (X): CALL
    769: NEXT
390 PRINT : PRINT "Dateityp:    $";: POKE 768, PEEK (A +
    0 + 16): CALL 769
400 PRINT : PRINT "Hauptzeiger: $";: POKE 768, PEEK (A +
    0 + 18): CALL 769: POKE 768, PEEK (A + 0 + 17): CALL
    769: PRINT "   ($HHLL)";
410 PRINT : PRINT "Blockanzahl: $";: POKE 768, PEEK (A +
    0 + 20): CALL 769: POKE 768, PEEK (A + 0 + 19): CALL
    769: PRINT "   ($HHLL)";
420 PRINT : PRINT "Dateilänge:  $";: POKE 768, PEEK (A +
    0 + 23): CALL 769: POKE 768, PEEK (A + 0 + 22): CALL
    769: POKE 768, PEEK (A + 0 + 21): CALL 769: PRINT "
    ($HHMMLL)";
430 PRINT : PRINT "Create-Dat:  $";: POKE 768, PEEK (A +
    0 + 24): CALL 769: PRINT " $";: POKE 768, PEEK (A + 0
    + 25): CALL 769
440 PRINT : PRINT "Create-Uhr:  $";: POKE 768, PEEK (A +
    0 + 26): CALL 769: PRINT " $";: POKE 768, PEEK (A + 0
    + 27): CALL 769
450 PRINT : PRINT "Version-Nr:  $";: POKE 768, PEEK (A +
    0 + 28): CALL 769: PRINT " $";: POKE 768, PEEK (A + 0
    + 29): CALL 769
460 PRINT : PRINT "Zugriff:     $";: POKE 768, PEEK (A +
    0 + 30): CALL 769
470 PRINT : PRINT "Zusatzinfo:  $";: POKE 768, PEEK (A +
    0 + 32): CALL 769: POKE 768, PEEK (A + 0 + 31): CALL
    769: PRINT "   ($HHLL)";
480 PRINT : PRINT "Mod-Datum:   $";: POKE 768, PEEK (A +
    0 + 33): CALL 769: PRINT " $";: POKE 768, PEEK (A + 0
    + 34): CALL 769
490 PRINT : PRINT "Mod-Uhr:     $";: POKE 768, PEEK (A +
    0 + 35): CALL 769: PRINT " $";: POKE 768, PEEK (A + 0
    + 36): CALL 769
500 PRINT : PRINT "Kopfzeiger:  $";: POKE 768, PEEK (A +
    0 + 38): CALL 769: POKE 768, PEEK (A + 0 + 37): CALL
    769: PRINT "   ($HHLL)";
510 END
520 REM Dateieintrag per Textfile suchen
530 PRINT  CHR$ (4)"PREFIX": INPUT P$
```

```
540  PRINT  CHR$ (4);"OPEN";P$;",TDIR"
550  PRINT  CHR$ (4);"READ";P$
560  LL =  LEN (S$)
570  INPUT D$: INPUT K$: INPUT D$: REM Ignorieren!
580  F = 0: INPUT D$: IF  MID$ (D$,2,LL) = S$ AND  MID$
     (D$,LL + 2,1) = " " THEN F = 1: GOTO 600: REM
     F=Such-Flag
590  IF D$ <  > "" THEN 580
600  PRINT  CHR$ (4);"CLOSE";P$
610  PRINT P$;S$
620  IF F = 1 THEN  PRINT : PRINT K$: PRINT D$
630  PRINT : RETURN
```

3.1.3. CAT.ARRAY

Dieses Programm, das mit RUN CAT.ARRAY gestartet wird, legt das durch das momentane Präfix definierte Directory in einem String-Array ab und zeigt dann eine *gewünschte Auswahl* der Dateieinträge an. Es ist als Unterroutine für eigene Applesoft-Programme gedacht. Sollte ein Directory voraussichtlich mehr als 51 Dateieinträge enthalten, so erhöhe man die DIM-Anweisung in Zeile 60020, da sonst nämlich nur die ersten 51 Dateieinträge eingelesen werden. Ferner beachte man Zeile 60130. Diese filtert z.Zt. alle BAS-Dateien aus dem Directory heraus, die hier als Beispiel in 40 Z/Z angezeigt werden. Durch Änderung und Erweiterung dieser Zeile kann man Eintragzeilen nach beliebigen Merkmalen selektieren.

```
  100  GOSUB 60000: END
60000  HOME : REM CAT.ARRAY
60010  PRINT  CHR$ (4)"PREFIX": INPUT P$
60020  DIM D$(51):D1 = 0:D2 = 51
60030  PRINT  CHR$ (4);"OPEN";P$;",TDIR"
60040  PRINT  CHR$ (4);"READ";P$
60050  INPUT D1$: INPUT D2$: INPUT D3$
60060  D1 = D1 + 1: IF D1 > D2 THEN 60080
60070  INPUT D$(D1): IF D$(D1) <  > "" THEN 60060
60080  INPUT D3$: PRINT  CHR$ (4);"CLOSE";P$
60090  PRINT "Diskette ";D1$: REM Prefix
60100  PRINT : PRINT  LEFT$ (D2$,39): REM Kopfzeile
60110  PRINT :D1 = D1 - 1
60120  FOR X = 1 TO D1
60130  IF  MID$ (D$(X),18,3) = "BAS" THEN  PRINT  LEFT$
       (D$(X),39)
60140  NEXT
60150  PRINT : PRINT  LEFT$ (D3$,39): REM Fußzeile
60160  RETURN
```

3.1.4. CAT.SAVER

Dieses Programm, das mit RUN CAT.SAVER gestartet wird, speichert das durch das momentane Präfix defininierte Directory in einem Textfile namens CAT.SAVE ab. Das Programm ließe sich erweitern, indem man es zunächst auf das Volume-Directory anwendet, sich dabei die DIR-Einträge merkt und dann bei den einzelnen Subdirectories fortfährt, die ihrerseits Subsubdirectories enthalten könnten. Auf diese Weise kann man sich dann eine Gesamtdatei aller Dateieinträge aller Directories eines Datenträgers anlegen. Dies ist namentlich bei Festplatten von Interesse.

```
100  HOME : REM CATSAVER
110  PRINT  CHR$ (4)"PREFIX": INPUT P$
120  DIM D$(51):D1 = 0:D2 = 51
130  PRINT  CHR$ (4);"OPEN";P$;",TDIR"
140  PRINT  CHR$ (4);"READ";P$
150  INPUT D1$: INPUT D2$: INPUT D3$
160  D1 = D1 + 1: IF D1 > D2 THEN 180
170  INPUT D$(D1): IF D$(D1) <  > "" THEN 160
180  INPUT D3$: PRINT  CHR$ (4);"CLOSE";P$
190  PRINT  CHR$ (4)"OPEN CAT.SAVE": PRINT  CHR$ (4)"WRITE
     CAT.SAVE"
200  PRINT D1$
210  PRINT : PRINT D2$
220  PRINT :D1 = D1 - 1
230  FOR X = 1 TO D1
240  PRINT D$(X)
250  NEXT
260  PRINT : PRINT D3$
270  PRINT  CHR$ (4)"CLOSE"
```

3.2. Dateileseprogramme

3.2.1. FILE.READ.ASC und ASC

Dieses Programm, das mit RUN FILE.READ.ASC gestartet wird und das seinerseits noch das kurze Maschinenprogramm ASC einliest, erwartet die Eingabe des Namens einer beliebig großen (Text)datei, die dann schubweise eingelesen und am Bildschirm in ASCII angezeigt wird. Wenn man eine beliebige Taste drückt, kann man die Anzeige anhalten; wenn man ESC drückt, die Anzeige abbrechen. Man beachte, daß Ctrl-Buchstaben mit Ausnahme von Return in inverse = sichtbare Großbuchstaben umgewandelt werden.

FILE.READ.ASC

```
100  PRINT  CHR$ (4);"PREFIX": INPUT P$: PRINT  CHR$
     (4);"BLOAD";P$;"ASC"
110  TEXT : HOME : INVERSE : PRINT "FILE.READ.ASC": NORMAL
     : PRINT : PRINT "START J/N ";: GET X$: PRINT X$: ON
     X$ = "J" OR X$ = "j" GOTO 120: ON X$ <  > "N" AND X$
     <  > "n" GOTO 110: HOME : END
120  PRINT : PRINT  CHR$ (4);"PREFIX": INPUT P$: PRINT
     CHR$ (4);"CAT";P$: PRINT : INPUT "DATEI:";S$: IF S$ =
     "" THEN 110
130  REM Dateieintrag suchen
140  PRINT  CHR$ (4);"OPEN";P$;",TDIR": PRINT  CHR$
     (Ignorieren!
160  F = 0: INPUT D$: IF  MID$ (D$,2,L) = S$ AND  MID$
     (D$,L + 2,1) = " " THEN F = 1: GOTO 180: REM
     F=Such-Flag
170  IF D$ <  > "" THEN 160
180  PRINT  CHR$ (4);"CLOSE";P$
190  IF F = 0 THEN  PRINT P$;S$;" FEHLT!": GET X$: GOTO
     110
200  REM Lesepuffer $1000-$9000
210  T$ = "T" +  MID$ (D$,18,3):LL =  VAL ( MID$ (D$,66,6))
220  HOME : PRINT  CHR$ (4)"FRE": POKE 773,0: POKE
     774,144: REM $9000
230  IF LL < 32768 THEN R = LL: GOSUB 270:B = 0:L = R:
     ON L = 0 GOTO 290: GOSUB 280: GOTO 290
240  B = 0:L = 32768: GOSUB 280
250  B = B + L: IF B + L < LL THEN  GOSUB 280: GOTO 250
260  R = LL - B: ON R = 0 GOTO 290: GOSUB 270:L = R: GOSUB 280: GOTO 290
270  POKE 773,R -  INT (R / 256) * 256: POKE 774, INT (R /
     256) + 16: RETURN
280  PRINT  CHR$ (4)"BLOAD";P$;S$;",";T$;",A$1000";",B";B;",L";L:
     CALL 768: RETURN
290  PRINT : GET X$: GOTO 110
```

ASC

```
BSAVE ASC, A768, L104
```

```
                    1              ORG  $300
                    2    *
                    3    * ASC  zu FILE.READ.ASC
                    4    * ===
                    5    *
                    6    * 12.04.85/U.Stiehl
                    7    *
0300: 4C 07 03      8              JMP  START
                    9    *
```

```
                        10   COUT     EQU  $FDED
                        11   SETINV   EQU  $FE80
                        12   SETNORM  EQU  $FE84
                        13   *
0303: 00 10             14   ANFADR   DA   $1000
0305: 00 90             15   ENDADR   DA   $9000
                        16   *
0307: AD 03 03          17   START    LDA  ANFADR
030A: 8D 2F 03          18            STA  LOAD+1
030D: AD 04 03          19            LDA  ANFADR+1
0310: 8D 30 03          20            STA  LOAD+2
0313: 4C 1E 03          21            JMP  INC2
                        22   *
0316: EE 2F 03          23   INC1     INC  LOAD+1
0319: D0 03             24            BNE  INC2
031B: EE 30 03          25            INC  LOAD+2
031E: 18                26   INC2     CLC
031F: AD 05 03          27            LDA  ENDADR
0322: ED 2F 03          28            SBC  LOAD+1
0325: AD 06 03          29            LDA  ENDADR+1
0328: ED 30 03          30            SBC  LOAD+2
032B: B0 01             31            BCS  LOAD
032D: 60                32   EXIT     RTS
                        33   *
032E: AD FF FF          34   LOAD     LDA  $FFFF      ;Dummy
0331: 09 80             35            ORA  #$80
0333: C9 FF             36            CMP  #$FF
0335: F0 DF             37            BEQ  INC1       ;Del!
0337: C9 8D             38            CMP  #$8D
0339: F0 0F             39            BEQ  DISPLAY    ;Return!
033B: C9 A0             40            CMP  #$A0
033D: B0 0B             41            BCS  DISPLAY
033F: 69 40             42            ADC  #$40
0341: 8D 67 03          43            STA  CHAR
0344: 20 80 FE          44            JSR  SETINV
0347: AD 67 03          45            LDA  CHAR
034A: 20 ED FD          46   DISPLAY  JSR  COUT
034D: 20 84 FE          47            JSR  SETNORM
0350: AD 00 C0          48            LDA  $C000
0353: 10 C1             49            BPL  INC1
0355: 2C 10 C0          50            BIT  $C010
0358: C9 9B             51            CMP  #$9B       ;ESC
035A: F0 D1             52            BEQ  EXIT
035C: AD 00 C0          53   WAIT     LDA  $C000
035F: 10 FB             54            BPL  WAIT
0361: 2C 10 C0          55            BIT  $C010
0364: 4C 16 03          56            JMP  INC1
0367: 00                57   CHAR     HEX  00

104 Bytes
```

3.2.2. FILE.READ.HEX und HEX

Dieses Programm entspricht in der Handhabung FILE.READ.ASC, doch wird nicht nur ein ASCII-, sondern auch ein Hex-Dump produziert. Ctrl-Buchstaben werden durch „." ersetzt. Deshalb kann die eingelesene Datei nach PR#1 auch ausgedruckt werden. Die Anzahl der Hexzahlen pro Zeile kann man wahlweise auf 8 oder 16 einstellen durch

BLOAD HEX
CALL -151
307:08 oder 10
Ctrl-C
BSAVE HEX

FILE.READ.HEX

```
100  PRINT  CHR$ (4);"PREFIX": INPUT P$: PRINT  CHR$
     (4);"BLOAD";P$;"HEX": POKE 775,8: REM Auch POKE
     775,16 für 16 Zahlen pro Zeile möglich
110  TEXT : HOME : INVERSE : PRINT "FILE.READ.HEX": NORMAL
     : PRINT : PRINT "START J/N ";: GET X$: PRINT X$: ON
     X$ = "J" OR X$ = "j" GOTO 120: ON X$ <  > "N" AND X$
     <  > "n" GOTO 110: HOME : END
120  PRINT : PRINT  CHR$ (4);"PREFIX": INPUT P$: PRINT
     CHR$ (4);"CAT";P$: PRINT : INPUT "DATEI:";S$: IF S$ =
     "" THEN 110
130  REM Dateieintrag suchen
140  PRINT  CHR$ (4);"OPEN";P$;",TDIR": PRINT  CHR$
     (4);"READ";P$:L =  LEN (S$)
150  INPUT D$: INPUT D$: INPUT D$: REM Ignorieren!
160  F = 0: INPUT D$: IF  MID$ (D$,2,L) = S$ AND  MID$
     (D$,L + 2,1) = " " THEN F = 1: GOTO 180: REM
     F=Such-Flag
170  IF D$ <  > "" THEN 160
180  PRINT  CHR$ (4);"CLOSE";P$
190  IF F = 0 THEN  PRINT P$;S$;" FEHLT!": GET X$: GOTO
     110
200  REM Lesepuffer $1000-$9000
210  T$ = "T" +  MID$ (D$,18,3):LL =  VAL ( MID$ (D$,66,6))
220  HOME : PRINT  CHR$ (4)"FRE": POKE 773,0: POKE
     774,144: REM $9000
230  IF LL < 32768 THEN R = LL: GOSUB 270:B = 0:L = R:
     ON L = 0 GOTO 290: GOSUB 280: GOTO 290
240  B = 0:L = 32768: GOSUB 280
250  B = B + L: IF B + L < LL THEN  GOSUB 280: GOTO 250
260  R = LL - B: ON R = 0 GOTO 290: GOSUB 270:L = R: GOSUB 280: GOTO 290
270  POKE 773,R -  INT (R / 256) * 256: POKE 774, INT (R /
     256) + 16: RETURN
280  PRINT  CHR$ (4)"BLOAD";P$;S$;",";T$;",A$1000";",B";B;",L";L:
     CALL 768: RETURN
290  PRINT : GET X$: GOTO 110
```

HEX

```
BSAVE HEX, A768, L147

                        1           ORG  $300
                        2   *
                        3   * HEX zu FILE.READ.HEX
                        4   * ===
                        5   *
                        6   IND     EQU  $CE        ;-$CF
                        7   HEXOUT  EQU  $FDDA
                        8   PRINT   EQU  $FDED
                        9   *
0300: 4C 08 03          10  INIT1   JMP  INIT2
                       11   *
                       12   * Anzahl der Bytes/Zeile sowie
                       13   * Anfangs- und Endadresse poken,
                       14   * dann Hexdump aufrufen
                       15   *
0303: 00 10            16   ANFADR  DA   $1000      ;LL-HH
0305: FF 1F            17   ENDADR  DA   $1FFF      ;LL-HH
0307: 10               18   ANZAHL  DFB  16         ;oder 8
                       19   *
0308: AD 03 03         20   INIT2   LDA  ANFADR
030B: 85 CE            21           STA  IND
030D: AD 04 03         22           LDA  ANFADR+1
0310: 85 CF            23           STA  IND+1
                       24   *
0312: 20 47 03         25   DUMP1   JSR  DUMP2
0315: AD 00 C0         26           LDA  $C000
0318: 10 0F            27           BPL  DUMP1A
031A: 2C 10 C0         28           BIT  $C010
031D: C9 9B            29           CMP  #$9B       ;ESC
031F: F0 25            30           BEQ  EXIT
0321: AD 00 C0         31   KEYWAIT LDA  $C000
0324: 10 FB            32           BPL  KEYWAIT
0326: 2C 10 C0         33           BIT  $C010
                       34   *
                       35   * Adresse um Anzahl erhöhen
                       36   *
0329: 18               37   DUMP1A  CLC
032A: AD 07 03         38           LDA  ANZAHL
032D: 65 CE            39           ADC  IND
032F: 85 CE            40           STA  IND
0331: A5 CF            41           LDA  IND+1
0333: 69 00            42           ADC  #0
0335: 85 CF            43           STA  IND+1
0337: B0 0E            44           BCS  DUMP2      ;Wrap!
                       45   *
                       46   * und mit Endadresse vergleichen
                       47   *
0339: 38               48           SEC
033A: AD 05 03         49           LDA  ENDADR
033D: E5 CE            50           SBC  IND
```

```
033F: AD 06 03  51              LDA  ENDADR+1
0342: E5 CF     52              SBC  IND+1
0344: B0 CC     53              BCS  DUMP1
0346: 60        54   EXIT       RTS              ;Exit
                55   *
                56   * $ + Adresse anzeigen
                57   *
0347: A9 A4     58   DUMP2      LDA  #"$"
0349: 20 ED FD  59              JSR  PRINT
034C: A5 CF     60              LDA  IND+1
034E: 20 DA FD  59              JSR  HEXOUT
0351: A5 CE     63              LDA  IND
0353: 20 ED FD  64              JSR  HEXOUT
0356: A9 A0     65              LDA  #$A0        ;Space
0358: 20 ED FD  66              JSR  PRINT
                67   *
                68   * Hex-Bytes anzeigen
                69   *
035B: A0 00     70              LDY  #0
035D: B1 CE     71   HEXPRT     LDA  (IND),Y
035F: 20 DA FD  72              JSR  HEXOUT
0362: A9 A0     73              LDA  #$A0
0364: 20 ED FD  74              JSR  PRINT
0367: C8        75              INY
0368: CC 07 03  76              CPY  ANZAHL
036B: 90 F0     77              BCC  HEXPRT
                78   *
036D: A9 A0     79              LDA  #$A0        ;Space
036F: 20 ED FD  80              JSR  PRINT
                81   *
                82   * Ascii-Buchstaben anzeigen
                83   *
0372: A0 00     84              LDY  #0
0374: B1 CE     85   ASCPRT     LDA  (IND),Y
                86   *
                87   * Ctrl-Buchstaben ignorieren
                88   *
0376: 09 80     89              ORA  #%10000000
0378: C9 A0     90              CMP  #$A0
037A: B0 02     91              BCS  ASCPRT1     ;>=$A0
037C: A9 AE     92              LDA  #"."        ;Ctrl="."
037E: C9 FF     93   ASCPRT1    CMP  #$FF        ;DEL="."
0380: D0 02     94              BNE  ASCPRT2
0382: A9 AE     95              LDA  #"."
0384: 20 ED FD  96   ASCPRT2    JSR  PRINT
0387: C8        97              INY
0388: CC 07 03  98              CPY  ANZAHL
038B: 90 E7     99              BCC  ASCPRT
                100  *
038D: A9 8D     101             LDA  #$8D        ;Return
038F: 20 ED FD  102             JSR  PRINT
0392: 60        103             RTS

147 Bytes
```

3.3. Dateikopierprogramme

3.3.1. PRODOS.FID.A und PRODOS.FID.O

Dieses Programm, das man mit RUN PRODOS.FID.A startet und das seinerseits ein Maschinenprogramm namens PRODOS.FID.O einlädt, ist ein kleines FID (analog zum DOS-3.3-FID bzw. ProDOS-FILER), das nur eine Einschränkung hat: Es können keine Dateien, die größer als 32K sind, kopiert werden. Alle Funktionen sind aus dem Menü ersichtlich (s. Programmzeile 200) und werden durch Eintippen des ersten Buchstabens aktiviert. Man beachte, daß man vor dem K(opierbefehl)
Präfix 1 = Ausgangsdiskette und
Präfix 2 = Zieldiskette definieren muß.
Nachdem die BASIC.SYSTEM-Version 1.1 nunmehr auch einen APPEND-BSAVE, d.h. einen BSAVE über den momentanen ENDFILE hinaus zuläßt, könnte man die Zeilen 670-700 sogar so abändern, daß beliebig große Dateien kopiert werden können.
PRODOS.FID.A ist nur für 2-Drive-Besitzer gedacht. Wenn man jedoch vor den Zeilen 380, 680 usw. entsprechende Meldungen einbaut, kann es auch von 1-Drive-Besitzern benutzt werden.
Dieses Programm zeigt wohl besser als alle anderen Programme dieses Buches, daß die BASIC.SYSTEM-Befehle erheblich leistungsfähiger als die DOS-3.3-Befehle sind, denn dort wäre ein überwiegend in Applesoft geschriebenes FID-Programm nicht denkbar gewesen.

3.3.2. PROFID.DEMO und PROFID

Dieses Maschinenprogramm, das mit BRUN PROFID gestartet wird, ersetzt den FILER. Es erfordert jedoch 2 Volumes (2 Drives oder 1 Drive + Profile/RAM-Disk usw.). Die implementierten Befehle sind aus dem Listing Zeile 12-44 zu ersehen. Zu Testzwecken wurde eine 110.000 Zeichen umfassende Datei jeweils kopiert mit
a) ProDOS-FILER (99s: ca. 2220 Bytes/s)
b) PROFID (73s: ca. 3010 Bytes/s)
c) DOS-3.3-FID (49s: ca. 4480 Bytes/s)
Bei seriöser Programmierung läßt sich somit unter ProDOS die Geschwindigkeit des alten FID nicht erreichen, da der schnelle MLI-Read-Befehl durch den sehr langsamen MLI-Write-Befehl entwertet wird.
Man beachte die Zeilen 98-123 des Assemblerlistings, die einen 32K-Kopierpuffer ausweisen, wobei PROFID normalerweise HIMEM bei $9600 erwartet. Wie erst nachträglich durch Experimente ermittelt werden konnte, wird die Kopiergeschwindigkeit durch eine Reduzierung des Kopierpuffers nicht gravierend beeinträchtigt. Für die o.g. 110.000-Zei-

chen-Datei gelten folgenden Kopierzeiten (Read + Write) als Anhaltspunkte:
$8000 Puffer = 73s
$4000 Puffer = 77s
$2000 Puffer = 83s
$1000 Puffer = 95s
$0E00 Puffer = 99s
Daraus folgt, daß erst bei einem Puffer von $0E00 die Geschwindigkeit des FILERs unterschritten wird. Die Puffergröße wird wie folgt gekürzt:
BLOAD PROFID
CALL -151
0C8F: 60 (für Puffergröße $6000 als Beispiel)
0C91: 00 (Bit-Map-Check desaktivieren)
Ctrl-C
BSAVE PROFID
Speziell für den Kopierbefehl gibt es einen zweiten Entry (CALL 3203), der es ermöglicht, über ein kurzes Applesoft-Programm namens PROFID.DEMO den Kopiervorgang bekannter Dateien zu automatisieren, z.B. um häufig benötigte Dateien auf die RAM-Disk zu übertragen. Da PROFID jedoch bei der Adresse 3200 (dezimal) beginnt, bleibt nur Platz für ca. 10 längere Pfadnamen. Bei vielen Dateien (Profile usw.) kann man sich jedoch behelfen, indem man aus dem ersten PROFID.DEMO heraus ein weiteres startet. Darüber hinaus kann man auch das eigentliche PROFID mit CALL 3200 aus einem Applesoft-Programm heraus aufrufen, wenn man mit
BLOAD PROFID
CALL -151
0C95: 60
Ctrl-C
BSAVE PROFID
den Exit-Sprung zum Warmstart desaktiviert.
Neben den BASIC.SYSTEM-Fehlernummern ($02-$15) und den MLI-Fehlernummern ($27-$5A) sind einige spezielle Kopierfehlernummern implementiert worden:
$80 = Syntax-Fehler
$81 = System-Bit-Map-Konflikt
$82 = Keine Datei, sondern Directory
$83 = Leerdatei (nackte CREATE-Datei)
$84 = Datei größer als ca. 8 Megabytes

PRODOS.FID.A

```
100  REM *** PRODOS.FID.A ***
110  PRINT  CHR$ (21): POKE 242,0: PR# 0: IN# 0: POKE
     49164,0: POKE 49166,0: CALL 39447
120  TEXT : HOME : INVERSE : PRINT "PRODOS.FID": PRINT
     "U.STIEHL85": NORMAL : PRINT : PRINT  CHR$ (4)"BRUN
     PRODOS.FID.O": GOTO 200
130  VTAB 23: HTAB 1: CALL  - 958: RETURN
140  VTAB 23: HTAB H: INPUT "";A1$: RETURN
150  VTAB 24: HTAB H: INPUT "";A2$: RETURN
160  P1$ = "/" + A1$: VTAB 3: HTAB 1: INVERSE : PRINT
     "P1";: PRINT  LEFT$ (P1$,37);: GOSUB 180: VTAB 23:
     HTAB 1: RETURN
170  P2$ = "/" + A2$: VTAB 4: HTAB 1: INVERSE : PRINT
     "P2";: PRINT  LEFT$ (P2$,37);: GOSUB 180: VTAB 24:
     HTAB 1: RETURN
180  P =  POS (0):P = 39 - P: FOR X = 1 TO P: PRINT " ";:
     NEXT : NORMAL : RETURN
190  X =  PEEK (222): PRINT : POKE 216,0: PRINT "FEHLER-NR.
     ";X;" "; CHR$ (7);: GET B$: RUN 210
200  TEXT : HOME : INVERSE : PRINT "C(AT   P(REFIX
     O(NLINE  M(AKE  R(ENAME": PRINT "L(OCK  U(NLOCK
     D(ELETE  K(OPY  E(ND   ": NORMAL : VTAB 6: POKE 34,4
210  CLEAR : LOMEM: 6144:O = 4992
220  PRINT : PRINT  CHR$ (4)"PREFIX": INPUT A1$:A1$ =
     MID$ (A1$,2,255): IF  RIGHT$ (A1$,1) = "/" THEN A1$ =
     LEFT$ (A1$, LEN (A1$) - 1)
230  GOSUB 160:A2$ = "": GOSUB 170: PRINT  CHR$ (4)"CAT"
240  ONERR  GOTO 190
250  REM *** BEFEHL ***
260  PRINT
270  PRINT  CHR$ (4)"FRE": GOSUB 130: PRINT "BEFEHL:";
280  GET B$:B = 0
290  B = B + 1: ON B$ =  MID$ ("CPOMRLUDKE",B,1) GOTO 300:
     ON B < 10 GOTO 290: GOTO 280
300  ON B GOTO 320,350,410,430,450,490,510,530,560: TEXT :
     HOME : POKE 214,0: PRINT  CHR$ (4)"PREFIX"P1$: END
310  REM *** C(AT ***
320  GOSUB 130: PRINT "CAT";P1$;:H = 5: GOSUB 140: IF A1$
     = "" THEN  PRINT  CHR$ (4)"CAT";P1$: GOTO 260
330  PRINT  CHR$ (4)"CAT/";A1$: GOTO 260
340  REM *** P(REFIX ***
350  GOSUB 130: VTAB 23: HTAB 1: PRINT "P1"P1$:H = 4:
     GOSUB 140: IF A1$ = "" THEN 370
360  PRINT  CHR$ (4)"PREFIX/"A1$: GOSUB 160
370  VTAB 24: HTAB 1: PRINT "P2"P2$;:H = 4: GOSUB 150: IF
     A2$ = "" THEN 390
380  PRINT  CHR$ (4)"PREFIX/"A2$: GOSUB 170
390  PRINT : PRINT  CHR$ (4)"PREFIX"P1$: GOTO 260
400  REM *** O(NLINE ***
410  PRINT : CALL O + 3: PRINT : PRINT : GOTO 260
```

```
420  REM *** M(AKE ***
430  GOSUB 130: VTAB 23: HTAB 1: PRINT "MAKE";P1$;"/";:H =
     POS (0) + 1: GOSUB 140: ON A1$ = "" GOTO 270: PRINT
     CHR$ (4)"CREATE";P1$;"/";A1$;",TDIR": GOTO 260
440  REM *** R(ENAME ***
450  GOSUB 130: VTAB 23: HTAB 1: PRINT "ALT";P1$;"/";:H =
     POS (0) + 1: GOSUB 140: ON A1$ = "" GOTO 270
460  VTAB 24: HTAB 1: PRINT "NEU";P1$;"/";:H =  POS (0) +
     1: GOSUB 150: ON A2$ = A1$ GOTO 270
470  PRINT : PRINT  CHR$
     (4)"RENAME";P1$;"/";A1$;",";P1$;"/";A2$: GOTO 260
480  REM *** L(OCK ***
490  B$ = "LOCK": GOTO 540
500  REM *** U(NLOCK ***
510  B$ = "UNLOCK": GOTO 540
520  REM *** D(ELETE ***
530  B$ = "DELETE"
540  GOSUB 130: VTAB 23: HTAB 1: PRINT B$;P1$;"/";:H =
     POS (0) + 1: GOSUB 140: ON A1$ = "" GOTO 270: PRINT
     CHR$ (4);B$;P1$;"/";A1$: PRINT : GOTO 260
550  REM *** K(OPY ***
560  IF P2$ = "/" THEN  VTAB 23: HTAB 1: PRINT "ERST
     PREFIX/P2 FESTLEGEN! "; CHR$ (7);: GET B$: GOTO 260
570  GOSUB 130: VTAB 23: HTAB 1: PRINT "VON ";P1$;"/";:H =
     POS (0) + 1: GOSUB 140: ON A1$ = "" GOTO 270: PRINT
     CHR$ (4)"VERIFY";P1$;"/";A1$
580  VTAB 24: HTAB 1: PRINT "NACH";P2$;"/";:H =  POS (0) +
     1: PRINT A1$;: GOSUB 150: IF A2$ = "" THEN A2$ = A1$
590  PRINT  CHR$ (4)"PREFIX";P2$
600  PRINT  CHR$ (4)"OPEN";P1$;",TDIR": PRINT  CHR$
     (4)"READ";P1$:X =  LEN (A1$)
610  INPUT B$: INPUT B$: INPUT B$
620  INPUT B$: IF  MID$ (B$,2,X) = A1$ AND  MID$ (B$,X +
     2,1) = " " GOTO 650
630  IF B$ < > "" THEN 620
640  B =  VAL ( MID$ (B$,67,5): IF B > 30720 THEN  PRINT
     "ZU GROSS "; CHR$ (7);: GET B$: GOTO 260
650  PRINT  CHR$ (4);"CLOSE";P1$
660  A1$ = P1$ + "/" + A1$:A2$ = P2$ + "/" + A2$
670  PRINT  CHR$ (4)"BLOAD";A1$;",T"; MID$
     (B$,18,3);",A6656"
680  PRINT  CHR$ (4)"BSAVE";A2$;",A6656,L"; MID$ (B$,67,5)
690  P = 5632:B =  LEN (A1$): FOR X = 1 TO B: POKE P + X,
     ASC ( MID$ (A1$,X,1)): NEXT : POKE P + X, ASC ("/"):
     POKE P,B + 1
700  P = P + 256:B =  LEN (A2$): FOR X = 1 TO B: POKE P +
     X, ASC ( MID$ (A2$,X,1)): NEXT : POKE P + X, ASC
     ("/"): POKE P,B + 1
710  CALL 0 + 6: GOTO 260
```

PRODOS.FID.0

```
BSAVE PRODOS.FID.0, A4992, L582

                        1               ORG  $1380       ;4992
                        2   *
                        3   * PRODOS.FID.0/U.Stiehl/1985
                        4   * ==========================
                        5   *
1380: 4C 89 13          6   J1          JMP  DATUM       ;4992
1383: 4C 8F 14          7   J2          JMP  ONLINE      ;4995
1386: 4C 49 15          8   J3          JMP  SETINFO     ;4998
                        9   *
                       10   * $0800-$15FF: FID + FID.0
                       11   * $1600-$16FF: Namen-Puffer 1
                       12   * $1700-$17FF: Namen-Puffer 2
                       13   * $1800-$19FF: LOMEM 6144
                       14   * $1A00-$91FF: Copy-Puffer
                       15   *
                       16   IND         EQU  $CE         ;-$CF
                       17   PROMPT      EQU  $33
                       18   NAME1       EQU  $1600       ;5632
                       19   NAME2       EQU  $1700       ;5888
                       20   PUFFER      EQU  $0200
                       21   DATE1       EQU  $BF90
                       22   PRINT       EQU  $FDED
                       23   GETLN       EQU  $FD6A
                       24   MLI         EQU  $BF00
                       25   BELL        EQU  $FF3A
                       26   HEXOUT      EQU  $FDDA
                       27   *
                       28   * Datumeingabe
                       29   * ============
                       30   *
1389: A5 33            31   DATUM       LDA  PROMPT
138B: 8D 7C 14         32               STA  PROMPT1
138E: A9 A0            33               LDA  #$A0
1390: 85 33            34               STA  PROMPT
1392: A2 00            35   GETLN0      LDX  #0
1394: BD 7D 14         36   GETLN1      LDA  PROMPT2,X
1397: F0 06            37               BEQ  GETLN2
1399: 20 ED FD         38               JSR  PRINT
139C: E8               39               INX
139D: D0 F5            40               BNE  GETLN1
139F: 20 6A FD         41   GETLN2      JSR  GETLN
13A2: 8A               42               TXA
13A3: D0 06            43               BNE  GETLN2A
13A5: AD 7C 14         44               LDA  PROMPT1
13A8: 85 33            45               STA  PROMPT
13AA: 60               46               RTS              ;Nur Rtn
                       47   *                 01234567
                       48   * Größer als 7: TT.MM.JJ ?
                       49   *
13AB: CA               50   GETLN2A     DEX
```

```
13AC: E0 07      51              CPX  #7
13AE: D0 E2      52              BNE  GETLN0      ;Fehler!
                 53   *
                 54   * Nur Ziffern und "." ?
                 55   *
13B0: BD 00 02   56   GETLN3     LDA  PUFFER,X
13B3: 29 7F      57              AND  #$7F
13B5: 9D 87 14   58              STA  DATE2,X
13B8: C9 2E      59              CMP  #'.'
13BA: F0 0D      60              BEQ  GETLN5
13BC: C9 30      61              CMP  #'0'
13BE: 90 D2      62              BCC  GETLN0      ;Fehler!
13C0: C9 3A      63              CMP  #':'
13C2: B0 CE      64              BCS  GETLN0      ;Fehler!
13C4: CA         65   GETLN4     DEX
13C5: 10 E9      66              BPL  GETLN3
13C7: 30 0A      67              BMI  CHECK
13C9: E0 02      68   GETLN5     CPX  #2
13CB: F0 F7      69              BEQ  GETLN4
13CD: E0 05      70              CPX  #5
13CF: F0 F3      71              BEQ  GETLN4
13D1: D0 BF      72              BNE  GETLN0      ;Fehler!
                 73   *
                 74   * Datum korrekt ?
                 75   *
13D3: A2 FF      76   CHECK      LDX  #$FF        ;Wraparoud
13D5: 20 4C 14   77              JSR  PACKEN
13D8: 8D 76 14   78              STA  TT          ;TAG
13DB: E8         79              INX              ;1.Punkt
13DC: 20 4C 14   80              JSR  PACKEN
13DF: 8D 77 14   81              STA  MM          ;MONAT
13E2: E8         82              INX              ;2.Punkt
13E3: 20 4C 14   83              JSR  PACKEN
13E6: 8D 78 14   84              STA  JJ          ;JAHR
                 85   *
                 86   * Jahr okay (0-99) ?
                 87   *
13E9: C9 64      88              CMP  #100
13EB: 90 02      89              BCC  JAHR
13ED: B0 A3      90              BCS  GETLN0
13EF: C9 00      91   JAHR       CMP  #0
13F1: B0 02      92              BCS  MONAT1
13F3: 90 9D      93              BCC  GETLN0
                 94   *
                 95   * Monat okay (1-12) ?
                 96   *
13F5: AD 77 14   97   MONAT1     LDA  MM
13F8: C9 01      98              CMP  #1
13FA: B0 02      99              BCS  MONAT2
13FC: 90 94     100              BCC  GETLN0
13FE: C9 0D     101   MONAT2     CMP  #13
1400: 90 02     102              BCC  TAG1
1402: B0 8E     103              BCS  GETLN0
                104   *
```

```
                        105  * Tag okay (nach Liste) ?
                        106  *
1404: AA                107  TAG1     TAX
1405: CA                108          DEX
1406: AD 76 14          109          LDA  TT
1409: C9 01             110          CMP  #1
140B: B0 02             111          BCS  TAG2
140D: 90 83             112          BCC  GETLN0
140F: DD 6A 14          113  TAG2     CMP  TAGMON,X
1412: 90 05             114          BCC  POKEN1
1414: F0 03             115          BEQ  POKEN1
1416: 4C 92 13          116          JMP  GETLN0
                        117  *
                        118  * In Prodos Global Page poken
                        119  * ---------------------------
                        120  *
                        121  * $BF91    $BF90
                        122  * Poke1    Poke2
                        123  * FEDCBA98 76543210
                        124  * JJJJJJJM MMMTTTTT
                        125  *
1419: AD 78 14          126  POKEN1   LDA  JJ
141C: 0A                127           ASL
141D: 8D 7A 14          128           STA  POKE1
1420: AD 77 14          129           LDA  MM
1423: 0A                130           ASL
1424: 0A                131           ASL
1425: 0A                132           ASL
1426: 0A                133           ASL
1427: 0A                134           ASL
1428: 8D 79 14          135           STA  TEMP
142B: 90 03             136           BCC  POKEN2
142D: EE 7A 14          137           INC  POKE1
1430: 18                138  POKEN2   CLC
1431: AD 79 14          139           LDA  TEMP
1434: 6D 76 14          140           ADC  TT
1437: 8D 7B 14          141           STA  POKE2
                        142  *
                        143  * "Low Byte first", d.h.
                        144  * erst MM/TT, dann JJ/MM
                        145  *
143A: AD 7B 14          146           LDA  POKE2
143D: 8D 90 BF          147           STA  DATE1
1440: AD 7A 14          148           LDA  POKE1
1443: 8D 91 BF          149           STA  DATE1+1
                        150  *
1446: AD 7C 14          151           LDA  PROMPT1
1449: 85 33             152           STA  PROMPT
144B: 60                153           RTS
                        154  *
                        155  * Packen in 1 Byte
                        156  *
144C: E8                157  PACKEN   INX
144D: BD 87 14          158           LDA  DATE2,X
```

```
1450: 29 0F      159           AND  #$0F
1452: 0A         160           ASL               ;mal 2
1453: 8D 79 14   161           STA  TEMP
1456: 0A         162           ASL               ;mal 2
1457: 0A         163           ASL               ;mal 2
1458: 18         164           CLC
1459: 6D 79 14   165           ADC  TEMP
145C: 8D 79 14   166           STA  TEMP
145F: E8         167           INX
1460: BD 87 14   168           LDA  DATE2,X
1463: 29 0F      169           AND  #$0F
1465: 18         170           CLC
1466: 6D 79 14   171           ADC  TEMP
1469: 60         172           RTS
                 173  *
                 174  * Tage pro Monat maximal
                 175  *
146A: 1F         176  TAGMON    DFB  31          ;Januar
146B: 1D         177           DFB  29          ;Februar
146C: 1F         178           DFB  31          ;März
146D: 1E         179           DFB  30          ;April
146E: 1F         180           DFB  31          ;Mai
146F: 1E         181           DFB  30          ;Juni
1470: 1F         182           DFB  31          ;Juli
1471: 1F         183           DFB  31          ;August
1472: 1E         184           DFB  30          ;September
1473: 1F         185           DFB  31          ;Oktober
1474: 1E         186           DFB  30          ;November
1475: 1F         187           DFB  31          ;Dezember
                 188  *
1476: 00         189  TT        HEX  00          ;TT
1477: 00         190  MM        HEX  00          ;MM
1478: 00         191  JJ        HEX  00          ;JJ
1479: 00         192  TEMP      HEX  00
                 193  *
147A: 00         194  POKE1     HEX  00          ;JJ+MM
147B: 00         195  POKE2     HEX  00          ;MM+TT
                 196  *
147C: 00         197  PROMPT1   HEX  00          ;saven!
147D: D4 D4 AE   198  PROMPT2   ASC  "TT.MM.JJ:"
1480: CD CD AE CA CA BA
1486: 00         199           HEX  00
1487: 54 54 2E   200  DATE2     ASC  'TT.MM.JJ'
148A: 4D 4D 2E 4A 4A
                 201  *
                 202  ********************************
                 203  *
                 204  * Online
                 205  * ======
                 206  *
148F: A9 00      207  ONLINE    LDA  #<NAME1
1491: 85 CE      208           STA  IND
1493: A9 16      209           LDA  #>NAME1
1495: 85 CF      210           STA  IND+1
                 211  *
```

```
                        212  * 256-Byte-Puffer löschen
                        213  * Status-Register auf Null!
                        214  *
1497: A0 00             215           LDY  #0
1499: 84 48             216           STY  $48          ;P-Reg.
149B: 98                217           TYA
149C: 91 CE             218  ONLINE1  STA  (IND),Y
149E: C8                219           INY
149F: D0 FB             220           BNE  ONLINE1
                        221  *
14A1: B9 3F 15          222  ONLINE2  LDA  ONLINE4,Y
14A4: F0 06             223           BEQ  ONLINE3
14A6: 20 ED FD          224           JSR  PRINT
14A9: C8                225           INY
14AA: D0 F5             226           BNE  ONLINE2
                        227  *
                        228  * On Line ausführen
                        229  *
14AC: 20 00 BF          230  ONLINE3  JSR  MLI
14AF: C5                231           HEX  C5           ;Online
14B0: B7 14             232           DA   COUNT1
14B2: F0 07             233           BEQ  LOOP0
14B4: 4C 39 15          234           JMP  ERROR1
                        235  *
14B7: 02                236  COUNT1   HEX  02
14B8: 00                237           HEX  00           ;Alle Units
14B9: 00 16             238           DA   NAME1
                        239  *
                        240  * Endmarker erreicht?
                        241  *
14BB: A0 00             242  LOOP0    LDY  #0
14BD: B1 CE             243           LDA  (IND),Y
14BF: D0 06             244           BNE  LOOP1
14C1: C8                245           INY
14C2: B1 CE             246           LDA  (IND),Y
14C4: D0 01             247           BNE  LOOP1
14C6: 60                248           RTS               ;fertig
                        249  *
14C7: A9 8D             250  LOOP1    LDA  #$8D
14C9: 20 ED FD          251           JSR  PRINT
14CC: A0 00             252           LDY  #0
                        253  *
                        254  * Slot anzeigen
                        255  *
14CE: A9 D3             256           LDA  #"S"
14D0: 20 ED FD          257           JSR  PRINT
14D3: B1 CE             258           LDA  (IND),Y
14D5: 29 70             259           AND  #%01110000
14D7: 4A                260           LSR
14D8: 4A                261           LSR
14D9: 4A                262           LSR
14DA: 4A                263           LSR
14DB: 09 B0             264           ORA  #$B0         ;0
14DD: 20 ED FD          265           JSR  PRINT        ;Slot
```

```
14E0: A9 AC     266           LDA  #","
14E2: 20 ED FD  267           JSR  PRINT
14E5: A9 A0     268           LDA  #$A0
14E7: 20 ED FD  269           JSR  PRINT
14EA: B1 CE     270           LDA  (IND),Y
14EC: 29 0F     271           AND  #%00001111
14EE: 8D 48 15  272           STA  LENGTH
                273  *
                274  * Drive anzeigen
                275  *
14F1: A9 C4     276           LDA  #"D"
14F3: 20 ED FD  277           JSR  PRINT
14F6: B1 CE     278           LDA  (IND),Y
14F8: 10 04     279           BPL  LOOP2
14FA: A9 B2     280           LDA  #"2"        ;D2
14FC: D0 02     281           BNE  LOOP3
14FE: A9 B1     282  LOOP2    LDA  #"1"        ;D1
1500: 20 ED FD  283  LOOP3    JSR  PRINT
1503: A9 BA     284           LDA  #":"
1505: 20 ED FD  285           JSR  PRINT
                286  *
1508: AD 48 15  287           LDA  LENGTH      ;Len = 0
150B: F0 17     288           BEQ  LOOP6       ;Fehler!
150D: A9 AF     289           LDA  #"/"
150F: 20 ED FD  290           JSR  PRINT
                291  *
                292  * Name anzeigen
                293  *
1512: C8        294  LOOP4    INY
1513: CC 48 15  295           CPY  LENGTH
1516: 90 02     296           BCC  LOOP5
1518: D0 0F     297           BNE  LOOP7
                298  *
151A: B1 CE     299  LOOP5    LDA  (IND),Y
151C: 09 80     300           ORA  #$80
151E: 20 ED FD  301           JSR  PRINT
1521: 4C 12 15  302           JMP  LOOP4
                303  *
                304  * Name leer, d.h. Diskette
                305  * nicht eingelegt: I/O-Fehler
                306  *
1524: A9 BF     307  LOOP6    LDA  #"?"
1526: 20 ED FD  308           JSR  PRINT
                309  *
                310  * Nächster Name
                311  *
1529: 18        312  LOOP7    CLC
152A: A5 CE     313           LDA  IND
152C: 69 10     314           ADC  #16
152E: 85 CE     315           STA  IND
1530: A5 CF     316           LDA  IND+1
1532: 69 00     317           ADC  #0
1534: 85 CF     318           STA  IND+1
1536: 4C BB 14  319           JMP  LOOP0
```

```
                        320  *
1539: 20 DA FD          321  ERROR1   JSR  HEXOUT
153C: 4C 3A FF          322           JMP  BELL
                        323  *
153F: 8D                324  ONLINE4  HEX  8D
1540: CF CE CC          325           ASC  "ONLINE"
1543: C9 CE C5
1546: 8D 00             326           HEX  8D00
1548: 00                327  LENGTH   HEX  00
                        328  *
                        329  ********************************
                        330  *
                        331  * Get/Set File-Info
                        332  * ================
                        333  *
                        334  * Altname laden, dessen Parameter
                        335  * (Access/Filetype/Auxtype)
                        336  * zwischenspeichern, dann Neuname
                        337  * laden und mit geänderten Para-
                        338  * metern erneut speichern:
                        339  *
1549: A9 0A             340  SETINFO  LDA  #$0A        ;Altname
154B: 8D B0 15          341           STA  CNTA
154E: A9 00             342           LDA  #<NAME1
1550: 8D B1 15          343           STA  NAMEA
1553: A9 16             344           LDA  #>NAME1
1555: 8D B2 15          345           STA  NAMEA+1
1558: 20 00 BF          346           JSR  MLI
155B: C4                347           HEX  C4           ;Getinfo
155C: B0 15             348           DA   CNTA
155E: D0 D9             349           BNE  ERROR1
                        350  *
1560: AD B3 15          351           LDA  ACCESSA      ;saven
1563: 8D C2 15          352           STA  ACCESSB
1566: AD B4 15          353           LDA  FILETYPA
1569: 8D C3 15          354           STA  FILETYPB
156C: AD B5 15          355           LDA  AUXTYPEA
156F: 8D C4 15          356           STA  AUXTYPEB
1572: AD B6 15          357           LDA  AUXTYPEA+1
1575: 8D C5 15          358           STA  AUXTYPEB+1
                        359  *
1578: A9 00             360           LDA  #<NAME2      ;Neuname
157A: 8D B1 15          361           STA  NAMEA
157D: A9 17             362           LDA  #>NAME2
157F: 8D B2 15          363           STA  NAMEA+1
1582: 20 00 BF          364           JSR  MLI
1585: C4                365           HEX  C4           ;Getinfo
1586: B0 15             366           DA   CNTA
1588: D0 AF             367           BNE  ERROR1
                        368  *
158A: AD C2 15          369           LDA  ACCESSB      ;loaden
158D: 8D B3 15          370           STA  ACCESSA
1590: AD C3 15          371           LDA  FILETYPB
1593: 8D B4 15          372           STA  FILETYPA
```

```
1596: AD C4 15  373           LDA  AUXTYPEB
1599: 8D B5 15  374           STA  AUXTYPEA
159C: AD C5 15  375           LDA  AUXTYPEB+1
159F: 8D B6 15  376           STA  AUXTYPEA+1
                377  *
15A2: A9 07     378           LDA  #$07
15A4: 8D B0 15  379           STA  CNTA
15A7: 20 00 BF  380           JSR  MLI
15AA: C3        381           HEX  C3          ;Setinfo
15AB: B0 15     382           DA   CNTA
15AD: D0 8A     383           BNE  ERROR1
15AF: 60        384           RTS
                385  *
15B0: 00        386  CNTA     HEX  00
15B1: 00 50     387  NAMEA    DA   $5000
15B3: 00        388  ACCESSA  HEX  00
15B4: 00        389  FILETYPA HEX  00
15B5: 00 00     390  AUXTYPEA HEX  0000
15B7: 00        391  STORAGEA HEX  00
15B8: 00 00     392  BLOCKSA  HEX  0000
15BA: 00 00     393  MDATEA   HEX  0000
15BC: 00 00     394  MTIMEA   HEX  0000
15BE: 00 00     395  CDATEA   HEX  0000
15C0: 00 00     396  CTIMEA   HEX  0000
                397  *
15C2: 00        398  ACCESSB  HEX  00
15C3: 00        399  FILETYPB HEX  00
15C4: 00 00     400  AUXTYPEB HEX  0000
582 Bytes
```

PROFID.DEMO

```
10  PRINT  CHR$ (21): PRINT  CHR$ (4)"BLOAD PROFID": POKE
    242,0: PR# 0: IN# 0
20  POKE 766,1: POKE 767,3
30  CLEAR : FOR X = 1 TO  PEEK (766): READ X$: NEXT : FOR
    X = 1 TO  LEN (X$): POKE 511 + X, ASC ( MID$
    (X$,X,1)): NEXT : POKE 511 + X,13: CALL 3203
40  POKE 766, PEEK (766) + 1: ON  PEEK (766) < = PEEK
    (767) GOTO 30: CLEAR : CALL 39447
50  DATA "COPY/D1/AAA,/D2/AAA","COPY/D1/BBB,/D2/BBB",
    "COPY/D1/CCC,/D2/CCC"
60  REM PROFID.DEMO für 3 Dateien
```

PROFID

```
BSAVE PROFID, A3200, L1387

                1             ORG  3200        ;$0C80
                2  *
                3  * PROFID (ProDOS File Developer)
                4  * ======
                5  *
                6  * von U.Stiehl, 07.04.1985
                7  *
```

```
8     * Mit BRUN PROFID oder -PROFID
9     * oder aus einem Basic-Programm
10    * mit CALL 3200 starten.
11    *
12    * BASIC.SYSTEM-Befehle
13    * --------------------
14    *
15    * CATALOG  Mit Präfix oder
16    * CAT      mit Ss, Dd usw.
17    * CREATE   als Parameter,
18    * DELETE   d.h. übliche Syntax.
19    * LOCK     Kleinbuchstaben und
20    * UNLOCK   Leertasten sind
21    * PREFIX   erlaubt
22    * PR#
23    * RENAME
24    * VERIFY
25    *
26    * ONLINE   Zusätzlicher Befehl,
27    *          ohne Parameter
28    *
29    * Copy-Befehl
30    * -----------
31    *
32    * COPY/DIR1/FILE1,/DIR2/FILE2
33    *
34    * Bei Copy sind keine Leertasten
35    * erlaubt. Außerdem müssen voll-
36    * ständige Pfadnamen in obiger
37    * Form eingegeben werden,
38    * getrennt durch ein Komma.
39    *
40    * Exit-Befehle
41    * ------------
42    *
43    * a) END       = Basic-Modus
44    * b) -PROGRAMM = Programmstart
45    *
46    IND      EQU  $00CE      ;Online
47    PROMPT   EQU  $0033
48    PREG     EQU  $0048
49    PUFFER   EQU  $0200      ;512
50    DOSWARM  EQU  $03D0
51    DOSCMD   EQU  $BE03
52    GETLN    EQU  $FD6A
53    PRBYTE   EQU  $FDDA
54    COUT     EQU  $FDED
55    CROUT    EQU  $FD8E
56    HOME     EQU  $FC58
57    BELL1A   EQU  $FBDD
58    PRERR    EQU  $FF2D
59    SETINV   EQU  $FE80
60    SETNORM  EQU  $FE84
61    MLI      EQU  $BF00
62    LEVEL    EQU  $BF94
```

```
                        63   *
0C80: 4C AC 0C          64   START1   JMP  PROFID      ;normal
                        65   *
                        66   * Vorher Copy-Befehl in Puffer
                        67   * poken und dann mit CALL 3203
                        68   * aufrufen. Beispiel:
                        69   *
                        70   * 10 READ X$
                        71   * 20 FOR X = 1 TO LEN (X$)
                        72   * 30 POKE 511+X,ASC(MID$(X$,X,1))
                        73   * 40 NEXT X: POKE 511+X,13
                        74   * 50 CALL 3203
                        75   * 60 DATA "COPY/US/FILER,/RAM/FILER"
                        76   * 70 CLEAR: REM Stringpuffer
                        77   *    wird überschrieben!
                        78   *
0C83: A0 03             79   START2   LDY  #3          ;extern
0C85: 8C D6 0D          80            STY  KONST
0C88: 20 CB 0C          81            JSR  MENU0
0C8B: 4C 2A 0E          82            JMP  COPY1
                        83   *
                        84   * Speicherverteilung
                        85   * ------------------
                        86   *
                        87   * $0800-$0C7F: Basic-Programm
                        88   * $0C80-$11FF: PROFID (3200)
                        89   * $1200-$15FF: IO-Readpuffer
                        90   * $1600-$19FF: IO-Writepuffer
                        91   * $1A00-$99FF: Datenpuffer $8000
                        92   * $9A00-$BEFF: BASIC.SYSTEM
                        93   *
                        94   PUFREAD  EQU  $1200
                        95   PUFWRITE EQU  $1600
                        96   PUFBEG   EQU  $1A00
                        97   *
                        98   * Bei Bedarf BITMAP-Flag auf 0
                        99   * und PUFLENMM herabsetzen,
                       100   * falls vor HIMEM ein anderes
                       101   * Maschinenprogramm liegt.
                       102   * Für meinen "PRODOS.EDITOR",
                       103   * der im Bereich $8400-$99FF
                       104   * liegt, würde z.B. gelten:
                       105   *
                       106   *  $8400 = 33792 PUFEND + 1
                       107   * -$1A00 =  6656 PUFBEG
                       108   * ----------------------
                       109   *  $6A00 = 27136:256=106=$6A
                       110   *
                       111   * Man würde dann also BITMAP
                       112   * auf $00 und PUFLENMM auf $6A
                       113   * setzen.
                       114   *
0C8E: 00                115   PUFLEN   HEX  00          ;LL = 0
0C8F: 80                116   PUFLENMM HEX  80          ;MM > 0
0C90: 00                117            HEX  00          ;HH = 0
```

```
                      118  *
                      119  * System-Bit-Map-Flag
                      120  * 0 = kein Check
                      121  * 1 = Check
                      122  *
0C91: 01              123  BITMAP   HEX  01
                      124  *
                      125  * "END"-Exit (auch Del-Taste)
                      126  *
0C92: 20 A5 0C        127  ENDE     JSR  PATCH2
                      128  *
                      129  * "NOP" (=$EA) durch RTS (=$60)
                      130  * ersetzen, falls PROFID aus
                      131  * laufendem (!) Applesoftprogramm
                      132  * mit CALL 3200 aufgerufen wurde.
                      133  *
0C95: EA              134           NOP              ;RTS
0C96: 4C D0 03        135           JMP  DOSWARM
                      136  *
                      137  * "-"-Exit (z.B. -PRODOS)
                      138  *
0C99: 20 A5 0C        139  SYSTEM   JSR  PATCH2
0C9C: 4C 03 BE        140           JMP  DOSCMD
                      141  *
0C9F: A5 33           142  PATCH1   LDA  PROMPT
0CA1: 8D AB 0C        143           STA  PROMPT1
0CA4: 60              144           RTS
0CA5: AD AB 0C        145  PATCH2   LDA  PROMPT1
0CA8: 85 33           146           STA  PROMPT
0CAA: 60              147           RTS
0CAB: 00              148  PROMPT1  HEX  00
                      149  *
                      150  * PROFID-Start
                      151  * ------------
                      152  *
0CAC: 20 9F 0C        153  PROFID   JSR  PATCH1
0CAF: 20 CB 0C        154           JSR  MENU0
0CB2: A2 00           155           LDX  #0
0CB4: BD 6E 0D        156  TAB1     LDA  TABELLE,X
0CB7: 30 0C           157           BMI  TAB2        ;Bit 7 on
0CB9: 48              158           PHA
0CBA: 20 8E FD        159           JSR  CROUT
0CBD: 68              160           PLA
0CBE: CD BD 0D        161           CMP  JUMPANZ
0CC1: F0 3B           162           BEQ  BEFEHL2
0CC3: D0 03           163           BNE  TAB3
0CC5: 20 ED FD        164  TAB2     JSR  COUT
0CC8: E8              165  TAB3     INX
0CC9: D0 E9           166           BNE  TAB1        ;stets
                      167  *
```

```
0CCB: 20 58 FC  168  MENU0    JSR  HOME
0CCE: 20 80 FE  169           JSR  SETINV
0CD1: A2 00     170           LDX  #0
0CD3: BD E2 0C  171  MENU1    LDA  MENU3,X
0CD6: F0 06     172           BEQ  MENU2
0CD8: 20 ED FD  173           JSR  COUT
0CDB: E8        174           INX
0CDC: D0 F5     175           BNE  MENU1
0CDE: 20 84 FE  176  MENU2    JSR  SETNORM
0CE1: 60        177           RTS
                178  *
0CE2: 8D        179  MENU3    HEX  8D
0CE3: AA A0 D0  180           ASC  "* PROFID *"
0CE6: D2 CF C6 C9 C4 A0 AA
0CED: 8D        181           HEX  8D
0CEE: D5 AE D3  182           ASC  "U.STIEHL85"
0CF1: D4 C9 C5 C8 CC B8 B5
0CF8: 8D 8D 00  183           HEX  8D8D00
                184  *
0CFB: 20 DD FB  185  BEFEHL1  JSR  BELL1A
                186  *
                187  * Prompt + Status-Register
                188  *
0CFE: A9 BA     189  BEFEHL2  LDA  #":"
0D00: 85 33     190           STA  PROMPT
0D02: A9 00     191           LDA  #0          ;Status
0D04: 85 48     192           STA  PREG
0D06: 48        193           PHA
0D07: 28        194           PLP
                195  *
                196  * Nach GETLN Befehl im PUFFER
                197  *
0D08: 20 6A FD  198           JSR  GETLN
0D0B: AD 00 02  199           LDA  PUFFER
                200  *
                201  * "Del" und "-" vorab
                202  *
0D0E: C9 FF     203           CMP  #$FF        ;Del
0D10: F0 80     204           BEQ  ENDE
0D12: C9 AD     205           CMP  #"-"        ;Strich
0D14: F0 83     206           BEQ  SYSTEM
                207  *
                208  * Weniger als 3 Stellen ("CAT")?
                209  *
0D16: E0 03     210           CPX  #3
0D18: 90 E4     211           BCC  BEFEHL2     ;nur Rtn
                212  *
                213  * Befehlstabelle durchsuchen
                214  * Danach Y-Reg. Puffer-Pointer
                215  *
0D1A: A2 FF     216           LDX  #$FF
0D1C: A0 FF     217           LDY  #$FF
0D1E: E8        218  BEFEHL3  INX
0D1F: C8        219  BEFEHL4  INY
0D20: BD 6E 0D  220           LDA  TABELLE,X
```

```
0D23: CD BD 0D  221           CMP  JUMPANZ
0D26: F0 D3     222           BEQ  BEFEHL1     ;Ende
0D28: D9 00 02  223           CMP  PUFFER,Y
0D2B: F0 F1     224           BEQ  BEFEHL3
0D2D: 18        225           CLC
0D2E: 69 20     226           ADC  #$20        ;A->a
0D30: D9 00 02  227           CMP  PUFFER,Y
0D33: F0 E9     228           BEQ  BEFEHL3
0D35: B9 00 02  229           LDA  PUFFER,Y
0D38: C9 A0     230           CMP  #$A0        ;Space
0D3A: F0 E3     231           BEQ  BEFEHL4
0D3C: BD 6E 0D  232           LDA  TABELLE,X
0D3F: CD BD 0D  233           CMP  JUMPANZ
0D42: 90 0D     234           BCC  BEFEHL6     ;00-02
0D44: A0 FF     235           LDY  #$FF
0D46: E8        236 BEFEHL5   INX
0D47: BD 6E 0D  237           LDA  TABELLE,X
0D4A: CD BD 0D  238           CMP  JUMPANZ
0D4D: B0 F7     239           BCS  BEFEHL5
0D4F: 90 CD     240           BCC  BEFEHL3
                241 *
                242 * Indirekten Jump poken
                243 *
0D51: 0A        244 BEFEHL6   ASL
0D52: AA        245           TAX
0D53: BD BE 0D  246           LDA  JUMPTAB,X
0D56: 8D 61 0D  247           STA  JUMPER+1
0D59: E8        248           INX
0D5A: BD BE 0D  249           LDA  JUMPTAB,X
0D5D: 8D 62 0D  250           STA  JUMPER+2
                251 *
                252 * LLHH werden gepokt
                253 *
0D60: 4C FF FF  254 JUMPER    JMP  $FFFF       ;Dummy
                255 *
                256 * Normaler ProDOS-Befehl
                257 *
0D63: 20 03 BE  258 PRODOS    JSR  DOSCMD
0D66: 90 03     259           BCC  ERNEUT
0D68: 20 16 0E  260           JSR  ERROR
0D6B: 4C FE 0C  261 ERNEUT    JMP  BEFEHL2     ;Exit
                262 *
0D6E: C3 C1 D4  263 TABELLE   ASC  "CAT"
0D71: 00        264           HEX  00
0D72: C3 C1 D4  265           ASC  "CATALOG"
0D75: C1 CC CF C7
0D79: 00        266           HEX  00
0D7A: C3 D2 C5  267           ASC  "CREATE"
0D7D: C1 D4 C5
0D80: 00        268           HEX  00
0D81: C4 C5 CC  269           ASC  "DELETE"
0D84: C5 D4 C5
0D87: 00        270           HEX  00
0D88: CC CF C3  271           ASC  "LOCK"
0D8B: CB
```

```
0D8C: 00          272           HEX  00
0D8D: D5 CE CC    273           ASC  "UNLOCK"
0D90: CF C3 CB
0D93: 00          274           HEX  00
0D94: D0 D2 C5    275           ASC  "PREFIX"
0D97: C6 C9 D8
0D9A: 00          276           HEX  00
0D9B: D0 D2 A3    277           ASC  "PR#"
0D9E: 00          278           HEX  00
0D9F: D2 C5 CE    279           ASC  "RENAME"
0DA2: C1 CD C5
0DA5: 00          280           HEX  00
0DA6: D6 C5 D2    281           ASC  "VERIFY"
0DA9: C9 C6 D9
0DAC: 00          282           HEX  00
0DAD: C3 CF D0    283           ASC  "COPY"
0DB0: D9
0DB1: 01          284           HEX  01
0DB2: CF CE CC    285           ASC  "ONLINE"
0DB5: C9 CE C5
0DB8: 02          286           HEX  02
0DB9: C5 CE C4    287           ASC  "END"
0DBC: 03          288           HEX  03
0DBD: 04          289 JUMPANZ   HEX  04          ;Endmark
                  290 *
0DBE: 63 0D       291 JUMPTAB   DA   PRODOS      ;00
0DC0: C6 0D       292           DA   COPY        ;01
0DC2: 49 11       293           DA   ONLINE      ;02
0DC4: 92 0C       294           DA   ENDE        ;03
                  295 *
                  296 * Copy - Interner Einsprung
                  297 * ------------------------
                  298 *
0DC6: 88          299 COPY      DEY              ;Puffer-
0DC7: 8C D6 0D    300           STY  KONST       ;Pointer
0DCA: 20 2A 0E    301           JSR  COPY1
0DCD: 4C FE 0C    302           JMP  BEFEHL2     ;Exit
                  303 *
                  304 * Puffer ab $0200:
                  305 * 200:COPY/DIR1/FILE1,/DIR2/FILE2
                  306 * 200:0  1...........
                  307 * 200:0              2...........
                  308 * 200:0              3.....
                  309 * 200:0123 = KONST = Y = vor "/"
                  310 *
                  311 * Pufferoffset + Längebyte
                  312 *
0DD0: 03 0B       313 FILE1     HEX  030B        ;FILE1
0DD2: 0F 0B       314 FILE2     HEX  0F0B        ;FILE2
0DD4: 0F 05       315 DIR2      HEX  0F05        ;DIR2
0DD6: 03          316 KONST     HEX  03          ;COP"Y"
                  317 *
                  318 * Name in Versalien konvertieren
                  319 *
0DD7: C9 2E       320 CONVERT   CMP  #'.'        ;="."
```

```
0DD9: F0 20     321           BEQ  CONOKAY
0DDB: C9 2F     322           CMP  #'/'        ;<"/"
0DDD: F0 20     323           BEQ  STRFLAG
0DDF: 90 1C     324           BCC  CONERR      ;<"0"
0DE1: C9 3A     325           CMP  #':'        ;<":"
0DE3: 90 16     326           BCC  CONOKAY     ;bis "9"
0DE5: C9 41     327           CMP  #'A'        ;<"A"
0DE7: 90 14     328           BCC  CONERR
0DE9: C9 5B     329           CMP  #'Ä'        ;<"Ä"
0DEB: 90 0E     330           BCC  CONOKAY
                331  *
0DED: C9 61     332           CMP  #'a'        ;<"a"
0DEF: 90 0C     333           BCC  CONERR
0DF1: C9 7B     334           CMP  #'ä'        ;>"z"
0DF3: B0 08     335           BCS  CONERR
0DF5: 38        336           SEC
0DF6: E9 20     337           SBC  #$20        ;a->A
0DF8: 99 00 02  338           STA  PUFFER,Y
0DFB: 18        339  CONOKAY  CLC
0DFC: 60        340           RTS
0DFD: 38        341  CONERR   SEC
0DFE: 60        342           RTS
0DFF: EE 04 0E  343  STRFLAG  INC  STRICHFL
0E02: D0 F7     344           BNE  CONOKAY
0E04: 00        345  STRICHFL HEX  00
                346  *
                347  * Bit 7 wegblenden
                348  *
0E05: AC D6 0D  349  BITOFF   LDY  KONST
0E08: C8        350  BITOFF1  INY
0E09: B9 00 02  351           LDA  PUFFER,Y
0E0C: 29 7F     352           AND  #%01111111
0E0E: 99 00 02  353           STA  PUFFER,Y
0E11: C9 0D     354           CMP  #$0D        ;Return
0E13: D0 F3     355           BNE  BITOFF1
0E15: 60        356           RTS
                357  *
                358  * $02-$15: BASIC.SYSTEM-Fehler
                359  * $27-$5A: MLI-Fehler
                360  * $81-$84: Copy-Fehler
                361  *
0E16: 48        362  ERROR    PHA
0E17: 20 2D FF  363           JSR  PRERR
0E1A: A9 BA     364           LDA  #":"
0E1C: 20 ED FD  365           JSR  COUT
0E1F: 68        366           PLA
0E20: 20 DA FD  367           JSR  PRBYTE
0E23: 4C 8E FD  368           JMP  CROUT
                369  *
                370  * $80 = Syntax Error
                371  *
0E26: A9 80     372  ERR80    LDA  #$80
0E28: D0 EC     373           BNE  ERROR
                374  *
```

```
                        375  * Copy - Externer Einsprung
                        376  * -------------------------
                        377  *
                        378  * FILE1 prüfen
                        379  *
0E2A: 20 05 0E          380  COPY1    JSR  BITOFF
0E2D: AC D6 0D          381           LDY  KONST
0E30: 8C D0 0D          382           STY  FILE1       ;Offset
0E33: C8                383           INY
0E34: B9 00 02          384           LDA  PUFFER,Y
0E37: C9 2F             385           CMP  #'/'
0E39: D0 EB             386           BNE  ERR80
0E3B: A2 00             387           LDX  #0
0E3D: 8E 04 0E          388           STX  STRICHFL
0E40: F0 0C             389           BEQ  COPY3
0E42: B9 00 02          390  COPY2    LDA  PUFFER,Y
0E45: C9 2C             391           CMP  #','        ;Komma
0E47: F0 0B             392           BEQ  COPY4
0E49: 20 D7 0D          393           JSR  CONVERT
0E4C: B0 D8             394           BCS  ERR80
0E4E: E8                395  COPY3    INX
0E4F: C8                396           INY
0E50: D0 F0             397           BNE  COPY2
0E52: F0 D2             398           BEQ  ERR80
0E54: AD 04 0E          399  COPY4    LDA  STRICHFL
0E57: F0 CD             400           BEQ  ERR80
0E59: 8E D1 0D          401           STX  FILE1+1     ;Länge
0E5C: E0 40             402           CPX  #64
0E5E: B0 C6             403           BCS  ERR80
0E60: E0 04             404           CPX  #4          ;"/A/B"
0E62: 90 C2             405           BCC  ERR80
0E64: 8C D2 0D          406           STY  FILE2       ;Offset
0E67: 8C D4 0D          407           STY  DIR2        ;Offset
                        408  *
                        409  * FILE2 prüfen
                        410  *
0E6A: C8                411           INY
0E6B: B9 00 02          412           LDA  PUFFER,Y
0E6E: C9 2F             413           CMP  #'/'
0E70: D0 B4             414           BNE  ERR80
0E72: A2 00             415           LDX  #0
0E74: 8E 04 0E          416           STX  STRICHFL
0E77: F0 0C             417           BEQ  COPY6
0E79: B9 00 02          418  COPY5    LDA  PUFFER,Y
0E7C: C9 0D             419           CMP  #$0D        ;Return
0E7E: F0 0B             420           BEQ  COPY7
0E80: 20 D7 0D          421           JSR  CONVERT
0E83: B0 A1             422           BCS  ERR80
0E85: E8                423  COPY6    INX
0E86: C8                424           INY
0E87: D0 F0             425           BNE  COPY5
0E89: F0 9B             426           BEQ  ERR80
0E8B: AD 04 0E          427  COPY7    LDA  STRICHFL
0E8E: F0 96             428           BEQ  ERR80
```

```
0E90: E0 40     429           CPX  #64
0E92: B0 92     430           BCS  ERR80
0E94: E0 04     431           CPX  #4
0E96: 90 8E     432           BCC  ERR80
0E98: 8E D3 0D  433           STX  FILE2+1     ;Länge
                434  *
                435  * Länge von DIR2 ermitteln
                436  *
0E9B: AC D4 0D  437           LDY  DIR2
0E9E: C8        438           INY
0E9F: A2 00     439           LDX  #0
0EA1: C8        440  COPY8     INY
0EA2: E8        441           INX
0EA3: B9 00 02  442           LDA  PUFFER,Y
0EA6: C9 2F     443           CMP  #'/'
0EA8: D0 F7     444           BNE  COPY8
0EAA: 8E D5 0D  445           STX  DIR2+1      ;Länge
                446  *
                447  * 1. Zeichen nach "/" Versal?
                448  *
0EAD: AE D0 0D  449           LDX  FILE1
0EB0: E8        450  COPY9     INX
0EB1: BD 00 02  451           LDA  PUFFER,X
0EB4: C9 2F     452           CMP  #'/'
0EB6: F0 06     453           BEQ  COPY10
0EB8: C9 0D     454           CMP  #$0D        ;Return
0EBA: F0 22     455           BEQ  MLI01
0EBC: D0 F2     456           BNE  COPY9
0EBE: E8        457  COPY10    INX
0EBF: BD 00 02  458           LDA  PUFFER,X
0EC2: C9 0D     459           CMP  #$0D        ;Return
0EC4: F0 18     460           BEQ  MLI01
0EC6: C9 41     461           CMP  #'A'
0EC8: B0 E6     462           BCS  COPY9
0ECA: 4C 26 0E  463           JMP  ERR80
                464  *
                465  * Fileinfo von FILE1
                466  *
0ECD: 00        467  ACCESSA   HEX  00          ;0
0ECE: 00        468  FILETYPA  HEX  00
0ECF: 00 00     469  AUXTYPA   HEX  0000
0ED1: 00        470  STORAGEA  HEX  00
0ED2: 00 00     471  BLOCKSA   HEX  0000
0ED4: 00 00     472  MDATEA    HEX  0000
0ED6: 00 00     473  MTIMEA    HEX  0000      ;10
0ED8: 00 00     474  CDATEA    HEX  0000
0EDA: 00 00     475  CTIMEA    HEX  0000      ;14
                476  *
                477  * File-Reference-Number
                478  *
0EDC: 00        479  FILE1REF  HEX  00        ;FILE1
0EDD: 00        480  FILE2REF  HEX  00        ;FILE2
                481  *
```

```
                        482  * MLI-Aufrufe
                        483  * ------------
                        484  *
                        485  * Netto-Datenübertragungsrate
                        486  * (Lesen + Schreiben)
                        487  *
                        488  * 1. FILER:  ca. 2245 Bytes/s
                        489  * 2. PROFID: ca. 3055 Bytes/s
                        490  * 3. FID:    ca. 4585 Bytes/s
                        491  *
0EDE: A9 00     BF      492  MLI01    LDA  #0
0EE0: 8D 94 BF          493           STA  LEVEL
0EE3: AD 91 0C          494           LDA  BITMAP
0EE6: F0 1C             495           BEQ  MLI03
                        496  *
                        497  * System-Bit-Map-Konflikt?
                        498  *
0EE8: A0 59             499           LDY  #$59        ;$BF59
0EEA: B9 00 BF          500  MLI02    LDA  MLI,Y
0EED: D0 0C             501           BNE  ERR81
0EEF: C8                502           INY
0EF0: C0 6B             503           CPY  #$6B        ;$BF6A
0EF2: D0 F6             504           BNE  MLI02
0EF4: B9 00 BF          505           LDA  MLI,Y       ;$BF6B
0EF7: C9 3F             506           CMP  #%00111111
0EF9: F0 09             507           BEQ  MLI03
                        508  *
                        509  * $81 = System-Bit-Map-Konflikt
                        510  *
0EFB: A9 81             511  ERR81    LDA  #$81
0EFD: D0 02             512           BNE  ERRMLI1
                        513  *
                        514  * $82 = Keine Datei, sondern DIR
                        515  *
0EFF: A9 82             516  ERR82    LDA  #$82
                        517  *
0F01: 4C 16 0E          518  ERRMLI1  JMP  ERROR
                        519  *
                        520  * Get Fileinfo von FILE1
                        521  * Wenn Directory, dann Exit
                        522  *
0F04: AE D0 0D          523  MLI03    LDX  FILE1
0F07: 8E BB 10          524           STX  NAME0
0F0A: AD D1 0D          525           LDA  FILE1+1
0F0D: 9D 00 02          526           STA  PUFFER,X
0F10: 20 B3 10          527           JSR  GETINFO0
0F13: B0 EC             528           BCS  ERRMLI1
0F15: AD C1 10          529           LDA  STORAGE0
0F18: C9 0D             530           CMP  #$0D        ;D/E/F
0F1A: B0 E3             531           BCS  ERR82       ;DIR!
0F1C: A2 0E             532           LDX  #14
0F1E: BD BD 10          533  MLI04    LDA  ACCESS0,X
0F21: 9D CD 0E          534           STA  ACCESSA,X
0F24: CA                535           DEX
0F25: 10 F7             536           BPL  MLI04
```

```
                        537  *
                        538  * Open FILE1
                        539  *
0F27: AE D0 0D          540           LDX  FILE1
0F2A: 8E 06 11          541           STX  NAME4
0F2D: A9 12             542           LDA  #>PUFREAD
0F2F: 8D 09 11          543           STA  IOBUF4+1
0F32: 20 FE 10          544           JSR  OPEN4
0F35: B0 19             545           BCS  ERRMLI2
0F37: AD 0A 11          546           LDA  FILEREF4
0F3A: 8D DC 0E          547           STA  FILE1REF
                        548  *
                        549  * Newline desaktivieren
                        550  *
0F3D: 8D 13 11          551           STA  FILEREF5
0F40: 20 0B 11          552           JSR  NEWLINE5
0F43: B0 0B             553           BCS  ERRMLI2
                        554  *
                        555  * Get EOF von FILE1
                        556  *
0F45: AD DC 0E          557           LDA  FILE1REF
0F48: 8D 45 11          558           STA  FILEREF9
0F4B: 20 3D 11          559           JSR  GETEOF9
0F4E: 90 13             560           BCC  MLI05
                        561  *
                        562  * CLOSE auszuführen versuchen
                        563  * und dann stets Exit
                        564  *
0F50: 48                565  ERRMLI2  PHA
0F51: 20 34 11          566           JSR  CLOSE8
0F54: 68                567           PLA
0F55: 4C 16 0E          568           JMP  ERROR
                        569  *
                        570  * $83 = Leere Datei nicht kopieren
                        571  *
0F58: A9 83             572  ERR83    LDA  #$83
0F5A: D0 F4             573           BNE  ERRMLI2
                        574  *
                        575  * Anzahl der Schübe ausrechnen
                        576  *
0F5C: 00                577  SCHUB    HEX  00
0F5D: 00 00 00          578  LAENGE   HEX  000000      ;LLMMHH
0F60: 00 00 00          579  REST     HEX  000000      ;LLMMHH
                        580  *
                        581  * Leerdatei?
                        582  *
0F63: A2 00             583  MLI05    LDX  #0
0F65: A0 02             584           LDY  #2
0F67: 38                585           SEC
0F68: BD 46 11          586  MLI06    LDA  EOF9,X
0F6B: 9D 5D 0F          587           STA  LAENGE,X
0F6E: F0 01             588           BEQ  MLI07
0F70: 18                589           CLC
0F71: E8                590  MLI07    INX
```

```
0F72: 88          591           DEY
0F73: 10 F3       592           BPL  MLI06
0F75: B0 E1       593           BCS  ERR83      ;leer!
                  594  *
                  595  * PUFLEN kleiner als LAENGE?
                  596  *
0F77: A9 00       597           LDA  #0
0F79: 8D 5C 0F    598           STA  SCHUB
0F7C: A2 00       599  MLI08    LDX  #0
0F7E: A0 02       600           LDY  #2
0F80: 38          601           SEC
0F81: BD 5D 0F    602  MLI08A   LDA  LAENGE,X   ;LL
0F84: FD 8E 0C    603           SBC  PUFLEN,X
0F87: 9D 60 0F    604           STA  REST,X
0F8A: E8          605           INX
0F8B: 88          606           DEY
0F8C: 10 F3       607           BPL  MLI08A
0F8E: 90 14       608           BCC  MLI09      ;<PUFLEN
                  609  *
                  610  * Momentanen Rest übertragen
                  611  *
0F90: A0 02       612           LDY  #2
0F92: B9 60 0F    613  MLI08B   LDA  REST,Y
0F95: 99 5D 0F    614           STA  LAENGE,Y
0F98: 88          615           DEY
0F99: 10 F7       616           BPL  MLI08B
0F9B: EE 5C 0F    617           INC  SCHUB
0F9E: D0 DC       618           BNE  MLI08
                  619  *
                  620  * $84 = FILE1 > 255 * PUFLEN
                  621  *
0FA0: A9 84       622  ERR84    LDA  #$84
0FA2: D0 AC       623           BNE  ERRMLI2
                  624  *
0FA4: AD 5D 0F    625  MLI09    LDA  LAENGE
0FA7: 8D 60 0F    626           STA  REST
0FAA: AD 5E 0F    627           LDA  LAENGE+1
0FAD: 8D 61 0F    628           STA  REST+1
                  629  *
                  630  * 1. Schub lesen
                  631  *
0FB0: 20 67 10    632           JSR  RDHUNK1
0FB3: 90 03       633           BCC  MLI10
                  634  *
0FB5: 4C 50 0F    635  ERRMLI3  JMP  ERRMLI2
                  636  *
                  637  * DIR2 vorhanden?
                  638  *
0FB8: AE D4 0D    639  MLI10    LDX  DIR2
0FBB: 8E BB 10    640           STX  NAME0
0FBE: AD D5 0D    641           LDA  DIR2+1
0FC1: 9D 00 02    642           STA  PUFFER,X
0FC4: 20 B3 10    643           JSR  GETINF00
0FC7: B0 EC       644           BCS  ERRMLI3
```

```
                        645  *
                        646  * FILE2 bereits vorhanden?
                        647  * Wenn ja, erst löschen
                        648  * Wenn nein, anlegen
                        649  *
0FC9: AE D2 0D          650          LDX  FILE2
0FCC: 8E BB 10          651          STX  NAME0
0FCF: AD D3 0D          652          LDA  FILE2+1
0FD2: 9D 00 02          653          STA  PUFFER,X
0FD5: 20 B3 10          654          JSR  GETINF00
0FD8: B0 0B             655          BCS  MLI11       ;nein
                        656  *
                        657  * Destroy FILE2
                        658  * Aus Sicherheitsgründen jedoch
                        659  * keinen gelockten FILE2 löschen!
                        660  *
0FDA: AE D2 0D          661          LDX  FILE2
0FDD: 8E E9 10          662          STX  NAME2
0FE0: 20 E1 10          663          JSR  DESTROY2
0FE3: B0 D0             664          BCS  ERRMLI3
                        665  *
                        666  * Create FILE2
                        667  *
0FE5: AE D2 0D          668  MLI11   LDX  FILE2
0FE8: 8E F3 10          669          STX  NAME3
0FEB: AD CE 0E          670          LDA  FILETYPA
0FEE: 8D F6 10          671          STA  FILETYP3
0FF1: AD CF 0E          672          LDA  AUXTYPA
0FF4: 8D F7 10          673          STA  AUXTYP3
0FF7: AD D0 0E          674          LDA  AUXTYPA+1
0FFA: 8D F8 10          675          STA  AUXTYP3+1
0FFD: A2 03             676          LDX  #3
0FFF: BD D8 0E          677  MLI12   LDA  CDATEA,X
1002: 9D FA 10          678          STA  CDATE3,X
1005: CA                679          DEX
1006: 10 F7             680          BPL  MLI12
1008: 20 EB 10          681          JSR  CREATE3
100B: 90 11             682          BCC  MLI13
                        683  *
                        684  * Wenn jetzt noch ein Fehler
                        685  * auftritt, dann FILE1/FILE2
                        686  * zu schließen und dann FILE2
                        687  * zu löschen versuchen.
                        688  *
100D: 48                689  ERRMLI4 PHA
100E: 20 34 11          690          JSR  CLOSE8
1011: AE D2 0D          691          LDX  FILE2
1014: 8E E9 10          692          STX  NAME2
1017: 20 E1 10          693          JSR  DESTROY2
101A: 68                694          PLA
101B: 4C 16 0E          695          JMP  ERROR
                        696  *
                        697  * Open FILE2
                        698  *
```

```
101E: AE D2 0D  699  MLI13    LDX  FILE2
1021: 8E 06 11  700          STX  NAME4
1024: A9 16     701          LDA  #>PUFWRITE
1026: 8D 09 11  702          STA  IOBUF4+1
1029: 20 FE 10  703          JSR  OPEN4
102C: B0 DF     704          BCS  ERRMLI4
102E: AD 0A 11  705          LDA  FILEREF4
1031: 8D DD 0E  706          STA  FILE2REF
                707  *
                708  * Newline desaktivieren
                709  *
1034: 8D 13 11  710          STA  FILEREF5
1037: 20 0B 11  711          JSR  NEWLINE5
103A: B0 D1     712          BCS  ERRMLI4
103C: 90 05     713          BCC  MLI15      ;stets
                714  *
                715  * Nächsten Schub lesen/schreiben
                716  *
103E: 20 67 10  717  MLI14    JSR  RDHUNK1
1041: B0 CA     718          BCS  ERRMLI4
                719  *
                720  * 1. Schub schreiben
                721  *
1043: 20 8D 10  722  MLI15    JSR  WRHUNK1
1046: B0 C5     723          BCS  ERRMLI4
                724  *
1048: CE 5C 0F  725          DEC  SCHUB
104B: 10 F1     726          BPL  MLI14
                727  *
                728  * FILE1 und FILE2 schließen
                729  *
104D: 20 34 11  730          JSR  CLOSE8
                731  *
                732  * Set Fileinfo FILE2
                733  *
1050: A2 0A     734          LDX  #10
1052: BD CD 0E  735  MLI16    LDA  ACCESSA,X
1055: 9D D6 10  736          STA  ACCESS1,X
1058: CA        737          DEX
1059: 10 F7     738          BPL  MLI16
105B: AE D2 0D  739          LDX  FILE2
105E: 8E D4 10  740          STX  NAME1
1061: 20 CC 10  741          JSR  SETINFO1
1064: B0 A7     742          BCS  ERRMLI4
                743  *
1066: 60        744          RTS             ;Exit
                745  *
                746  * Schub lesen/schreiben
                747  * --------------------
                748  *
1067: AD DC 0E  749  RDHUNK1  LDA  FILE1REF
106A: 8D 1E 11  750          STA  FILEREF6
106D: AD 8E 0C  751          LDA  PUFLEN     ;LL
1070: 8D 21 11  752          STA  REQUEST6
```

```
1073: AD 8F 0C  753            LDA  PUFLEN+1    ;HH
1076: 8D 22 11  754            STA  REQUEST6+1
1079: AD 5C 0F  755            LDA  SCHUB
107C: D0 0C     756            BNE  RDHUNK2
107E: AD 60 0F  757            LDA  REST
1081: 8D 21 11  758            STA  REQUEST6
1084: AD 61 0F  759            LDA  REST+1
1087: 8D 22 11  760            STA  REQUEST6+1
108A: 4C 16 11  761  RDHUNK2   JMP  READ6       ;RTS
                762  *
108D: AD DD 0E  763  WRHUNK1   LDA  FILE2REF
1090: 8D 2D 11  764            STA  FILEREF7
1093: AD 8E 0C  765            LDA  PUFLEN      ;LL
1096: 8D 30 11  766            STA  REQUEST7
1099: AD 8F 0C  767            LDA  PUFLEN+1    ;HH
109C: 8D 31 11  768            STA  REQUEST7+1
109F: AD 5C 0F  769            LDA  SCHUB
10A2: D0 0C     770            BNE  WRHUNK2
10A4: AD 60 0F  771            LDA  REST
10A7: 8D 30 11  772            STA  REQUEST7
10AA: AD 61 0F  773            LDA  REST+1
10AD: 8D 31 11  774            STA  REQUEST7+1
10B0: 4C 25 11  775  WRHUNK2   JMP  WRITE7      ;RTS
                776  *
                777  * Standard-MLI-Aufrufe
                778  * --------------------
                779  *
                780  * "*" = zu pokende Parameter
                781  *
                782  * Get Fileinfo
                783  *
10B3: 20 00 BF  784  GETINFO0 JSR  MLI
10B6: C4        785            HEX  C4           ;Getinfo
10B7: BA 10     786            DA   COUNT0
10B9: 60        787            RTS
10BA: 0A        788  COUNT0    HEX  0A
10BB: 00 02     789  NAME0     DA   PUFFER       ;* nur LL
10BD: 00        790  ACCESS0   HEX  00
10BE: 00        791  FILETYP0  HEX  00
10BF: 00 00     792  AUXTYP0   HEX  0000
10C1: 00        793  STORAGE0  HEX  00
10C2: 00 00     794  BLOCKS0   HEX  0000
10C4: 00 00     795  MDATE0    HEX  0000
10C6: 00 00     796  MTIME0    HEX  0000
10C8: 00 00     797  CDATE0    HEX  0000
10CA: 00 00     798  CTIME0    HEX  0000
                799  *
                800  * Set Fileinfo
                801  *
10CC: 20 00 BF  802  SETINFO1 JSR  MLI
10CF: C3        803            HEX  C3           ;Setinfo
10D0: D3 10     804            DA   COUNT1
10D2: 60        805            RTS
10D3: 07        806  COUNT1    HEX  07
```

```
10D4: 00 02     807  NAME1    DA   PUFFER     ;* nur LL
10D6: 00        808  ACCESS1  HEX  00         ;*
10D7: 00        809  FILETYP1 HEX  00         ;*
10D8: 00 00     810  AUXTYP1  HEX  0000       ;*
10DA: 00        811  STORAGE1 HEX  00         ;Filler
10DB: 00 00     812  BLOCKS1  HEX  0000       ;Filler
10DD: 00 00     813  MDATE1   HEX  0000       ;*
10DF: 00 00     814  MTIME1   HEX  0000       ;*
                815  *
                816  * Destroy
                817  *
10E1: 20 00 BF  818  DESTROY2 JSR  MLI
10E4: C1        819           HEX  C1         ;Destroy
10E5: E8 10     820           DA   COUNT2
10E7: 60        821           RTS
10E8: 01        822  COUNT2   HEX  01
10E9: 00 02     823  NAME2    DA   PUFFER     ;* nur LL
                824  *
                825  * Create
                826  *
10EB: 20 00 BF  827  CREATE3  JSR  MLI
10EE: C0        828           HEX  C0         ;Create
10EF: F2 10     829           DA   COUNT3
10F1: 60        830           RTS
10F2: 07        831  COUNT3   HEX  07
10F3: 00 02     832  NAME3    DA   PUFFER     ;* nur LL
10F5: C3        833  ACCESS3  HEX  C3         ;unlocked
10F6: 00        834  FILETYP3 HEX  00         ;*
10F7: 00 00     835  AUXTYP3  HEX  0000       ;*
10F9: 01        836  STORAGE3 HEX  01         ;Sämling
10FA: 00 00     837  CDATE3   HEX  0000       ;*
10FC: 00 00     838  CTIME3   HEX  0000       ;*
                839  *
                840  * Open
                841  *
10FE: 20 00 BF  842  OPEN4    JSR  MLI
1101: C8        843           HEX  C8         ;Open
1102: 05 11     844           DA   COUNT4
1104: 60        845           RTS
1105: 03        846  COUNT4   HEX  03
1106: 00 02     847  NAME4    DA   PUFFER     ;* nur LL
1108: 00 00     848  IOBUF4   HEX  0000       ;* nur HH
110A: 00        849  FILEREF4 HEX  00
                850  *
                851  * Newline
                852  *
110B: 20 00 BF  853  NEWLINE5 JSR  MLI
110E: C9        854           HEX  C9         ;Newline
110F: 12 11     855           DA   COUNT5
1111: 60        856           RTS
1112: 03        857  COUNT5   HEX  03
1113: 00        858  FILEREF5 HEX  00         ;*
1114: 00        859  ENABLE5  HEX  00         ;disable
1115: 00        860  NEWCHAR  HEX  00         ;disable
```

```
                        861  *
                        862  * Read
                        863  *
1116: 20 00 BF          864  READ6    JSR  MLI
1119: CA                865           HEX  CA          ;Read
111A: 1D 11             866           DA   COUNT6
111C: 60                867           RTS
111D: 04                868  COUNT6   HEX  04
111E: 00                869  FILEREF6 HEX  00          ;*
111F: 00 1A             870  DATABUF6 DA   PUFBEG      ;$1A00
1121: 00 00             871  REQUEST6 HEX  0000        ;* LLHH
1123: 00 00             872  TRANS6   HEX  0000
                        873  *
                        874  * Write
                        875  *
1125: 20 00 BF          876  WRITE7   JSR  MLI
1128: CB                877           HEX  CB          ;Write
1129: 2C 11             878           DA   COUNT7
112B: 60                879           RTS
112C: 04                880  COUNT7   HEX  04
112D: 00                881  FILEREF7 HEX  00          ;*
112E: 00 1A             882  DATABUF7 DA   PUFBEG      ;$1A00
1130: 00 00             883  REQUEST7 HEX  0000        ;* LLHH
1132: 00 00             884  TRANS7   HEX  0000
                        885  *
                        886  * Close
                        887  *
1134: 20 00 BF          888  CLOSE8   JSR  MLI
1137: CC                889           HEX  CC          ;Close
1138: 3B 11             890           DA   COUNT8
113A: 60                891           RTS
113B: 01                892  COUNT8   HEX  01
113C: 00                893  FILEREF8 HEX  00          ;0=alle!
                        894  *
                        895  * Get EOF
                        896  *
113D: 20 00 BF          897  GETEOF9  JSR  MLI
1140: D1                898           HEX  D1          ;Get EOF
1141: 44 11             899           DA   COUNT9
1143: 60                900           RTS
1144: 02                901  COUNT9   HEX  02
1145: 00                902  FILEREF9 HEX  00          ;*
1146: 00 00 00          903  EOF9     HEX  000000
                        904  *
                        905  *
                        906  *
                        907  * Online
                        908  * ------
                        909  *
                        910  * 256-Byte-Puffer löschen
                        911  *
1149: A0 00             912  ONLINE   LDY  #0
114B: 84 CE             913           STY  IND
114D: A9 1A             914           LDA  #>PUFBEG
```

```
114F: 85 CF     915           STA  IND+1
1151: 98        916           TYA
1152: 91 CE     917 ONLINE1   STA  (IND),Y
1154: C8        918           INY
1155: D0 FB     919           BNE  ONLINE1
                920 *
                921 * On Line ausführen
                922 *
1157: 20 00 BF  923 ONLINE2   JSR  MLI
115A: C5        924           HEX  C5          ;Online
115B: 65 11     925           DA   ONCOUNT
115D: F0 0A     926           BEQ  ONLOOP0
115F: 20 16 0E  927           JSR  ERROR
1162: 4C FE 0C  928           JMP  BEFEHL2     ;Exit
                929 *
1165: 02        930 ONCOUNT   HEX  02
1166: 00        931 ONUNIT    HEX  00          ;0=alle
1167: 00 1A     932 ONBUFFER  DA   PUFBEG
                933 *
                934 * Endmarker erreicht?
                935 *
1169: A0 00     936 ONLOOP0   LDY  #0
116B: B1 CE     937           LDA  (IND),Y
116D: D0 0B     938           BNE  ONLOOP1
116F: C8        939           INY
1170: B1 CE     940           LDA  (IND),Y
1172: D0 06     941           BNE  ONLOOP1
1174: 20 8E FD  942           JSR  CROUT
1177: 4C FE 0C  943           JMP  BEFEHL2     ;Exit
                944 *
117A: 20 8E FD  945 ONLOOP1   JSR  CROUT
117D: A0 00     946           LDY  #0
                947 *
                948 * Slot anzeigen
                949 *
117F: A9 D3     950           LDA  #"S"
1181: 20 ED FD  951           JSR  COUT
1184: B1 CE     952           LDA  (IND),Y
1186: 29 70     953           AND  #%01110000
1188: 4A        954           LSR
1189: 4A        955           LSR
118A: 4A        956           LSR
118B: 4A        957           LSR
118C: 09 B0     958           ORA  #$B0        ;0
118E: 20 ED FD  959           JSR  COUT        ;Slot
1191: A9 AC     960           LDA  #","
1193: 20 ED FD  961           JSR  COUT
1196: B1 CE     962           LDA  (IND),Y
1198: 29 0F     963           AND  #%00001111
119A: 8D EA 11  964           STA  ONLEN
                965 *
                966 * Drive anzeigen
                967 *
119D: A9 C4     968           LDA  #"D"
```

```
119F: 20 ED FD  969           JSR  COUT
11A2: B1 CE     970           LDA  (IND),Y
11A4: 10 04     971           BPL  ONLOOP2
11A6: A9 B2     972           LDA  #"2"        ;D2
11A8: D0 02     973           BNE  ONLOOP3
11AA: A9 B1     974 ONLOOP2   LDA  #"1"        ;D1
11AC: 20 ED FD  975 ONLOOP3   JSR  COUT
11AF: A9 A0     976           LDA  #$A0        ;Space
11B1: 20 ED FD  977           JSR  COUT
                978 *
11B4: AD EA 11  979           LDA  ONLEN       ;Len = 0
11B7: F0 17     980           BEQ  ONLOOP6     ;Fehler!
11B9: A9 AF     981           LDA  #"/"
11BB: 20 ED FD  982           JSR  COUT
                983 *
                984 * Name anzeigen
                985 *
11BE: C8        986 ONLOOP4   INY
11BF: CC EA 11  987           CPY  ONLEN
11C2: 90 02     988           BCC  ONLOOP5
11C4: D0 0F     989           BNE  ONLOOP7
                990 *
11C6: B1 CE     991 ONLOOP5   LDA  (IND),Y
11C8: 09 80     992           ORA  #$80
11CA: 20 ED FD  993           JSR  COUT
11CD: 4C BE 11  994           JMP  ONLOOP4
                995 *
                996 * Name leer, d.h. Diskette
                997 * nicht eingelegt: I/O-Fehler
                998 *
11D0: A9 BF     999 ONLOOP6   LDA  #"?"
11D2: 20 ED FD 1000           JSR  COUT
               1001 *
               1002 * Nächster Name
               1003 *
11D5: A9 AF    1004 ONLOOP7   LDA  #"/"
11D7: 20 ED FD 1005           JSR  COUT
11DA: 18       1006           CLC
11DB: A5 CE    1007           LDA  IND
11DD: 69 10    1008           ADC  #16
11DF: 85 CE    1009           STA  IND
11E1: A5 CF    1010           LDA  IND+1
11E3: 69 00    1011           ADC  #0
11E5: 85 CF    1012           STA  IND+1
11E7: 4C 69 11 1013           JMP  ONLOOP0
               1014 *
11EA: 00       1015 ONLEN     HEX  00

1387 Bytes
```

3.4. Diskettenkopierprogramme

3.4.1. PRODOS.COPY.A und PRODOS.COPY.O

Dieses Programm, das mit RUN PRODOS.COPY.A gestartet wird und das seinerseits ein Maschinenprogramm namens PRODOS.COPY.O einlädt, dient zum Kopieren ganzer Disketten. In Zeile 130 kann man folgende Parameter einstellen:

OS = Original-Slot, z.B. 6
OD = Original-Drive, z.B. 1
DS = Duplikat-Slot, z.B. 6
DD = Duplikat-Drive, z.B. 2
S = Spurenanzahl, z.B. 40 (vgl. QUICKCOPY!)

PRODOS.COPY.A eignet sich sowohl für 1-Drive- wie für 2-Drive-Besitzer.

PRODOS.COPY.A

```
100  REM *** PRODOS.COPY.A ***
110  ON  PEEK (3000) = 76 AND  PEEK (3001) = 198 AND  PEEK
     (3002) = 11 GOTO 120: PRINT  CHR$ (4)"BLOAD
     PRODOS.COPY.O"
120  PRINT  CHR$ (21): HOME : INVERSE : HTAB 14: PRINT
     "PRODOS.COPYA": HTAB 14: PRINT "U. STIEHL 84": NORMAL
     : PRINT : PRINT
130  OS = 6:OD = 1:DS = 6:DD = 1:S = 35
140  PRINT "ORIGINAL-SLOT: "OS;: HTAB 22: PRINT
     "DUPLIKAT-SLOT: "DS: PRINT "ORIGINAL-DRIVE:"OD;: HTAB
     22: PRINT "DUPLIKAT-DRIVE:"DD
150  PRINT : HTAB 13: PRINT "SPURENANZAHL:"S
160  PRINT : HTAB 14: PRINT "START J/N ";
170  GET X$: ON X$ = "J" OR X$ = "j" GOTO 180: ON X$ <  >
     "N" AND X$ <  > "n" GOTO 170: HOME : END
180  IF OS = DS AND OD = DD THEN  VTAB 8:: HTAB 11: CALL
     - 958: PRINT "DUPLIKAT EINLEGEN": PRINT : HTAB 14:
     PRINT "W = WEITER ";: GOTO 200
190  VTAB 8:: HTAB 11: CALL  - 958: PRINT "DISKETTEN
     EINLEGEN": PRINT : HTAB 14: PRINT "W = WEITER ";
200  GET X$: IF X$ <  > "W" AND X$ <  > "w" GOTO 200
210  C = 3000: POKE C + 3,OS: POKE C + 4,OD: POKE C + 5,DS:
     POKE C + 6,DD: POKE C + 7,S
220  B = S * 8: POKE C + 9, INT (B / 256): POKE C + 8,B -
     PEEK (C + 9) * 256
230  CALL C
240  IF  PEEK (C + 10) <  > 0 THEN  HOME : PRINT "FEHLER
     "; CHR$ (7);: GET X$
250  RUN 120
```

PRODOS.COPY.0

```
BSAVE PRODOS.COPY.0, A3000, L1631

                        1             ORG  3000
                        2     *
                        3     * PRODOS.COPY.0
                        4     * ==============
                        5     *
                        6     * Für Apple IIc/IIe/II Plus.
                        7     * Ohne 280-Block-Patch für
                        8     * alle ProDOS-Versionen.
                        9     * Für wahlweise 1 oder 2 Drives.
                       10     *
                       11     * 12.04.85
                       12     *
                       13     HOME     EQU  $FC58
                       14     PRINT    EQU  $FDED
                       15     RDKEY    EQU  $FD0C
                       16     MLI      EQU  $BF00
                       17     SCREEN   EQU  $0400
                       18     *
                       19     PUFFBEG  EQU  $1400       ;5120
                       20     PUFFEND  EQU  $9A00       ;39424
                       21     *
0BB8: 4C C6 0B         22              JMP  START1
                       23     *
0BBB: 06               24     OSLOT    HEX  06
0BBC: 01               25     ODRIVE   HEX  01
0BBD: 06               26     DSLOT    HEX  06
0BBE: 02               27     DDRIVE   HEX  02
0BBF: 23               28     SPUREN   HEX  23         ;35
0BC0: 18 01            29     BLOCKS   HEX  1801       ;$0118
0BC2: 00               30     ERROR    HEX  00
0BC3: FE               31     VOLUME   HEX  FE
                       32     *
0BC4: 60               33     OUNIT    HEX  60         ;S6,D1
0BC5: E0               34     DUNIT    HEX  E0         ;S6,D2
                       35     *
                       36     *-------------------------------
                       37     *
0BC6: 20 58 FC         38     START1   JSR  HOME
0BC9: A2 00            39              LDX  #0
0BCB: BD D9 0B         40     START2   LDA  STRING,X
0BCE: F0 06            41              BEQ  START3
0BD0: 20 ED FD         42              JSR  PRINT
0BD3: E8               43              INX
0BD4: D0 F5            44              BNE  START2
0BD6: 4C 4A 10         45     START3   JMP  START4
                       46     *
0BD9: 85               47     STRING   HEX  85         ;CHR$(21)
0BDA: 10 12 0F         48              INV  "PRODOS.COPYA"
0BDD: 04 0F 13 2E 03 0F 10 19
0BE5: 01
```

```
0BE6: 8D          49              HEX  8D
0BE7: 15 2E 20    50              INV  "U. STIEHL 84"
0BEA: 13 14 09 05 08 0C 20 38
0BF2: 34
0BF3: 8D 8D       51              HEX  8D8D
0BF5: D3 D4 C1    52              ASC  "START J/N "
0BF8: D2 D4 A0 CA AF CE A0
0BFF: 00          53              HEX  00
                  54   *
                  55   *-----------------------------
                  56   *
                  57   * ProDOS-Format der Fa. Apple
                  58   * Hier nicht gelistet
                  699  *-----------------------------
                  700  *
                  701  * ProDOS-Copya
                  702  * ==========
                  703  *
103A: 00 00 00    704  ZAREA    HEX  0000000000000000
103D: 00 00 00 00 00
1042: 00 00 00    705           HEX  0000000000000000
1045: 00 00 00 00 00
                  706  *
104A: 2C 10 C0    707  START4   BIT  $C010
104D: A9 00       708           LDA  #0
104F: 8D C2 0B    709           STA  ERROR       ;okay
1052: 20 0C FD    710  START5   JSR  RDKEY
1055: C9 CA       711           CMP  #"J"
1057: F0 0F       712           BEQ  START6
1059: C9 EA       713           CMP  #"j"
105B: F0 0B       714           BEQ  START6
105D: C9 CE       715           CMP  #"N"
105F: F0 06       716           BEQ  EXIT1
1061: C9 EE       717           CMP  #"n"
1063: F0 02       718           BEQ  EXIT1
1065: D0 EB       719           BNE  START5
                  720  *
1067: 60          721  EXIT1    RTS              ;Exit
                  722  *
                  723  * Volume-Nr. und Spurenanzahl
                  724  * poken
                  725  *
1068: AD C3 0B    726  START6   LDA  VOLUME
106B: 8D 6F 0C    727           STA  VOLNR+1
106E: AD BF 0B    728           LDA  SPUREN
1071: 8D F4 0C    729           STA  SPURNR+1
1074: 20 58 FC    730           JSR  HOME
                  731  *
                  732  * Mehr als 280 Blocks zulassen
                  733  *
                  734  *        JSR  PATCH       ;Spuren
1077: EA EA EA    735           HEX  EAEAEA      ;Füller
                  736  *
```

```
                        * Slot/Drive in Unit-Number
                        * packen
                        *
107A: AD BB 0B          OSLOT   LDA  OSLOT
107D: AE BC 0B          ODRIVE  LDX  ODRIVE
1080: 20 5A 11          SLOTDRIV JSR SLOTDRIV
1083: 8D C4 0B          OUNIT   STA  OUNIT
1086: AD BD 0B          DSLOT   LDA  DSLOT
1089: AE BE 0B          DDRIVE  LDX  DDRIVE
108C: 20 5A 11          SLOTDRIV JSR SLOTDRIV
108F: 8D C5 0B          DUNIT   STA  DUNIT
                        *
                        *-------------------------------
                        *
                        * Duplikat initialisieren
                        * Entry mit A = Unit-Number
                        * Exit  mit A = Error-Number
                        *
                        * Mit Block $0000 anfangen
                        *
1092: 20 DE 11          DUPL    JSR  DUPL
1095: A9 00             LDA  #0
1097: 8D 91 11          STA  RDBLOCK
109A: 8D 92 11          STA  RDBLOCK+1
109D: 8D A3 11          STA  WRBLOCK
10A0: 8D A4 11          STA  WRBLOCK+1
                        *
                        * Block 0 von nicht-formatierter
                        * Diskette lesen, um Lesekopf
                        * zu justieren
                        *
10A3: A9 14             LDA  #>PUFFBEG
10A5: 8D 90 11          STA  RDPUFF+1
10A8: AD C5 0B          LDA  DUNIT
10AB: 8D 8E 11          STA  RDUNIT
10AE: 20 81 11          JSR  READER
                        *
10B1: 20 44 11          JSR  ZSAVER
10B4: AD C5 0B          LDA  DUNIT
10B7: 20 00 0C          JSR  FORMAT
10BA: 8D C2 0B          STA  ERROR
10BD: 20 4F 11          JSR  ZLOADER
10C0: AD C2 0B          LDA  ERROR
10C3: D0 0F             BNE  EXIT2      ;<> 0
                        *
                        *-------------------------------
                        *
                        * Kopieren
                        * ========
                        *
                        * MLI-Unit-Number poken
                        *
10C5: AD C4 0B          LDA  OUNIT
10C8: 8D 8E 11          STA  RDUNIT
10CB: AD C5 0B          LDA  DUNIT
```

```
10CE: 8D A0 11  792           STA  WRUNIT
                793  *
10D1: 4C D8 10  794           JMP  RDHUNK1
                795  *
                796  * Fehler-Nr. bereits in ERROR
                797  * gepokt
                798  *
10D4: 2C 10 C0  799  EXIT2    BIT  $C010
10D7: 60        800           RTS              ;Exit
                801  *
                802  *-------------------------------
                803  *
                804  * Schubweise Original in den Puffer
                805  * $1400-$9A00 einlesen und von
                806  * dort auf Duplikat schreiben
                807  *
                808  * Read a Hunk
                809  * -----------
                810  *
10D8: A9 14     811  RDHUNK1  LDA  #>PUFFBEG
10DA: 8D 90 11  812           STA  RDPUFF+1
10DD: 20 A5 11  813           JSR  ORIG
10E0: 20 81 11  814  RDHUNK2  JSR  READER
10E3: B0 EF     815           BCS  EXIT2
10E5: EE 90 11  816           INC  RDPUFF+1
10E8: EE 90 11  817           INC  RDPUFF+1
10EB: EE 91 11  818           INC  RDBLOCK
10EE: D0 03     819           BNE  RDHUNK3
10F0: EE 92 11  820           INC  RDBLOCK+1
10F3: 38        821  RDHUNK3  SEC
10F4: AD C0 0B  822           LDA  BLOCKS      ;LL
10F7: E9 01     823           SBC  #1          ;280-1
10F9: ED 91 11  824           SBC  RDBLOCK     ;LL
10FC: AD C1 0B  825           LDA  BLOCKS+1    ;HH
10FF: ED 92 11  826           SBC  RDBLOCK+1   ;HH
1102: 90 07     827           BCC  WRHUNK1     ;Overflow
1104: AD 90 11  828  RDHUNK4  LDA  RDPUFF+1
1107: C9 9A     829           CMP  #>PUFFEND
1109: 90 D5     830           BCC  RDHUNK2
                831  *
                832  * Write a Hunk
                833  * ------------
                834  *
110B: A9 14     835  WRHUNK1  LDA  #>PUFFBEG
110D: 8D A2 11  836           STA  WRPUFF+1
1110: 20 DE 11  837           JSR  DUPL
1113: 20 93 11  838  WRHUNK2  JSR  WRITER
1116: B0 BC     839           BCS  EXIT2
1118: EE A2 11  840           INC  WRPUFF+1
111B: EE A2 11  841           INC  WRPUFF+1
111E: EE A3 11  842           INC  WRBLOCK
1121: D0 03     843           BNE  WRHUNK3
1123: EE A4 11  844           INC  WRBLOCK+1
1126: 38        845  WRHUNK3  SEC
1127: AD C0 0B  846           LDA  BLOCKS      ;LL
```

```
112A: E9 01     847             SBC  #1          ;280-1
112C: ED A3 11  848             SBC  WRBLOCK     ;LL
112F: AD C1 0B  849             LDA  BLOCKS+1    ;HH
1132: ED A4 11  850             SBC  WRBLOCK+1   ;HH
1135: 90 0A     851             BCC  EXIT3       ;Overflow
1137: AD A2 11  852  WRHUNK4    LDA  WRPUFF+1
113A: C9 9A     853             CMP  #>PUFFEND
113C: 90 D5     854             BCC  WRHUNK2
113E: 4C D8 10  855             JMP  RDHUNK1     ;erneut
                856  *
1141: 4C C6 0B  857  EXIT3      JMP  START1      ;Exit
                858  *
                859  *-------------------------------
                860  *
1144: A2 0F     861  ZSAVER     LDX  #$0F
1146: B5 D0     862  ZSAVE      LDA  ZD0,X
1148: 9D 3A 10  863             STA  ZAREA,X
114B: CA        864             DEX
114C: 10 F8     865             BPL  ZSAVE
114E: 60        866             RTS
                867  *
114F: A2 0F     868  ZLOADER    LDX  #$0F
1151: BD 3A 10  869  ZLOAD      LDA  ZAREA,X
1154: 95 D0     870             STA  ZD0,X
1156: CA        871             DEX
1157: 10 F8     872             BPL  ZLOAD
1159: 60        873             RTS
                874  *
115A: 0A        875  SLOTDRIV   ASL
115B: 0A        876             ASL
115C: 0A        877             ASL
115D: 0A        878             ASL
115E: E0 01     879             CPX  #1          ;Drive 1
1160: F0 02     880             BEQ  SLOTDR
1162: 09 80     881             ORA  #%10000000
1164: 60        882  SLOTDR     RTS
                883  *
                884  * Spurprüfroutine patchen (1.0.1)
                885  * Hier desaktiviert, damit Programm
                886  * mit allen ProDOS-Versionen läuft.
                887  *
1165: AD 83 C0  888  PATCH      LDA  $C083
1168: AD 83 C0  889             LDA  $C083
116B: A9 18     890             LDA  #$18
116D: 8D 09 F8  891             STA  $F809
1170: A9 90     892             LDA  #$90
1172: 8D 0A F8  893             STA  $F80A
1175: A9 04     894             LDA  #$04
1177: 8D 0B F8  895             STA  $F80B
117A: AD 81 C0  896             LDA  $C081
117D: AD 81 C0  897             LDA  $C081
1180: 60        898             RTS
                899  *
                900  * Read-Block
```

```
                        901  *
1181: 20 00 BF          902  READER   JSR  MLI
1184: 80                903  RDCOM    HEX  80          ;Read
1185: 8D 11             904           DA   RDCOUNT
1187: 90 03             905           BCC  RDOKAY
1189: 8D C2 0B          906           STA  ERROR
118C: 60                907  RDOKAY   RTS
                        908  *
118D: 03                909  RDCOUNT  HEX  03
118E: 60                910  RDUNIT   HEX  60
118F: 00 16             911  RDPUFF   HEX  0016        ;LLHH
1191: 00 00             912  RDBLOCK  HEX  0000        ;LLHH
                        913  *
                        914  * Write-Block
                        915  *
1193: 20 00 BF          916  WRITER   JSR  MLI
1196: 81                917  WRCOM    HEX  81          ;Write
1197: 9F 11             918           DA   WRCOUNT
1199: 90 03             919           BCC  WROKAY
119B: 8D C2 0B          920           STA  ERROR
119E: 60                921  WROKAY   RTS
                        922  *
119F: 03                923  WRCOUNT  HEX  03
11A0: E0                924  WRUNIT   HEX  E0
11A1: 00 16             925  WRPUFF   HEX  0016        ;LLHH
11A3: 00 00             926  WRBLOCK  HEX  0000        ;LLHH
                        927  *
                        928  * Original bzw. Duplikat einlegen
                        929  *
11A5: A2 00             930  ORIG     LDX  #0
11A7: AD C4 0B          931           LDA  OUNIT
11AA: CD C5 0B          932           CMP  DUNIT
11AD: D0 17             933           BNE  ORIG4
11AF: 2C 10 C0          934           BIT  $C010
11B2: BD C7 11          935  ORIG1    LDA  ORIG3,X
11B5: F0 06             936           BEQ  ORIG2
11B7: 9D 00 04          937           STA  SCREEN,X
11BA: E8                938           INX
11BB: D0 F5             939           BNE  ORIG1
11BD: AD 00 C0          940  ORIG2    LDA  $C000
11C0: 10 FB             941           BPL  ORIG2
11C2: C9 CF             942           CMP  #"O"
11C4: D0 DF             943           BNE  ORIG
11C6: 60                944  ORIG4    RTS
11C7: 0F                945  ORIG3    INV  "O"
11C8: A0 BD A0          946           ASC  " = "
11CB: 0F 12 09          947           INV  "ORIGINAL"
11CE: 07 09 0E 01 0C
11D3: A0 C5 C9          948           ASC  " EINLEGEN!"
11D6: CE CC C5 C7 C5 CE A1
11DD: 00                949           HEX  00
                        950  *
11DE: A2 00             951  DUPL     LDX  #0
11E0: AD C4 0B          952           LDA  OUNIT
```

```
11E3: CD C5 0B   953            CMP  DUNIT
11E6: D0 17      954            BNE  DUPL4
11E8: 2C 10 C0   955            BIT  $C010
11EB: BD 00 12   956  DUPL1     LDA  DUPL3,X
11EE: F0 06      957            BEQ  DUPL2
11F0: 9D 00 04   958            STA  SCREEN,X
11F3: E8         959            INX
11F4: D0 F5      960            BNE  DUPL1
11F6: AD 00 C0   961  DUPL2     LDA  $C000
11F9: 10 FB      962            BPL  DUPL2
11FB: C9 C4      963            CMP  #"D"
11FD: D0 DF      964            BNE  DUPL
11FF: 60         965  DUPL4     RTS
1200: 04         966  DUPL3     INV  "D"
1201: A0 BD A0   967            ASC  " = "
1204: 04 15 10   968            INV  "DUPLIKAT"
1207: 0C 09 0B 01 14
120C: A0 C5 C9   969            ASC  " EINLEGEN!"
120F: CE CC C5 C7 C5 CE A1
1216: 00         970            HEX  00

1631 Bytes
```

3.4.2. QUICKCOPYPUFFER und QUICKCOPY

Dieses Programm, das mit BRUN QUICKCOPY gestartet wird, ist eine geringfügig modifizierte Version des bereits im „Peeker", Heft 1/1985, erschienenen Programms, das jetzt unter *allen* ProDOS-Versionen läuft. Es setzt zwingend 2 Laufwerke sowie die 64K-Karte des Apple IIe oder einen IIc voraus. Wenn man über diese Gerätekonfiguration nicht verfügt, muß man ersatzweise PRODOS.COPY.A verwenden. QUICKCOPY-Parameter können wie folgt geändert werden:

BLOAD QUICKCOPY
CALL -151
08B1: 06 (Original-Slot)
08B2: 01 (Original-Drive)
08B3: 06 (Duplikat-Slot)
08B4: 02 (Duplikat-Drive)
08B5: FE (Volume-Nummer)
08B6: 01 (FORMAT ja; 00 = FORMAT nein)
08B7: 23 (35 Spuren, s. u.!)
08B8: 02 (Läufe)
08B9: 9A (Pufferende)
Ctrl-C
BSAVE QUICKCOPY

Für die Änderung von Spuren, Läufen und Pufferende bediene man sich des Applesoft-Programms QUICKCOPYPUFFER, das für eine gewünschte Spurenanzahl automatisch die erforderlichen Läufe (= Kopierdurchgänge) und das bestmögliche Pufferende berechnet. Man beachte jedoch, daß die Neuversion von QUICKCOPY ProDOS nicht mehr dergestalt patcht, daß mehr als 35 Spuren vom Disk-Driver gestattet werden. Dieser Patch muß für die benutzte ProDOS-Version selbst vorgenommen werden, da die Patch-Stellen bei jeder Version woanders liegen (vgl. Band 1, S. 65).

QUICKCOPYPUFFER

```
100  REM QUICKCOPYPUFFER
110  INPUT "Spuren:";S
120  BG = 140: REM Blockanzahl
130  PE = 154: REM HH $9A00
140  B = S * 8
150  L = B / BG: REM L=Läufe
160  IF L =  INT (L) THEN 180
170  L =  INT (L) + 1
180  T = BG -  INT (B / L)
190  PE = 154 - T
200  HOME : PRINT "Spuren:"S
210  PRINT "Läufe:"L
220  PRINT "Pufferende:"PE
230  G = (PE * 256 - 5120 + PE * 256 - 2048) * L
240  PRINT "Gesamt:"G: PRINT "Kilobyte:"G / 1024
250  BL = G / 512: IF B > BL THEN  GOTO 170
260  PRINT "Blocks:"BL
270  PRINT "Puffergröße:"G / L
```

QUICKCOPY

```
BSAVE QUICKCOPY, A2222, L1881

                1              ORG  2222
                2    *
                3    * QUICKCOPY für ProDOS
                4    * ====================
                5    *
                6    * Gerätevoraussetzung: Apple IIc
                7    * oder IIe mit 64K-Karte und
                8    * 2 DISK-II-Laufwerke
                9    * Läuft ohne den 280-Block-Patch
               10    * auf allen ProDOS-Versionen
               11    *
               12    * von U.Stiehl/06.10.84
               13    *
               14    HOME     EQU  $FC58
               15    PRINT    EQU  $FDED
               16    PRERR    EQU  $FF2D
```

```
                        17  HEXOUT   EQU  $FDDA
                        18  RDKEY    EQU  $FD0C
                        19  MLI      EQU  $BF00
                        20  *
                        21  * Main $1400-$9A00 = 33.5K
                        22  * Aux. $0800-$9A00 = 36.5K
                        23  * ------------------------
                        24  * Zusammen          = 70.0K
                        25  *
                        26  PUFFBEGM EQU  $1400      ;Main
                        27  PUFFBEGA EQU  $0800      ;Aux.
                        28  *
08AE: 4C BD 08          29          JMP  START1
                        30  *
                        31  *-----------------------------
                        32  *
                        33  * In diesem Bereich können die
                        34  * gewünschten Parameter gepokt
                        35  * werden.
                        36  *
                        37  * Original-Slot und -Drive
                        38  *
08B1: 06                39  OSLOT    HEX  06
08B2: 01                40  ODRIVE   HEX  01
                        41  *
                        42  * Duplikat-Slot und -Drive
                        43  *
08B3: 06                44  DSLOT    HEX  06
08B4: 02                45  DDRIVE   HEX  02
                        46  *
                        47  * Volume-Nummer 1-254
                        48  *
08B5: FE                49  VOLUME   HEX  FE
                        50  *
                        51  * Initflag: 0=nein, 1=ja
                        52  *
08B6: 01                53  INITFLAG HEX  01         ;INIT=ja
                        54  *
                        55  *-----------------------------
                        56  *
                        57  * Anzahl der Spuren, Anzahl der
                        58  * Durchläufe und Ende des Puffers
                        59  * müssen aufeinander abgestimmt
                        60  * sein. Es gilt im einzelnen:
                        61  *
                        62  * $23 Spuren, $02 Läufe, $9A Ende
                        63  * $24 Spuren, $03 Läufe, $6E Ende
                        64  * $28 Spuren, $04 Läufe, $5E Ende
                        65  * $50 Spuren, $05 Läufe, $8E Ende
                        66  * $A0 Spuren, $0A Läufe, $8E Ende
                        67  *
                        68  * Jeder Lauf ist gleich groß,
                        69  * es gibt also keinen kleineren
                        70  * Restlauf. 100% optimal sind
```

```
                        71   * nur 35 = $23 Spuren, für die
                        72   * das Programm speziell
                        73   * geschrieben worden ist.
                        74   *
                        75   * Spuren
                        76   *
08B7: 23                77   SPUREN1  HEX  23          ;35
                        78   *
                        79   * Läufe
                        80   *
08B8: 02                81   LAEUFE1  HEX  02          ;2 mal
                        82   *
                        83   * Ende
                        84   *
08B9: 9A                85   PUFFEND1 HEX  9A          ;$9A00
                        86   *
                        87   *-----------------------------
                        88   *
                        89   * Nachträglich ERROR-Nummer hier
                        90   *
08BA: 00                91   ERROR    HEX  00
                        92   *
                        93   *-----------------------------
                        94   *
                        95   * Original- und Duplikat-Unit
                        96   *
08BB: 60                97   OUNIT    HEX  60          ;S6,D1
08BC: E0                98   DUNIT    HEX  E0          ;S6,D2
                        99   *
                        100  *-----------------------------
                        101  *
08BD: 20 58 FC          102  START1   JSR  HOME
08C0: A2 00             103          LDX  #0
08C2: BD D0 08          104  START2   LDA  STRING,X
08C5: F0 06             105          BEQ  START3
08C7: 20 ED FD          106          JSR  PRINT
08CA: E8                107          INX
08CB: D0 F5             108          BNE  START2
08CD: 4C 3A 0E          109  START3   JMP  START4
                        110  *
08D0: 2A 11 15          111  STRING   INV  "*QUICK-COPY*"
08D3: 09 03 0B 2D 03 0F 10 19
08DB: 2A
08DC: 8D                112          HEX  8D
08DD: 15 2E 20          113          INV  "U. STIEHL 84"
08E0: 13 14 09 05 08 0C 20 38
08E8: 34
08E9: 8D 8D             114          HEX  8D8D
08EB: D3 D4 C1          115          ASC  "START J/N "
08EE: D2 D4 A0 CA AF CE A0
08F5: 00                116          HEX  00
08F6: 00 00 00          117          HEX  0000000000
08F9: 00 00
08FB: 00 00 00          118          HEX  0000000000
08FE: 00 00
```

```
                    119  *
                    120  *-----------------------------
                    121  *
                    122  * ProDOS-Format
                    123  * ============
                    124  *
                    125  * Dem FILER der Firma Apple
                    126  * mit Modifikationen entnommen.
                    127  * Hier nicht gelistet.
                    767  *=============================================
                    768  *
                    769  ZAREA    DS   256
                    770  *
                    771  * ProDOS-Quickcopy
                    772  * ================
                    773  *
0E3A: 2C 10 C0      774  START4   BIT  $C010       ;Strobe
0E3D: A9 00         775           LDA  #0
0E3F: 8D BA 08      776           STA  ERROR      ;okay
                    777  *
                    778  * Menü
                    779  *
0E42: 20 0C FD      780  START5   JSR  RDKEY
0E45: C9 CA         781           CMP  #"J"
0E47: F0 0F         782           BEQ  COPY1
0E49: C9 EA         783           CMP  #"j"
0E4B: F0 0B         784           BEQ  COPY1
0E4D: C9 CE         785           CMP  #"N"
0E4F: F0 06         786           BEQ  EXIT1
0E51: C9 EE         787           CMP  #"n"
0E53: F0 02         788           BEQ  EXIT
0E55: D0 EB         789           BNE  START5
                    790  *
                    791  *
                    792  *
0E57: 60            793  EXIT1    RTS          ; EXIT
                    794  *
                    795  *-----------------------------
                    796  *
                    797  * Zero-Page retten und
                    798  * MLI-Routine in Zero-Page
                    799  * kopieren
                    800  *
0E58: 20 03 0F      801  COPY1    JSR  ZSAVER
                    802  *
                    803  * Durchläufe und Pufferende poken
                    804  *
0E5B: AD B8 08      805           LDA  LAEUFE1
0E5E: 8D 0A 00      806           STA: LAEUFE2
0E61: AD B9 08      807           LDA  PUFFEND1
0E64: 8D 0B 00      808           STA: PUFFEND2
                    809  *
                    810  * Global Page in 64K-Karte
                    811  * duplizieren
```

```
                        812  *
0E67: 8D 05 C0          813           STA  $C005       ;WRAUX
0E6A: A2 00             814           LDX  #0
0E6C: BD 00 BF          815  MLICOPY  LDA  MLI,X
0E6F: 9D 00 BF          816           STA  MLI,X
0E72: E8                817           INX
0E73: D0 F7             818           BNE  MLICOPY
0E75: 8D 04 C0          819           STA  $C004       ;WRMAIN
                        820  *
                        821  * Mehr als 280 Blocks zulassen
                        822  * Nicht aktiv, da nur mit
                        823  * ProDOS Version 1.0.1 kompatibel
                        824  *
                        825  *        JSR  PATCH      ;Spuren
0E78: EA EA EA          826           HEX  EAEAEA      ;Füller
                        827  *
                        828  * Volume-Nummer und Spur-Anzahl
                        829  * für Format poken
                        830  *
0E7B: AD B5 08          831           LDA  VOLUME
0E7E: 8D 6F 09          832           STA  VOLNR+1
0E81: AD B7 08          833           LDA  SPUREN1
0E84: 8D F4 09          834           STA  SPURNR+1
                        835  *
                        836  * Slot/Drive in Unit-Number
                        837  * packen
                        838  *
0E87: AD B1 08          839           LDA  OSLOT
0E8A: AE B2 08          840           LDX  ODRIVE
0E8D: 20 21 0F          841           JSR  SLOTDRIV
0E90: 8D BB 08          842           STA  OUNIT
0E93: AD B3 08          843           LDA  DSLOT
0E96: AE B4 08          844           LDX  DDRIVE
0E99: 20 21 0F          845           JSR  SLOTDRIV
0E9C: 8D BC 08          846           STA  DUNIT
                        847  *
                        848  *-------------------------------
                        849  *
                        850  * INIT-Routine aus PRODOS-FILER
                        851  * der Firma Apple aufrufen:
                        852  *
                        853  * Entry mit A = Unit-Number
                        854  * Exit  mit A = Fehler-Number
                        855  *
                        856  * Mit Block $0000 anfangen
                        857  *
0E9F: A9 00             858           LDA  #0
0EA1: 8D A0 00          859           STA: RDBLOCK
0EA4: 8D A1 00          860           STA: RDBLOCK+1
0EA7: 8D AD 00          861           STA: WRBLOCK
0EAA: 8D AE 00          862           STA: WRBLOCK+1
                        863  *
                        864  * Block 0 von nicht-formatierter
                        865  * Diskette lesen, um Lesekopf
                        866  * zu justieren
```

```
                        867  *
0EAD: A9 14             868          LDA  #>PUFFBEGM
0EAF: 8D 9F 00          869          STA: RDPUFF+1
0EB2: AD BC 08          870          LDA  DUNIT
0EB5: 8D 9D 00          871          STA: RDUNIT
0EB8: 20 95 00          872          JSR  READER
                        873  *
                        874  * Falls Lesefehler auf Duplikat,
                        875  * dann stets INIT, andernfalls
                        876  * INIT ja,   wenn INITFLAG = 1
                        877  * INIT nein, wenn INITFLAG = 0
                        878  *
0EBB: B0 05             879          BCS  INIT       ;Fehler
0EBD: AD B6 08          880          LDA  INITFLAG
0EC0: F0 2A             881          BEQ  COPY2
                        882  *
                        883  * Duplikat formatieren
                        884  *
0EC2: AD BC 08          885  INIT    LDA  DUNIT
0EC5: 20 00 09          886          JSR  FORMAT
0EC8: 8D BA 08          887          STA  ERROR
0ECB: F0 1F             888          BEQ  COPY2      ;okay
                        889  *
                        890  * Fehler-Exit
                        891  *
0ECD: 20 15 0F          892  EXIT2   JSR  ZLOADER
0ED0: 20 58 FC          893          JSR  HOME
0ED3: 20 2D FF          894          JSR  PRERR
0ED6: A9 BA             895          LDA  #":"
0ED8: 20 ED FD          896          JSR  PRINT
0EDB: AD BA 08          897          LDA  ERROR
0EDE: 20 DA FD          898          JSR  HEXOUT
0EE1: A9 A0             899          LDA  #$A0
0EE3: 20 ED FD          900          JSR  PRINT
0EE6: 20 0C FD          901          JSR  RDKEY
0EE9: 4C BD 08          902          JMP  START1     ;Exit
                        903  *
                        904  *-------------------------------
                        905  *
                        906  * Kopieren
                        907  * ========
                        908  *
                        909  * MLI-Unit-Number poken
                        910  *
0EEC: AD BB 08          911  COPY2   LDA  OUNIT
0EEF: 8D 9D 00          912          STA: RDUNIT
0EF2: AD BC 08          913          LDA  DUNIT
0EF5: 8D AA 00          914          STA: WRUNIT
                        915  *
0EF8: 20 00 00          916          JSR  COPY3
0EFB: B0 D0             917          BCS  EXIT2
                        918  *
                        919  * Okay-Exit nach Kopie
                        920  *
0EFD: 20 15 0F          921          JSR  ZLOADER
0F00: 4C BD 08          922          JMP  START1     ;Exit
```

```
                        923  *
                        924  *═══════════════════════════════
                        925  *
                        926  * Zero-Page saven und loaden
                        927  *
0F03: A2 00             928  ZSAVER   LDX  #0
0F05: BD 00 00          929  ZSAVE    LDA: $0000,X
0F08: 9D 3A 0D          930           STA  ZAREA,X
0F0B: BD 58 0F          931           LDA  MARKER+1,X
0F0E: 9D 00 00          932           STA: $0000,X
0F11: E8                933           INX
0F12: D0 F1             934           BNE  ZSAVE
0F14: 60                935           RTS
                        936  *
0F15: A2 00             937  ZLOADER  LDX  #0
0F17: BD 3A 0D          938  ZLOAD    LDA  ZAREA,X
0F1A: 9D 00 00          939           STA: $0000,X
0F1D: E8                940           INX
0F1E: D0 F7             941           BNE  ZLOAD
0F20: 60                942           RTS
                        943  *
                        944  * Slot + Drive in Unit-Number
                        945  *
0F21: 0A                946  SLOTDRIV ASL
0F22: 0A                947           ASL
0F23: 0A                948           ASL
0F24: 0A                949           ASL
0F25: E0 01             950           CPX  #1          ;Drive 1
0F27: F0 02             951           BEQ  SLOTDR
0F29: 09 80             952           ORA  #%10000000
0F2B: 60                953  SLOTDR   RTS
                        954  *
                        955  * Spurprüfroutine patchen
                        956  *
0F2C: AD 83 C0          957  PATCH    LDA  $C083
0F2F: AD 83 C0          958           LDA  $C083
                        959  *
0F32: A9 18             960           LDA  #$18
0F34: 8D 09 F8          961           STA  $F809
0F37: A9 90             962           LDA  #$90
0F39: 8D 0A F8          963           STA  $F80A
0F3C: A9 04             964           LDA  #$04
0F3E: 8D 0B F8          965           STA  $F80B
                        966  *
                        967  * RAM-Disk abstellen
                        968  *
0F41: A9 4C             969           LDA  #$4C
0F43: 8D 18 FF          970           STA  $FF18
0F46: A9 3B             971           LDA  #$3B
0F48: 8D 19 FF          972           STA  $FF19
0F4B: A9 FF             973           LDA  #$FF
0F4D: 8D 1A FF          974           STA  $FF1A
                        975  *
0F50: AD 81 C0          976           LDA  $C081
0F53: AD 81 C0          977           LDA  $C081
0F56: 60                978           RTS
```

```
                        979  *
0F57: EA                980  MARKER   NOP
                        981  *
                        982  *=============================
                        983  *
                        984  * In 2 Schüben je 70K kopieren
                        985  * $0800-$9A00 AUX  36.5K
                        986  * $1400-$9A00 MAIN 33.5K
                        987  *
                        988  * Kopierroutine in Zero-Page!
                        989  * -------------------------
                        990  *
                        991           ORG  $0000
                        992  *
                        993  * 2 mal 70K lesen und schreiben
                        994  *
0000: 20 0C 00          995  COPY3     JSR  COPY4
0003: B0 4B             996           BCS  MAINRAM     ;Fehler
0005: C6 0A             997           DEC  LAEUFE2
0007: D0 F7             998           BNE  COPY3
0009: 60                999           RTS
                        1000 *
000A: 02                1001 LAEUFE2  HEX  02          ;2 mal
000B: 9A                1002 PUFFEND2 HEX  9A          ;$9A00
                        1003 *
                        1004 * 70K lesen
                        1005 *
000C: 20 5C 00          1006 COPY4    JSR  AUXRAM
000F: A9 08             1007          LDA  #>PUFFBEGA
0011: 20 63 00          1008          JSR  RDHUNK1
0014: B0 3A             1009          BCS  MAINRAM     ;Fehler
                        1010 *
0016: 20 50 00          1011          JSR  MAINRAM
0019: A9 14             1012          LDA  #>PUFFBEGM
001B: 20 63 00          1013          JSR  RDHUNK1
001E: B0 30             1014          BCS  MAINRAM     ;Fehler
                        1015 *
                        1016 * 70K schreiben
                        1017 *
0020: 20 5C 00          1018          JSR  AUXRAM
0023: A9 08             1019          LDA  #>PUFFBEGA
0025: 20 7C 00          1020          JSR  WRHUNK1
0028: B0 26             1021          BCS  MAINRAM     ;Fehler
                        1022 *
002A: 20 50 00          1023          JSR  MAINRAM
002D: A9 14             1024          LDA  #>PUFFBEGM
002F: 20 7C 00          1025          JSR  WRHUNK1
0032: B0 1C             1026          BCS  MAINRAM     ;Fehler
0034: 60                1027          RTS
                        1028 *
                        1029 * Füller für MLI-Zero-Page
                        1030 * MLI benutzt $003A-$004F
                        1031 *
0035: 00 00 00          1032          HEX  000000
```

```
0038: 00 00 00  1033          HEX  0000000000000000
003B: 00 00 00 00 00
0040: 00 00 00  1034          HEX  0000000000000000
0043: 00 00 00 00 00
0048: 00 00 00  1035          HEX  0000000000000000
004B: 00 00 00 00 00
                1036 *
                1037 * Zwischen unteren und oberen
                1038 * 47.5K $0200-$BFFF umschalten
                1039 *
0050: 8D 02 C0  1040 MAINRAM  STA  $C002      ;RDMAIN
0053: 8D 04 C0  1041          STA  $C004      ;WRMAIN
0056: 90 03     1042          BCC  MAINRAM1
0058: 8D BA 08  1043          STA  ERROR
005B: 60        1044 MAINRAM1 RTS
                1045 *
005C: 8D 03 C0  1046 AUXRAM   STA  $C003      ;RDAUX
005F: 8D 05 C0  1047          STA  $C005      ;WRAUX
0062: 60        1048          RTS
                1049 *
                1050 * Read
                1051 * -----
                1052 *
0063: 85 9F     1053 RDHUNK1  STA  RDPUFF+1
0065: 20 95 00  1054 RDHUNK2  JSR  READER
0068: B0 11     1055          BCS  RDHUNK4
006A: E6 9F     1056          INC  RDPUFF+1
006C: E6 9F     1057          INC  RDPUFF+1
006E: E6 A0     1058          INC  RDBLOCK
0070: D0 02     1059          BNE  RDHUNK3
0072: E6 A1     1060          INC  RDBLOCK+1
0074: A5 9F     1061 RDHUNK3  LDA  RDPUFF+1
0076: C5 0B     1062          CMP  PUFFEND2
0078: 90 EB     1063          BCC  RDHUNK2
007A: 18        1064          CLC             ;C=0!
007B: 60        1065 RDHUNK4  RTS
                1066 *
                1067 * Write
                1068 * --------
                1069 *
007C: 85 AC     1070 WRHUNK1  STA  WRPUFF+1
007E: 20 A2 00  1071 WRHUNK2  JSR  WRITER
0081: B0 11     1072          BCS  WRHUNK4
0083: E6 AC     1073          INC  WRPUFF+1
0085: E6 AC     1074          INC  WRPUFF+1
0087: E6 AD     1075          INC  WRBLOCK
0089: D0 02     1076          BNE  WRHUNK3
008B: E6 AE     1077          INC  WRBLOCK+1
008D: A5 AC     1078 WRHUNK3  LDA  WRPUFF+1
008F: C5 0B     1079          CMP  PUFFEND2
0091: 90 EB     1080          BCC  WRHUNK2
0093: 18        1081          CLC             ;C=0!
0094: 60        1082 WRHUNK4  RTS
                1083 *
```

```
                    1084 * Read-Block
                    1085 *
0095: 20 00 BF      1086 READER   JSR  MLI
0098: 80            1087 RDCOM    HEX  80          ;Read
0099: 9C 00         1088          DA   RDCOUNT
009B: 60            1089          RTS              ;C=0/1
                    1090 *
009C: 03            1091 RDCOUNT  HEX  03
009D: 60            1092 RDUNIT   HEX  60
009E: 00 16         1093 RDPUFF   HEX  0016        ;LLHH
00A0: 00 00         1094 RDBLOCK  HEX  0000        ;LLHH
                    1095 *
                    1096 * Write-Block
                    1097 *
00A2: 20 00 BF      1098 WRITER   JSR  MLI
00A5: 81            1099 WRCOM    HEX  81          ;Write
00A6: A9 00         1100          DA   WRCOUNT
00A8: 60            1101          RTS              ;C=0/1
                    1102 *
00A9: 03            1103 WRCOUNT  HEX  03
00AA: E0            1104 WRUNIT   HEX  E0
00AB: 00 16         1105 WRPUFF   HEX  0016        ;LLHH
00AD: 00 00         1106 WRBLOCK  HEX  0000        ;LLHH

1881 Bytes
```

3.5. Formatierprogramm

3.5.1. FORMAT.A und FORMAT.O

Dieses Programm, das mit RUN FORMAT.A gestartet wird und das seinerseits das Maschinenprogramm FORMAT.O einlädt, liest zunächst von der Startdiskette im Original-Slot/Drive die Boot-Blocks 0 und 1 ein und formatiert dann nach der Eingabe eines legalen Volume-Namens (maximal 15 Zeichen usw.) die Diskette im Format-Slot/Drive in der gewünschten Spurenanzahl. Diese Werte können durch Pokes in den Zeilen 110-170 geändert werden. FORMAT läuft unter allen ProDOS-Versionen ohne Patches.

FORMAT.A

```
100  PRINT  CHR$ (21): HOME : PRINT  CHR$ (4)"BLOAD
     FORMAT.O"
110  POKE 768,6: REM Original-Slot
120  POKE 769,1: REM Original-Drive
130  CALL 31226: REM Original-Bootblocks einlesen
140  F =  PEEK (773): IF F > 0 THEN  PRINT "Fehler-Nr.
     ";F: END
150  POKE 770,6: REM Format-Slot
160  POKE 771,2: REM Format-Drive
170  POKE 772,35: REM Spuren
180  HOME : INVERSE : PRINT "PRODOS.FORMATTER": NORMAL :
     PRINT
190  PRINT "Zu formatierende Diskette in"
200  PRINT "Slot " PEEK (770);", Drive " PEEK (771);"
     einlegen"; CHR$ (7)
210  VTAB 7: HTAB 1: CALL  - 958: INPUT "VOLUMENAME:";V$
220  V =  LEN (V$):F = 0: IF V = 0 OR V > 15 THEN F = 1:
     GOTO 270
230  FOR X = 1 TO V
240  X$ =  MID$ (V$,X,1): IF X$ >  = "A" AND X$ <  = "Z"
     OR X$ >  = "0" AND X$ <  = "9" AND X > 1 OR X$ = "."
     AND X > 1 THEN 260
250  F = 1
260  NEXT
270  IF F = 1 THEN  PRINT : PRINT "Illegaler Volumename
     ";: GET X$: GOTO 210
280  POKE 774,V: REM Namenslänge
290  FOR X = 1 TO V: POKE 774 + X, ASC ( MID$ (V$,X,1)):
     NEXT
300  CALL 31229: REM Formatieren
310  F =  PEEK (773): IF F > 0 THEN  PRINT "Fehler-Nr.
     ";F: END
320  PRINT: PRINT "Erledigt": PRINT : PRINT "Erneut J/N ";
330  GET X$: ON X$ = "J" GOTO 180: ON X$ <  > "N" GOTO
     330: HOME : END
```

FORMAT.O

```
1. BLOAD FORMAT.O
2. BLOAD FILER, TSYS, A$7900, B$5900, L$043A
3. BSAVE FORMAT.O, A31226, L1470
```

```
                        1               ORG   $79FA
                        2    *
                        3    * FORMAT.O
                        4    * ========
                        5    *
                        6    * 21.04.85
                        7    *
                        8    MLI        EQU   $BF00
                        9    *
                        10   OSLOT      EQU   $0300       ;768
                        11   ODRIVE     EQU   $0301       ;769
                        12   DSLOT      EQU   $0302       ;770
                        13   DDRIVE     EQU   $0303       ;771
                        14   SPUREN     EQU   $0304       ;772
                        15   ERRNUM     EQU   $0305       ;773
                        16   NAMELEN    EQU   $0306       ;774
                        17   *
                        18   BLK0       EQU   $8000
                        19   BLK1       EQU   $8200
                        20   BLKX       EQU   $8400
                        21   *
                        22   *
                        23   * Programm: $79FA-$7FFF
                        24   * Puffer:   $8000-$8600
                        25   *
79FA: 4C 3A 7E          26             JMP   READBOOT    ;31226
79FD: 4C 62 7E          27             JMP   INITDISK    ;31229
                        28   *-----------------------------
                        29   * Format-Befehl aus "FILER"
                        30   * der Firma Apple.
                        31   * Hier nicht gelistet.
                        632  *-----------------------------
                        633  *
7E3A: AD 00 03          634  READBOOT  LDA   OSLOT
7E3D: AE 01 03          635            LDX   ODRIVE
7E40: 20 3B 7F          636            JSR   SLDR1
7E43: 8D 77 7F          637            STA   RDUNIT
7E46: A9 80             638            LDA   #>BLK0
7E48: 8D 79 7F          639            STA   RDPUFF+1
7E4B: A9 00             640            LDA   #0          ;Blk 0
7E4D: 8D 05 03          641            STA   ERRNUM
7E50: 8D 7A 7F          642            STA   RDBLCK
7E53: 20 6D 7F          643            JSR   RDBLOCK
                        644  *
7E56: A9 82             645            LDA   #>BLK1
7E58: 8D 79 7F          646            STA   RDPUFF+1
7E5B: EE 7A 7F          647            INC   RDBLCK      ;Blk 1
7E5E: 20 6D 7F          648            JSR   RDBLOCK
```

```
7E61: 60          649           RTS             ;Exit
                  650  *
                  651  *---------------------------
                  652  *
7E62: 20 91 7F    653  INITDISK JSR  ZLOAD
7E65: AD 04 03    654           LDA  SPUREN
7E68: 8D F4 7A    655           STA  SPURNR+1
7E6B: A9 00       656           LDA  #0
7E6D: 8D 05 03    657           STA  ERRNUM
7E70: AD 02 03    658           LDA  DSLOT
7E73: AE 03 03    659           LDX  DDRIVE
7E76: 20 3B 7F    660           JSR  SLDR1
7E79: 8D 86 7F    661           STA  WRUNIT
7E7C: 20 00 7A    662           JSR  FORMAT
7E7F: 08          663           PHP
7E80: 48          664           PHA
7E81: 20 9C 7F    665           JSR  ZSTORE
7E84: 68          666           PLA
7E85: 28          667           PLP
7E86: 90 04       668           BCC  BLOCK0
7E88: 8D 05 03    669           STA  ERRNUM
7E8B: 60          670           RTS
                  671  *
                  672  * Urlader Block 0-1
                  673  *
7E8C: A9 80       674  BLOCK0   LDA  #>BLK0
7E8E: 8D 88 7F    675           STA  WRPUFF+1
7E91: A9 00       676           LDA  #0           ;Blk 0
7E93: 8D 89 7F    677           STA  WRBLCK
7E96: 20 7C 7F    678           JSR  WRBLOCK
                  679  *
7E99: A9 82       680  BLOCK1   LDA  #>BLK1
7E9B: 8D 88 7F    681           STA  WRPUFF+1
7E9E: EE 89 7F    682           INC  WRBLCK       ;Blk 1
7EA1: 20 7C 7F    683           JSR  WRBLOCK
                  684  *
                  685  * Volume Directory Block 2-5
                  686  *
7EA4: 20 47 7F    687  BLOCK2   JSR  CLR1
7EA7: A9 03       688           LDA  #3           ;Nxt Blk
7EA9: 8D 02 84    689           STA  BLKX+2
                  690  *
7EAC: AD 06 03    691           LDA  NAMELEN
7EAF: AA          692           TAX
7EB0: 09 F0       693           ORA  #$F0         ;Vol-Dir
7EB2: 8D 04 84    694           STA  BLKX+$04
7EB5: BD 06 03    695  NAME     LDA  NAMELEN,X
7EB8: 9D 04 84    696           STA  BLKX+$04,X
7EBB: CA          697           DEX
7EBC: D0 F7       698           BNE  NAME
                  699  *
7EBE: A9 C3       700           LDA  #$C3         ,Access
7EC0: 8D 22 84    701           STA  BLKX+$22
                  702  *
7EC3: A9 27       703           LDA  #$27         ;Entr.Len
```

```
7EC5: 8D 23 84  704           STA  BLKX+$23
                705  *
7EC8: A9 0D     706           LDA  #$0D        ;Entries
7ECA: 8D 24 84  707           STA  BLKX+$24
                708  *
7ECD: A9 06     709           LDA  #$06        ;Bitmap-P.
7ECF: 8D 27 84  710           STA  BLKX+$27
                711  *
7ED2: 20 56 7F  712           JSR  MULT1       ;Blckanz.
7ED5: AD 54 7F  713           LDA  TNUML
7ED8: 8D 29 84  714           STA  BLKX+$29
7EDB: AD 55 7F  715           LDA  TNUMH
7EDE: 8D 2A 84  716           STA  BLKX+$2A
                717  *
7EE1: A9 84     718           LDA  #>BLKX
7EE3: 8D 88 7F  719           STA  WRPUFF+1
7EE6: EE 89 7F  720           INC  WRBLCK      ;Blk 2
7EE9: 20 7C 7F  721           JSR  WRBLOCK
                722  *
7EEC: 20 47 7F  723  BLOCK3   JSR  CLR1
7EEF: A9 02     724           LDA  #2          ;Pre Blk
7EF1: 8D 00 84  725           STA  BLKX
7EF4: A9 04     726           LDA  #4          ;Nxt Blk
7EF6: 8D 02 84  727           STA  BLKX+$02
7EF9: EE 89 7F  728           INC  WRBLCK      ;Blk 3
7EFC: 20 7C 7F  729           JSR  WRBLOCK
                730  *
7EFF: 20 47 7F  731  BLOCK4   JSR  CLR1
7F02: A9 03     732           LDA  #3          ;Pre Blk
7F04: 8D 00 84  733           STA  BLKX
7F07: A9 05     734           LDA  #5          ;Nxt Blk
7F09: 8D 02 84  735           STA  BLKX+2
7F0C: EE 89 7F  736           INC  WRBLCK      ;Blk 4
7F0F: 20 7C 7F  737           JSR  WRBLOCK
                738  *
7F12: 20 47 7F  739  BLOCK5   JSR  CLR1
7F15: A9 04     740           LDA  #4          ;Pre Blk
7F17: 8D 00 84  741           STA  BLKX
7F1A: EE 89 7F  742           INC  WRBLCK      ;Blk 5
7F1D: 20 7C 7F  743           JSR  WRBLOCK
                744  *
                745  * Volume Bit Map Block 6
                746  *
7F20: 20 47 7F  747  BLOCK6   JSR  CLR1
7F23: A9 FF     748           LDA  #%11111111
7F25: AE 04 03  749           LDX  SPUREN
7F28: CA        750           DEX              ;35-1
7F29: 9D 00 84  751  BLOCK6A  STA  BLKX,X
7F2C: CA        752           DEX
7F2D: D0 FA     753           BNE  BLOCK6A
7F2F: A9 01     754           LDA  #%00000001
7F31: 8D 00 84  755           STA  BLKX
7F34: EE 89 7F  756           INC  WRBLCK      ;Blk 6
7F37: 20 7C 7F  757           JSR  WRBLOCK
7F3A: 60        758           RTS              ;Exit
```

```
                        759  *
                        760  *-----------------------------
                        761  *
                        762  * Slot/Drive in UNIT
                        763  *
7F3B: 0A                764  SLDR1    ASL
7F3C: 0A                765          ASL
7F3D: 0A                766          ASL
7F3E: 0A                767          ASL
7F3F: E0 01             768          CPX  #1          ;Drive 1
7F41: F0 02             769          BEQ  SLDR2
7F43: 09 80             770          ORA  #%10000000
7F45: 60                771  SLDR2    RTS
7F46: 60                772          RTS
                        773  *
                        774  * Puffer für Blocks 2-6 löschen
                        775  *
7F47: A9 00             776  CLR1     LDA  #0
7F49: AA                777          TAX
7F4A: 9D 00 84          778  CLR2     STA  BLKX,X
7F4D: 9D 00 85          779          STA  BLKX+$100,X
7F50: E8                780          INX
7F51: D0 F7             781          BNE  CLR2
7F53: 60                782          RTS
                        783  *
                        784  * Blockanzahl = Tracks mal 16
                        785  * HH LL 3 mal links-shiften
                        786  *
7F54: 00                787  TNUML    HEX  00
7F55: 00                788  TNUMH    HEX  00
                        789  *
7F56: A9 00             790  MULT1    LDA  #0
7F58: 8D 55 7F          791          STA  TNUMH
7F5B: AD 04 03          792          LDA  SPUREN
7F5E: 8D 54 7F          793          STA  TNUML
7F61: A2 03             794          LDX  #3
7F63: 0E 54 7F          795  MULT2    ASL  TNUML
7F66: 2E 55 7F          796          ROL  TNUMH
7F69: CA                797          DEX
7F6A: D0 F7             798          BNE  MULT2
7F6C: 60                799          RTS
                        800  *
                        801  * Block-Read/Write
                        802  *
7F6D: 20 00 BF          803  RDBLOCK  JSR  MLI
7F70: 80                804          HEX  80          ;Read
7F71: 76 7F             805          DA   RDCNT
7F73: B0 16             806          BCS  RDWRERR
7F75: 60                807          RTS
7F76: 03                808  RDCNT    HEX  03
7F77: E0                809  RDUNIT   HEX  E0          ;S6,D2
7F78: 00 00             810  RDPUFF   HEX  0000        ;LLHH
7F7A: 00 00             811  RDBLCK   HEX  0000        ;LLHH
                        812  *
7F7C: 20 00 BF          813  WRBLOCK  JSR  MLI
```

```
7F7F: 81          814             HEX  81          ;Write
7F80: 85 7F       815             DA   WRCNT
7F82: B0 07       816             BCS  RDWRERR
7F84: 60          817             RTS
7F85: 03          818  WRCNT      HEX  03
7F86: 60          819  WRUNIT     HEX  60          ;S6,D1
7F87: 00 00       820  WRPUFF     HEX  0000        ;LLHH
7F89: 00 00       821  WRBLCK     HEX  0000        ;LLHH
                  822  *
7F8B: 8D 05 03    823  RDWRERR    STA  ERRNUM
7F8E: 68          824             PLA
7F8F: 68          825             PLA
7F90: 60          826             RTS
                  827  *
                  828  * Zero-Page retten
                  829  *
7F91: A2 0F       830  ZLOAD      LDX  #$0F
7F93: B5 D0       831  ZLOAD1     LDA  ZD0,X
7F95: 9D A7 7F    832             STA  ZER0,X
7F98: CA          833             DEX
7F99: 10 F8       834             BPL  ZLOAD1
7F9B: 60          835             RTS
                  836  *
7F9C: A2 0F       837  ZSTORE     LDX  #$0F
7F9E: BD A7 7F    838  ZSTORE1    LDA  ZER0,X
7FA1: 95 D0       839             STA  ZD0,X
7FA3: CA          840             DEX
7FA4: 10 F8       841             BPL  ZSTORE1
7FA6: 60          842             RTS
                  843  *
                  844  ZER0       DS   16
7FB7: 00          845             HEX  00

1470 Bytes
```

Die vorangehenden Programme enthalten die Formatierungsroutine des FILER-Programms der Firma Apple als Kleinzitat. Eine neue Formatierungsroutine von Arne Schäpers findet sich im „Peeker", Heft 7/1985.

3.6. Diskettenvergleichsprogramm

3.6.1. DISKCOMPARER

Dieses Programm, das mit BRUN DISKCOMPARER gestartet wird, vergleicht zwei Disketten in OSLOT/ODRIVE (= Original) und DSLOT/DDRIVE (= Duplikat) miteinander, wobei die Original-Blocks im OPUF $1000-$4FFF und die Duplikat-Blocks im DPUF $5000-$8FFF abgelegt werden (PUF = Puffer). Die gewünschten OSLOT/ODRIVE/DSLOT/DDRIVE-Werte können in die Speicherstellen 3003-3006 gepokt werden (s. Listing). Den Vergleichsvorgang kann man durch Drücken von ESC abbrechen. Übrigens kann man auch eine Diskette mit sich selbst vergleichen, wenn OSLOT/ODRIVE = DSLOT/DDRIVE (Lesetest). Die Anzahl der Blocks (z. Zt. 280) kann modifiziert werden.

DISKCOMPARER

```
BSAVE DISKCOMPARER, A3000, L464

                    1             ORG  3000
                    2     *
                    3     * DISKCOMPARER
                    4     * ============
                    5     *
                    6     * von U.Stiehl/14.04.85
                    7     *
                    8     * Vergleicht 2 ProDOS-Disketten
                    9     * - "Original" und "Duplikat" -
                   10     * auf ihre Identität.
                   11     * Vorher bei Bedarf
                   12     * OSLOT  = Original-Slot
                   13     * ODRIVE = Original-Drive
                   14     * DSLOT  = Duplikat-Slot
                   15     * DDRIVE = Duplikat-Drive
                   16     * poken. Ferner könnte man für
                   17     * größere Laufwerke die Block-
                   18     * gesamtzahl (hier 280) poken.
                   19     *
                   20     INDO      EQU  $FA
                   21     INDD      EQU  $FC
                   22     HOME      EQU  $FC58
                   23     COUT      EQU  $FDED
                   24     PRBYTE    EQU  $FDDA
                   25     PRERR     EQU  $FF2D
                   26     RDKEY     EQU  $FD0C
                   27     MLI       EQU  $BF00
                   28     *
0BB8: 4C C8 0B     29               JMP  START1
                   30     *
```

```
0BBB: 06          31   OSLOT    HEX  06          ;3003
0BBC: 01          32   ODRIVE   HEX  01          ;3004
0BBD: 06          33   DSLOT    HEX  06          ;3005
0BBE: 02          34   DDRIVE   HEX  02          ;3006
0BBF: 18 01       35   BLOCKS   DA   $0118       ;280
0BC1: 00          36   ERROR    HEX  00
0BC2: 60          37   OUNIT    HEX  60          ;S6,D1
0BC3: E0          38   DUNIT    HEX  E0          ;S6,D2
                  39   *
                  40   * $1000-$4FFF: Original-Puffer
                  41   * $5000-$8FFF: Duplikat-Puffer
                  42   *
0BC4: 10          43   PUFBEG0  HEX  10          ;$1000
0BC5: 50          44   PUFEND0  HEX  50          ;$5000
0BC6: 50          45   PUFBEGD  HEX  50          ;$5000
0BC7: 90          46   PUFENDD  HEX  90          ;$9000
                  47   *
0BC8: 20 58 FC    48   START1   JSR  HOME
0BCB: A2 00       49            LDX  #0
0BCD: BD 62 0D    50   START2   LDA  STRING,X
0BD0: F0 06       51            BEQ  START3
0BD2: 20 ED FD    52            JSR  COUT
0BD5: E8          53            INX
0BD6: D0 F5       54            BNE  START2
0BD8: 20 0C FD    55   START3   JSR  RDKEY
0BDB: C9 CA       56            CMP  #"J"
0BDD: F0 0D       57            BEQ  SLDR1
0BDF: C9 EA       58            CMP  #"j"
0BE1: F0 09       59            BEQ  SLDR1
0BE3: C9 CE       60            CMP  #"N"
0BE5: F0 04       61            BEQ  START4
0BE7: C9 EE       62            CMP  #"n"
0BE9: D0 ED       63            BNE  START3
0BEB: 60          64   START4   RTS             ;Exit
                  65   *
                  66   * Slot/Drive in Unit-Nr. packen
                  67   *
0BEC: AD BB 0B    68   SLDR1    LDA  OSLOT
0BEF: AE BC 0B    69            LDX  ODRIVE
0BF2: 20 0E 0C    70            JSR  SLDR2
0BF5: 8D C2 0B    71            STA  OUNIT
0BF8: AD BD 0B    72            LDA  DSLOT
0BFB: AE BE 0B    73            LDX  DDRIVE
0BFE: 20 0E 0C    74            JSR  SLDR2
0C01: 8D C3 0B    75            STA  DUNIT
                  76   *
0C04: 20 58 FC    77            JSR  HOME
0C07: A9 00       78            LDA  #0
0C09: 8D C1 0B    79            STA  ERROR
0C0C: F0 17       80            BEQ  BLOCK0
                  81   *
0C0E: 0A          82   SLDR2    ASL
0C0F: 0A          83            ASL
0C10: 0A          84            ASL
0C11: 0A          85            ASL
```

```
0C12: E0 01     86              CPX  #1          ;Drive 1
0C14: F0 02     87              BEQ  SLDR3
0C16: 09 80     88              ORA  #%10000000
0C18: 60        89  SLDR3       RTS
                90  *
                91  * MLI-Fehler-Exit
                92  *
0C19: 20 58 FC  93  MLIERR      JSR  HOME
0C1C: 20 2D FF  94              JSR  PRERR
0C1F: AD C1 0B  95              LDA  ERROR
0C22: 4C DA FD  96              JMP  PRBYTE     ;Exit
                97  *
                98  * Mit Block $0000 anfangen
                99  *
0C25: 8D 60 0D  100 BLOCK0      STA  BLOCKCNT
0C28: 8D 61 0D  101             STA  BLOCKCNT+1
                102 *
                103 * Original lesen
                104 *
0C2B: AD C4 0B  105 RDORIG1     LDA  PUFBEG0
0C2E: 8D 5D 0D  106             STA  RDPUFF+1
0C31: AD C2 0B  107             LDA  OUNIT
0C34: 8D 5B 0D  108             STA  RDUNIT
0C37: AD 60 0D  109             LDA  BLOCKCNT
0C3A: 8D 5E 0D  110             STA  RDBLOCK
0C3D: AD 61 0D  111             LDA  BLOCKCNT+1
0C40: 8D 5F 0D  112             STA  RDBLOCK+1
0C43: 20 4E 0D  113 RDORIG2     JSR  READER
0C46: B0 D1     114             BCS  MLIERR
0C48: EE 5D 0D  115             INC  RDPUFF+1
0C4B: EE 5D 0D  116             INC  RDPUFF+1
0C4E: EE 5E 0D  117             INC  RDBLOCK
0C51: D0 03     118             BNE  RDORIG3
0C53: EE 5F 0D  119             INC  RDBLOCK+1
0C56: 38        120 RDORIG3     SEC
0C57: AD BF 0B  121             LDA  BLOCKS     ;LL
0C5A: E9 01     122             SBC  #1         ;280-1
0C5C: ED 5E 0D  123             SBC  RDBLOCK    ;LL
0C5F: AD C0 0B  124             LDA  BLOCKS+1   ;HH
0C62: ED 5F 0D  125             SBC  RDBLOCK+1  ;HH
0C65: 90 08     126             BCC  RDDUPL1    ;Overflow
0C67: AD 5D 0D  127             LDA  RDPUFF+1
0C6A: CD C5 0B  128             CMP  PUFEND0
0C6D: 90 D4     129             BCC  RDORIG2
                130 *
                131 * Duplikat lesen
                132 *
0C6F: AD C6 0B  133 RDDUPL1     LDA  PUFBEGD
0C72: 8D 5D 0D  134             STA  RDPUFF+1
0C75: AD C3 0B  135             LDA  DUNIT
0C78: 8D 5B 0D  136             STA  RDUNIT
0C7B: AD 60 0D  137             LDA  BLOCKCNT
0C7E: 8D 5E 0D  138             STA  RDBLOCK
0C81: AD 61 0D  139             LDA  BLOCKCNT+1
0C84: 8D 5F 0D  140             STA  RDBLOCK+1
```

```
0C87: 20 4E 0D  141 RDDUPL2 JSR  READER
0C8A: B0 8D     142         BCS  MLIERR
0C8C: EE 5D 0D  143         INC  RDPUFF+1
0C8F: EE 5D 0D  144         INC  RDPUFF+1
0C92: EE 5E 0D  145         INC  RDBLOCK
0C95: D0 03     146         BNE  RDDUPL3
0C97: EE 5F 0D  147         INC  RDBLOCK+1
0C9A: 38        148 RDDUPL3 SEC
0C9B: AD BF 0B  149         LDA  BLOCKS      ;LL
0C9E: E9 01     150         SBC  #1          ;280-1
0CA0: ED 5E 0D  151         SBC  RDBLOCK     ;LL
0CA3: AD C0 0B  152         LDA  BLOCKS+1    ;HH
0CA6: ED 5F 0D  153         SBC  RDBLOCK+1   ;HH
0CA9: 90 0E     154         BCC  COMP1       ;Overflow
0CAB: AD 5D 0D  155         LDA  RDPUFF+1
0CAE: CD C7 0B  156         CMP  PUFENDD
0CB1: 90 D4     157         BCC  RDDUPL2
0CB3: 20 BF 0C  158         JSR  COMP2
0CB6: 4C 2B 0C  159         JMP  RDORIG1     ;erneut
                160 *
                161 * Blockpuffer vergleichen
                162 *
0CB9: 20 BF 0C  163 COMP1   JSR  COMP2
0CBC: 4C C8 0B  164         JMP  START1      ;Exit
0CBF: A0 00     165 COMP2   LDY  #0
0CC1: 84 FA     166         STY  INDO
0CC3: 84 FC     167         STY  INDD
0CC5: AD C4 0B  168         LDA  PUFBEG0
0CC8: 85 FB     169         STA  INDO+1
0CCA: AD C6 0B  170         LDA  PUFBEGD
0CCD: 85 FD     171         STA  INDD+1
                172 *
                173 * 1. Blockhälfte
                174 *
0CCF: A0 00     175 COMP3   LDY  #0
0CD1: B1 FA     176 COMP4   LDA  (INDO),Y
0CD3: D1 FC     177         CMP  (INDD),Y
0CD5: D0 31     178         BNE  COMP9
0CD7: C8        179         INY
0CD8: D0 F7     180         BNE  COMP4
0CDA: E6 FB     181         INC  INDO+1
0CDC: E6 FD     182         INC  INDD+1
                183 *
                184 * 2. Blockhälfte
                185 *
0CDE: B1 FA     186 COMP5   LDA  (INDO),Y
0CE0: D1 FC     187         CMP  (INDD),Y
0CE2: D0 20     188         BNE  COMP8
0CE4: C8        189         INY
0CE5: D0 F7     190         BNE  COMP5
0CE7: E6 FB     191         INC  INDO+1
0CE9: E6 FD     192         INC  INDD+1
                193 *
                194 * Blockzähler erhöhen
                195 *
```

```
0CEB: EE 60 0D  196  COMP6    INC  BLOCKCNT
0CEE: D0 03     197          BNE  COMP7
0CF0: EE 61 0D  198          INC  BLOCKCNT+1
0CF3: AD 61 0D  199  COMP7    LDA  BLOCKCNT+1
0CF6: CD 5F 0D  200          CMP  RDBLOCK+1
0CF9: 90 D4     201          BCC  COMP3
0CFB: AD 60 0D  202          LDA  BLOCKCNT
0CFE: CD 5E 0D  203          CMP  RDBLOCK
0D01: 90 CC     204          BCC  COMP3
0D03: 60        205          RTS
                206  *
                207  * Anweichenden Blockpuffer in
                208  * folgender Form anzeigen:
                209  *
                210  * B0000-1000-5000 Beispiel
                211  * BLOCK-PUF0-PUFD
                212  *
0D04: C6 FB     213  COMP8    DEC  IND0+1
0D06: C6 FD     214          DEC  INDD+1
0D08: A9 C2     215  COMP9    LDA  #"B"
0D0A: 20 ED FD  216          JSR  COUT
0D0D: AD 61 0D  217          LDA  BLOCKCNT+1
0D10: 20 DA FD  218          JSR  PRBYTE
0D13: AD 60 0D  219          LDA  BLOCKCNT
0D16: 20 DA FD  220          JSR  PRBYTE
0D19: A9 AD     221          LDA  #"-"
0D1B: 20 ED FD  222          JSR  COUT
0D1E: A5 FB     223          LDA  IND0+1
0D20: 20 DA FD  224          JSR  PRBYTE
0D23: A9 00     225          LDA  #0
0D25: 20 DA FD  226          JSR  PRBYTE
0D28: A9 AD     227          LDA  #"-"
0D2A: 20 ED FD  228          JSR  COUT
0D2D: A5 FD     229          LDA  INDD+1
0D2F: 20 DA FD  230          JSR  PRBYTE
0D32: A9 00     231          LDA  #0
0D34: 20 DA FD  232          JSR  PRBYTE
0D37: A9 8D     233          LDA  #$8D
0D39: 20 ED FD  234          JSR  COUT
0D3C: E6 FB     235          INC  IND0+1
0D3E: E6 FB     236          INC  IND0+1
0D40: E6 FD     237          INC  INDD+1
0D42: E6 FD     238          INC  INDD+1
                239  *
                240  * Bei ESC-Taste Exit
                241  *
0D44: 20 0C FD  242          JSR  RDKEY
0D47: C9 9B     243          CMP  #$9B       ;ESC
0D49: D0 A0     244          BNE  COMP6
0D4B: 68        245          PLA
0D4C: 68        246          PLA
0D4D: 60        247          RTS             ;Exit
                248  *
                249  * Read-Block
                250  *
```

```
0D4E: 20 00 BF   251  READER   JSR  MLI
0D51: 80         252           HEX  80          ;Read
0D52: 5A 0D      253           DA   RDCOUNT
0D54: 90 03      254           BCC  RDOKAY
0D56: 8D C1 0B   255           STA  ERROR
0D59: 60         256  RDOKAY   RTS
                 257  *
0D5A: 03         258  RDCOUNT  HEX  03
0D5B: 60         259  RDUNIT   HEX  60
0D5C: 00 16      260  RDPUFF   HEX  0016        ;LLHH
0D5E: 00 00      261  RDBLOCK  HEX  0000        ;LLHH
                 262  *
0D60: 00 00      263  BLOCKCNT HEX  0000
                 264  *
0D62: 04 09 13   265  STRING   INV  "DISKCOMPARER"
0D65: 0B 03 0F 0D 10 01 12 05
0D6D: 12
0D6E: 8D         266           HEX  8D
0D6F: 15 2E 20   267           INV  "U. STIEHL 85"
0D72: 13 14 09 05 08 0C 20 38
0D7A: 35
0D7B: 8D 8D      268           HEX  8D8D
0D7D: D3 D4 C1   269           ASC  "START J/N "
0D80: D2 D4 A0 CA AF CE A0
0D87: 00         270           HEX  00

464 Bytes
```

3.7. Bad-Block-Routine

3.7.1. BAD.BLOCKS

Dieses Programm, das mit BRUN BAD.BLOCKS gestartet wird, überprüft eine Diskette auf defekte Blocks. Die gewünschten Slot/Drive-Werte können in 771-772 gepokt werden (s. Listing).

BAD.BLOCKS

```
BSAVE BAD.BLOCKS, A768, L184
```

```
                        1               ORG   $0300       ;768
                        2    *
                        3    * BAD.BLOCKS
                        4    * ==========
                        5    *
                        6    * von U.Stiehl/14.04.85
                        7    *
                        8    HOME       EQU   $FC58
                        9    COUT       EQU   $FDED
                        10   PRBYTE     EQU   $FDDA
                        11   RDKEY      EQU   $FD0C
                        12   MLI        EQU   $BF00
                        13   *
0300: 4C 09 03          14              JMP   START1
                        15   *
0303: 06                16   SLOT       HEX   06          ;771
0304: 01                17   DRIVE      HEX   01          ;772
0305: 18 01             18   BLOCKS     DA    $0118       ;280
0307: 00 10             19   PUFFER     DA    $1000
                        20   *
0309: 20 58 FC          21   START1     JSR   HOME
030C: A2 00             22              LDX   #0
030E: BD 96 03          23   START2     LDA   STRING,X
0311: F0 06             24              BEQ   START3
0313: 20 ED FD          25              JSR   COUT
0316: E8                26              INX
0317: D0 F5             27              BNE   START2
0319: 20 0C FD          28   START3     JSR   RDKEY
031C: C9 CA             29              CMP   #"J"
031E: F0 05             30              BEQ   SLDR1
0320: C9 CE             31              CMP   #"N"
0322: D0 00             32              BNE   START4
0324: 60                33   START4     RTS               ;Exit
                        34   *
                        35   * Slot/Drive in Unit-Nr. packen
                        36   *
0325: AD 03 03          37   SLDR1      LDA   SLOT
0328: AE 04 03          38              LDX   DRIVE
032B: 0A                39              ASL
```

```
032C: 0A          40            ASL
032D: 0A          41            ASL
032E: 0A          42            ASL
032F: E0 01       43            CPX  #1          ;Drive 1
0331: F0 02       44            BEQ  SLDR2
0333: 09 80       45            ORA  #%10000000
0335: 8D 91 03    46  SLDR2     STA  RDUNIT
                  47  *
                  48  * Werte initialisieren
                  49  *
0338: 20 58 FC    50            JSR  HOME
033B: AD 08 03    51            LDA  PUFFER+1
033E: 8D 93 03    52            STA  RDPUFF+1
0341: A9 00       53            LDA  #0
0343: 8D 92 03    54            STA  RDPUFF
0346: 8D 94 03    55            STA  RDBLOCK
0349: 8D 95 03    56            STA  RDBLOCK+1
                  57  *
                  58  * Leseschleife
                  59  *
034C: 20 89 03    60  READ1     JSR  READER
034F: 90 1E       61            BCC  READ2
                  62  *
                  63  * Bad Block gefunden
                  64  *
0351: A9 C2       65            LDA  #"B"
0353: 20 ED FD    66            JSR  COUT
0356: AD 95 03    67            LDA  RDBLOCK+1
0359: 20 DA FD    68            JSR  PRBYTE
035C: AD 94 03    69            LDA  RDBLOCK
035F: 20 DA FD    70            JSR  PRBYTE
0362: A9 8D       71            LDA  #$8D
0364: 20 ED FD    72            JSR  COUT
                  73  *
                  74  * Bei ESC-Taste Exit
                  75  *
0367: 20 0C FD    76            JSR  RDKEY
036A: C9 9B       77            CMP  #$9B        ;ESC
036C: D0 01       78            BNE  READ2
036E: 60          79            RTS              ;Exit
                  80  *
036F: EE 94 03    81  READ2     INC  RDBLOCK
0372: D0 03       82            BNE  READ3
0374: EE 95 03    83            INC  RDBLOCK+1
0377: AD 95 03    84  READ3     LDA  RDBLOCK+1
037A: CD 06 03    85            CMP  BLOCKS+1
037D: 90 CD       86            BCC  READ1
037F: AD 94 03    87            LDA  RDBLOCK
0382: CD 05 03    88            CMP  BLOCKS
0385: 90 C5       89            BCC  READ1
0387: B0 80       90            BCS  START1
                  91  *
                  92  * Read-Block
                  93  *
0389: 20 00 BF    94  READER    JSR  MLI
```

```
038C: 80          95              HEX  80          ;Read
038D: 90 03       96              DA   RDCOUNT
038F: 60          97              RTS
                  98   *
0390: 03          99   RDCOUNT    HEX  03
0391: 60          100  RDUNIT     HEX  60
0392: 00 16       101  RDPUFF     HEX  0016        ;LLHH
0394: 00 00       102  RDBLOCK    HEX  0000        ;LLHH
                  103  *
0396: 02 01 04    104  STRING     INV  "BAD.BLOCKS"
0399: 2E 02 0C 0F 03 0B 13
03A0: 8D          105              HEX  8D
03A1: 15 2E 13    106              INV  "U.STIEHL85"
03A4: 14 09 05 08 0C 38 35
03AB: 8D 8D       107              HEX  8D8D
03AD: D3 D4 C1    108              ASC  "START J/N "
03B0: D2 D4 A0 CA AF CE A0
03B7: 00          109              HEX  00

184 Bytes
```

3.8. Blockeditor-Routine

3.8.1. BLOCK.EDIT

Dieses Programm ist kein ausgewachsener Block-Editor, sondern quasi ein „Quick-and-dirty"-Editor für Assemblerprogrammierer. BLOCK.EDIT wird mit BLOAD BLOCK.EDIT installiert. Danach kann man vom Monitor aus mit

300GhhllR lesen und

300GhhllW schreiben

„hhll" steht für die Block-Nummer. Der in den Editierpuffer $1000-$11FF eingelesene Block wird als Hex-Dump angezeigt. Slot und Drive lassen sich bei $0303 und $0304 ändern.

BLOCKEDIT

```
BSAVE BLOCKEDIT, A768, L208
```

```
1              ORG  $0300
2    *
3    * BLOCKEDIT  US/23.4.85
4    * ==========
5    *
6    * 300Ghhl1R liest Block    $hhl1
7    * 300Ghhl1W schreibt Block $hhl1
8    *
9    * Beispiele:
10   * 300G1R    liest Block $0001
11   * 300G10R   liest Block $0010
12   * 300G101R  liest Block $0101
13   * 300G0111R liest Block $0111
```

```
                        14  *
0300: 4C 58 03          15          JMP   COMMAND1
                        16  *
0303: 06                17  SLOT    HEX   06
0304: 01                18  DRIVE   HEX   01
0305: 00                19  BLOCKL  HEX   00
0306: 00                20  BLOCKH  HEX   00
0307: 10                21  PUFFANF HEX   10          ;$1000-$11FF
                        22  *
                        23  YSAV    EQU   $34
                        24  A2L     EQU   $3E
                        25  A2H     EQU   $3F
                        26  IND     EQU   $CE
                        27  PUFFEND EQU   $FE
                        28  COUT    EQU   $FDED
                        29  PRBYTE  EQU   $FDDA
                        30  CROUT   EQU   $FD8E
                        31  GETNUM  EQU   $FFA7
                        32  MON     EQU   $FF65
                        33  MONZ    EQU   $FF69
                        34  MLI     EQU   $BF00
                        35  *
                        36  * Block lesen/schreiben
                        37  * READ1  = CALL 776
                        38  * WRITE1 = CALL 780
                        39  *
0308: A9 80             40  READ1   LDA   #$80        ;80=Read
030A: D0 02             41          BNE   RDWR1
030C: A9 81             42  WRITE1  LDA   #$81        ;81=Write
030E: 8D 39 03          43  RDWR1   STA   RW
                        44  *
0311: AD 05 03          45          LDA   BLOCKL      ;Block-Nr.
0314: 8D 56 03          46          STA   RWBLCK
0317: AD 06 03          47          LDA   BLOCKH
031A: 8D 57 03          48          STA   RWBLCK+1
031D: AD 07 03          49          LDA   PUFFANF     ;Puffer
0320: 8D 55 03          50          STA   RWPUFF+1
                        51  *
0323: AD 03 03          52          LDA   SLOT        ;Unit-Nr.
0326: AE 04 03          53          LDX   DRIVE
0329: 0A                54          ASL
032A: 0A                55          ASL
032B: 0A                56          ASL
032C: 0A                57          ASL
032D: E0 01             58          CPX   #1          ;Drive 1
032F: F0 02             59          BEQ   UNIT
0331: 09 80             60          ORA   #$80        ;Drive 2
0333: 8D 53 03          61  UNIT    STA   RWUNIT
                        62  *
0336: 20 00 BF          63          JSR   MLI
0339: 80                64  RW      HEX   80          ;81
033A: 52 03             65          DA    RWCNT
033C: 90 06             66          BCC   RWOKAY
033E: 20 DA FD          67          JSR   PRBYTE      ;Fehler
```

```
0341: 4C 76 03  68            JMP  ERROR
0344: AD 39 03  69   RWOKAY   LDA  RW
0347: C9 81     70            CMP  #$81        ;Write?
0349: F0 04     71            BEQ  RWENDE
                72   *
                73   * Hier ggf. RTS, falls Anzeige
                74   * unterdrückt werden soll.
                75   *
034B: EA        76            NOP              ;RTS
034C: 20 7D 03  77            JSR  DISP1
034F: 4C 79 03  78   RWENDE   JMP  EXIT
0352: 03        79   RWCNT    HEX  03
0353: 60        80   RWUNIT   HEX  60          ;S6,D1
0354: 00 00     81   RWPUFF   HEX  0000        ;LLHH
0356: 00 00     82   RWBLCK   HEX  0000        ;LLHH
                83   *
                84   * Befehl auswerten
                85   *
0358: A4 34     86   COMMAND1 LDY  YSAV
035A: 20 A7 FF  87            JSR  GETNUM
035D: 88        88            DEY              ;Y=Y-1
035E: C4 34     89            CPY  YSAV
0360: F0 14     90            BEQ  ERROR       ;leer!
                91   *
0362: 48        92            PHA              ;->A
0363: A5 3E     93            LDA  A2L
0365: 8D 05 03  94            STA  BLOCKL
0368: A5 3F     95            LDA  A2H
036A: 8D 06 03  96            STA  BLOCKH
                97   *
                98   * Nach GETNUM ist Nicht-Hexzahl
                99   * A wie folgt modifiziert:
                100  * A EOR #$B0 SEC ADC #$88 -> A
                101  * Dies ergibt EB für "R"
                102  *          und F0 für "W"
                103  *
036D: 68        104           PLA              ;A<-
036E: C9 EB     105           CMP  #$EB        ;"R"
0370: F0 96     106           BEQ  READ1
0372: C9 F0     107           CMP  #$F0        ;"W"
0374: F0 96     108           BEQ  WRITE1
0376: 4C 65 FF  109  ERROR    JMP  MON         ;falsch
                110  *
                111  * Hier ggf. RTS, falls Parameter
                112  * von Applesoft gepokt und
                113  * Routine mit CALL 776/780
                114  * aufgerufen wird.
                115  *
0379: EA        116  EXIT     NOP              ;RTS
037A: 4C 69 FF  117           JMP  MONZ        ;okay
                118  *
                119  * Puffer-Hexdump
                120  * mit 32 Hexzahlen/Zeile
                121  *
```

```
037D: 20 8E FD  122  DISP1   JSR  CROUT
0380: A0 05     123          LDY  #5          ;Kopfzeile
0382: A9 A0     124          LDA  #$A0
0384: 20 ED FD  125  DISP2   JSR  COUT
0387: 88        126          DEY
0388: D0 FA     127          BNE  DISP2
038A: 98        128  DISP3   TYA
038B: 20 DA FD  129          JSR  PRBYTE
038E: C8        130          INY
038F: C0 20     131          CPY  #$20
0391: D0 F7     132          BNE  DISP3
                133  *
0393: A0 00     134          LDY  #0
0395: 84 CE     135          STY  IND
0397: AD 07 03  136          LDA  PUFFANF
039A: 85 CF     137          STA  IND+1
039C: 18        138          CLC
039D: 69 02     139          ADC  #2
039F: 85 FE     140          STA  PUFFEND
03A1: 20 8E FD  141  DISP4   JSR  CROUT
03A4: A5 CF     142          LDA  IND+1
03A6: 20 DA FD  143          JSR  PRBYTE
03A9: A5 CE     144          LDA  IND
03AB: 20 DA FD  145          JSR  PRBYTE
03AE: A9 A0     146          LDA  #$A0
03B0: 20 ED FD  147          JSR  COUT
03B3: A0 00     148          LDY  #0
03B5: B1 CE     149  DISP5   LDA  (IND),Y     ;Hexzeile
03B7: 20 DA FD  150          JSR  PRBYTE
03BA: C8        151          INY
03BB: C0 20     152          CPY  #$20
03BD: 90 F6     153          BCC  DISP5
03BF: 98        154          TYA
03C0: 18        155          CLC
03C1: 65 CE     156          ADC  IND
03C3: 85 CE     157          STA  IND
03C5: A9 00     158          LDA  #0
03C7: 65 CF     159          ADC  IND+1
03C9: 85 CF     160          STA  IND+1
03CB: C5 FE     161          CMP  PUFFEND
03CD: D0 D2     162          BNE  DISP4
03CF: 60        163          RTS

208 Bytes
```

3.9. DOS-ProDOS-Konvertierungsprogramm

3.9.1. DOSTOPRO.A und DOSTOPRO.O

Dieses Programm, das mit RUN DOSTOPRO.A gestartet wird und seinerseits ein Maschinenprogramm namens DOSTOPRO.O einlädt, dient zur Übertragung von DOS-3.3-Files (nur T-, B- und A-Files) auf ProDOS-Disketten. Die maximale Dateigröße beträgt (einschließlich des TSL-Sektors) 123 DOS-3.3-Sektoren (wie aus DOS-Catalog ersichtlich). Die Slot- und Drive-Nummer der DOS-3.3-Diskette kann mit
BLOAD DOSTOPRO.O
CALL -151
0BBE: 06 (Slot)
0BBF: 02 (Drive)
Ctrl-C
BSAVE DOSTOPRO.O
beliebig gepokt werden, so daß DOSTOPRO auch bei 1-Drive-Besitzern funktioniert. Nach dem Start wird zunächst der DOS-3.3-Catalog angezeigt, wobei vor jedem Dateinamen eine Hexzahl steht. Man muß lediglich die gewünschte Hexzahl und später den neuen, oft notwendigerweise kürzeren ProDOS-Dateinamen eingeben. Es gibt folgende spezielle Fehlernummern:
$01 = DOS-Datei nicht gefunden
$02 = Weder T- noch A- noch B-Dateityp
$03 = Mehr als 123 Sektoren
$04 = „Loch"- oder Leerdatei
Random-„Loch"-Dateien können nicht konvertiert werden. Dies kann übrigens das CONVERT-Programm der Firma Apple auch nicht.

DOSTOPRO.A

```
100  LOMEM: 3840: PRINT  CHR$ (4)"PREFIX": INPUT P$:
     PRINT  CHR$ (4)"PREFIX"P$: PRINT  CHR$
     (4)"BLOAD"P$"DOSTOPRO.O": REM 3840=$0F00
110  PRINT  CHR$ (4)"FRE": GOSUB 180: PRINT "DOS-DISK";:
     GOSUB 190
120  CALL 3000: IF  PEEK (3012) > 0 THEN  END :
     REM *** DOSREAD
130  GOSUB 180: PRINT "PRODOS-DISK";: GOSUB 190: PRINT
     CHR$ (4)"CATALOG"P$: PRINT "DATEINAME:"P$;: INPUT
     "";X$:N$ = P$ + X$:N =  LEN (N$): ON N > 127 OR  LEN
     (X$) > 15 OR X$ = "" GOTO 130
140  POKE 772,N: FOR X = 1 TO N: POKE 772 + X, ASC ( MID$
     (N$,X,1)): NEXT
```

```
150  PRINT  CHR$ (4)"BSAVE"N$",A" PEEK (768) +  PEEK
     (769) * 256",L" PEEK (770) +  PEEK (771) * 256
160  CALL 3003: IF  PEEK (3012) > 0 THEN  END :
     REM *** SETFILEINFO
170  GOSUB 180: PRINT "ERNEUT J/N ";: GET X$: ON X$ = "J"
     GOTO 110: ON X$ <  > "N" GOTO 170: END
180  HOME : INVERSE : PRINT "DOSTOPRO": PRINT "STIEHL85":
     NORMAL : VTAB 4: HTAB 1: RETURN
190  PRINT " EINLEGEN": PRINT : PRINT "W = WEITER ";
200  GET X$: ON X$ <  > "W" GOTO 200: PRINT : RETURN
```

DOSTOPRO.O

```
BSAVE DOSTOPRO.O, A3000, L799
```

```
                    1              ORG  $0BB8       ;3000
                    2   *
                    3   * DOSTOPRO.O (From DOS to ProDOS)
                    4   * ==========
                    5   *
                    6   * Konvertierung von DOS-3.3-Files
                    7   * (T, B, A) in ProDOS-Files.
                    8   * Limitierungen:
                    9   * DOS-3.3-File-Größe maximal
                   10   * 122 Datensektoren. Nicht für
                   11   * Random-"Loch"-Dateien gedacht!
                   12   *
                   13   * von U.Stiehl/19.04.85
                   14   *
                   15   IND       EQU  $00CE
                   16   PUFFER    EQU  $0200
                   17   *
                   18   ADDRESS   EQU  $0300       ;768
                   19   LENGTH    EQU  $0302       ;770
                   20   NAME      EQU  $0304       ;772
                   21   *
                   22   MLI       EQU  $BF00
                   23   HOME      EQU  $FC58
                   24   COUT      EQU  $FDED
                   25   PRBYTE    EQU  $FDDA
                   26   GETLN1    EQU  $FD6F
                   27   *
0BB8: 4C 56 0C     28             JMP  DOSREAD1    ;3000
0BBB: 4C 93 0E     29             JMP  GETINFO     ;3003
                   30   *
                   31   *-------------------------------
                   32   *
                   33   * Allgemeine DOS-3.3-Sektor-
                   34   * Read-Routine.
                   35   * Parameter poken und dann
                   36   * mit JSR SLDR1 aufrufen.
                   37   *
```

```
0BBE: 06          38  SLOT     HEX  06          ;01-07
0BBF: 02          39  DRIVE    HEX  02          ;01-02
0BC0: 00          40  TRACK    HEX  00          ;00-22
0BC1: 00          41  SECTOR   HEX  00          ;00-0F
0BC2: 10          42  BLCKPUF  HEX  10          ;HH
0BC3: 12          43  SECTPUF  HEX  12          ;HH
0BC4: 00          44  ERROR    HEX  00          ;3012
                  45  *
                  46  * Slot/Drive in Unit-Nr. packen
                  47  *
0BC5: AD BE 0B    48  SLDR1    LDA  SLOT
0BC8: AE BF 0B    49          LDX  DRIVE
0BCB: 0A          50          ASL
0BCC: 0A          51          ASL
0BCD: 0A          52          ASL
0BCE: 0A          53          ASL
0BCF: E0 01       54          CPX  #1          ;Drive 1
0BD1: F0 02       55          BEQ  SLDR2
0BD3: 09 80       56          ORA  #%10000000
0BD5: 8D 40 0C    57  SLDR2    STA  RDUNIT
                  58  *
                  59  * Track x 16 = Blocknummer-Basis
                  60  *
0BD8: A9 00       61          LDA  #0
0BDA: 8D 44 0C    62          STA  RDBLOCK+1
0BDD: AD C0 0B    63          LDA  TRACK
0BE0: 0A          64          ASL              ;x 16
0BE1: 0A          65          ASL
0BE2: 0A          66          ASL
0BE3: 8D 43 0C    67          STA  RDBLOCK     ;LL
0BE6: 90 05       68          BCC  OFFSET
0BE8: A9 01       69          LDA  #1
0BEA: 8D 44 0C    70          STA  RDBLOCK+1   ;HH
                  71  *
                  72  * + Spurblock = Blocknummer
                  73  *
0BED: AE C1 0B    74  OFFSET   LDX  SECTOR
0BF0: BD 45 0C    75          LDA  TABELLE,X
0BF3: 48          76          PHA
0BF4: 4A          77          LSR
0BF5: 4A          78          LSR
0BF6: 4A          79          LSR
0BF7: 4A          80          LSR
0BF8: 18          81          CLC
0BF9: 6D 43 0C    82          ADC  RDBLOCK
0BFC: 8D 43 0C    83          STA  RDBLOCK     ;LL
0BFF: 90 03       84          BCC  INITPUF
0C01: EE 44 0C    85          INC  RDBLOCK+1
                  86  *
                  87  * Block- und Sektorpuffer poken
                  88  *
0C04: AD C2 0B    89  INITPUF  LDA  BLCKPUF     ;HH
0C07: 8D 42 0C    90          STA  RDPUFF+1
0C0A: 8D 30 0C    91          STA  MOVER2+2
0C0D: AD C3 0B    92          LDA  SECTPUF     ;HH
```

```
0C10: 8D 33 0C  93              STA  MOVER3+2
                94  *
                95  * Linke oder rechte Blockhälfte?
                96  *
0C13: 68        97              PLA
0C14: 29 0F     98              AND  #$0F
0C16: F0 03     99              BEQ  READER
0C18: EE 30 0C  100             INC  MOVER2+2
                101 *
                102 * Block lesen
                103 *
0C1B: 20 38 0C  104 READER      JSR  READBLCK
0C1E: 90 0C     105             BCC  MOVER1
                106 *
                107 * 00 = Kein Fehler
                108 * 01 = Datei nicht gefunden
                109 * 02 = Weder T noch A noch B
                110 * 03 = Mehr als 122 Sektoren
                111 * 04 = "Loch" oder Leer-Datei
                112 *
0C20: 8D C4 0B  113 ERR0        STA  ERROR
0C23: 20 58 FC  114             JSR  HOME
0C26: AD C4 0B  115             LDA  ERROR
0C29: 4C DA FD  116             JMP  PRBYTE
                117 *
                118 * Blockhälfte -> Sektorpuffer
                119 *
0C2C: A2 00     120 MOVER1      LDX  #0
0C2E: BD 00 10  121 MOVER2      LDA  $1000,X    ;Dummy
0C31: 9D 00 12  122 MOVER3      STA  $1200,X    ;Dummy
0C34: E8        123             INX
0C35: D0 F7     124             BNE  MOVER2
0C37: 60        125             RTS
                126 *
                127 * Read-Block
                128 *
0C38: 20 00 BF  129 READBLCK    JSR  MLI
0C3B: 80        130             HEX  80         ;Read
0C3C: 3F 0C     131             DA   RDNUM1
0C3E: 60        132             RTS
                133 *
0C3F: 03        134 RDNUM1      HEX  03
0C40: 60        135 RDUNIT      HEX  60
0C41: 00 10     136 RDPUFF      HEX  0010       ;LLHH
0C43: 00 00     137 RDBLOCK     HEX  0000       ;LLHH
                138 *
                139 * Sektor-Konvertiertabelle
                140 *
                141 * Linkes Nibble:  Block-Nummer
                142 *                 (0-7)
                143 *
                144 * Rechtes Nibble: Blockhälfte
                145 *                 0 = linke
                146 *                 1 = rechte
                147 *
```

```
0C45: 00          148  TABELLE  HEX  00          ;0
0C46: 70          149           HEX  70          ;1
0C47: 61          150           HEX  61          ;2
0C48: 60          151           HEX  60          ;3
0C49: 51          152           HEX  51          ;4
0C4A: 50          153           HEX  50          ;5
0C4B: 41          154           HEX  41          ;6
0C4C: 40          155           HEX  40          ;7
0C4D: 31          156           HEX  31          ;8
0C4E: 30          157           HEX  30          ;9
0C4F: 21          158           HEX  21          ;A
0C50: 20          159           HEX  20          ;B
0C51: 11          160           HEX  11          ;C
0C52: 10          161           HEX  10          ;D
0C53: 01          162           HEX  01          ;E
0C54: 71          163           HEX  71          ;F
                  164  *
                  165  *-------------------------------
                  166  *
                  167  * Speicherverteilung
                  168  * ----------------
                  169  *
                  170  * DOSTOPRO.A      $0801-$0BB7
                  171  * DOSTOPRO.O      $0BB8-$0EFF
                  172  * LOMEM:          $0F00-$0FFF
                  173  * Blockpuffer:    $1000-$11FF
                  174  * Datenpuffer:    $1200-$8CFF
                  175  * Catalogpuffer:  $1200-$20FF
                  176  * TSL-Puffer:     $1200-$12FF
                  177  * Dateipuffer:    $1300-$8CFF
                  178  * Stringpuffer:   $8D00-$95FF
                  179  *
                  180  BLKPUF   EQU  $1000       ;-$11FF
                  181  DATPUF   EQU  $1200
                  182  CATEND   EQU  $2100
                  183  DATEND   EQU  $8D00
                  184  *
0C55: 60          185  ERR1     RTS              ;Exit
                  186  *
                  187  * Zunächst Block 0 einlesen,
                  188  * um Lesekopf zu justieren.
                  189  * Dreimal probieren und dann
                  190  * bei Fehler aufgeben!
                  191  *
0C56: A9 00       192  DOSREAD1 LDA  #0
0C58: 8D C0 0B    193           STA  TRACK
0C5B: 8D C1 0B    194           STA  SECTOR
0C5E: A9 12       195           LDA  #>DATPUF
0C60: 8D C3 0B    196           STA  SECTPUF
0C63: A9 10       197           LDA  #>BLKPUF
0C65: 8D C2 0B    198           STA  BLCKPUF
0C68: 20 C5 0B    199           JSR  SLDR1       ;1mal
0C6B: 20 C5 0B    200           JSR  SLDR1       ;2mal
0C6E: A9 00       201           LDA  #0
0C70: 8D C4 0B    202           STA  ERROR
```

```
0C73: 20 C5 0B  203           JSR  SLDR1       ;3mal
0C76: AD C4 0B  204           LDA  ERROR
0C79: F0 01     205           BEQ  CATALOG1
0C7B: 60        206           RTS              ;Exit
                207  *
                208  * DOS-3.3-Catalog-Sektoren 0F-01
                209  * einlesen in Puffer $1200-$20FF.
                210  *
0C7C: A9 11     211  CATALOG1 LDA  #17
0C7E: 8D C0 0B  212           STA  TRACK
0C81: A9 0F     213           LDA  #15
0C83: 8D C1 0B  214           STA  SECTOR
0C86: 20 C5 0B  215  CATALOG2 JSR  SLDR1
0C89: AD C4 0B  216           LDA  ERROR
0C8C: D0 C7     217           BNE  ERR1
0C8E: EE C3 0B  218           INC  SECTPUF
0C91: CE C1 0B  219           DEC  SECTOR
0C94: D0 F0     220           BNE  CATALOG2
                221  *
                222  * Flag = 0 = Catalog anzeigen
                223  *
0C96: A9 00     224  CATALOG3 LDA  #0
0C98: 8D F3 0C  225           STA  FLAG
0C9B: 20 F7 0C  226           JSR  DISP1
                227  *
                228  * 2stellige Hex-Nummer eingeben
                229  *
0C9E: A2 00     230           LDX  #0
0CA0: BD EA 0C  231  INPUT1   LDA  INPUT6,X
0CA3: F0 06     232           BEQ  INPUT2
0CA5: 20 ED FD  233           JSR  COUT
0CA8: E8        234           INX
0CA9: D0 F5     235           BNE  INPUT1
                236  *
                237  * Nach Return allein wird erneut
                238  * der Catalog von Diskette ein-
                239  * gelesen.
                240  *
0CAB: 20 6F FD  241  INPUT2   JSR  GETLN1
0CAE: AD 00 02  242           LDA  PUFFER
0CB1: 20 CE 0C  243           JSR  INPUT3
0CB4: B0 A0     244           BCS  DOSREAD1
0CB6: 0A        245           ASL
0CB7: 0A        246           ASL
0CB8: 0A        247           ASL
0CB9: 0A        248           ASL
0CBA: 8D F5 0C  249           STA  NUM2
0CBD: AD 01 02  250           LDA  PUFFER+1
0CC0: 20 CE 0C  251           JSR  INPUT3
0CC3: B0 D1     252           BCS  CATALOG3
0CC5: 6D F5 0C  253           ADC  NUM2
0CC8: 8D F5 0C  254           STA  NUM2
0CCB: 4C 69 0D  255           JMP  LOAD2
```

```
                        256  *
                        257  * 0-9 oder A-F ?
                        258  *
0CCE: C9 B0             259  INPUT3   CMP  #$B0
0CD0: 90 16             260          BCC  INPUT5
0CD2: C9 BA             261          CMP  #$BA
0CD4: B0 04             262          BCS  INPUT4
0CD6: 29 0F             263          AND  #$0F
0CD8: 18                264          CLC
0CD9: 60                265          RTS
0CDA: C9 C1             266  INPUT4   CMP  #$C1
0CDC: 90 0A             267          BCC  INPUT5
0CDE: C9 C7             268          CMP  #$C7
0CE0: B0 06             269          BCS  INPUT5
0CE2: 29 0F             270          AND  #$0F
0CE4: 18                271          CLC
0CE5: 69 09             272          ADC  #$09
0CE7: 60                273          RTS
0CE8: 38                274  INPUT5   SEC              ;Fehler
0CE9: 60                275          RTS
0CEA: 8D                276  INPUT6   HEX  8D
0CEB: CE D5 CD          277          ASC  "NUMMER:"
0CEE: CD C5 D2 BA
0CF2: 00                278          HEX  00
                        279  *
0CF3: 00                280  FLAG     HEX  00          ;0=Cat
0CF4: 00                281  NUM1     HEX  00          ;Zähler
0CF5: 00                282  NUM2     HEX  00          ;GETLN
0CF6: 00                283  YSAVE    HEX  00
                        284  *
                        285  * Zeiger initialisieren
                        286  *
0CF7: A9 12             287  DISP1    LDA  #>DATPUF
0CF9: 85 CF             288          STA  IND+1
0CFB: A9 00             289          LDA  #0
0CFD: 8D F4 0C          290          STA  NUM1
0D00: 20 58 FC          291          JSR  HOME
0D03: A9 0B             292  DISP2    LDA  #$0B        ;1.Eintrag
0D05: 85 CE             293          STA  IND
0D07: A0 00             294  DISP3    LDY  #0
0D09: B1 CE             295          LDA  (IND),Y
0D0B: F0 3C             296          BEQ  DISP7       ;nie belegt
0D0D: C9 FF             297          CMP  #$FF        ;gelöscht
0D0F: F0 38             298          BEQ  DISP7
                        299  *
                        300  * Nummer + Eintrag anzeigen
                        301  *
0D11: AD F3 0C          302          LDA  FLAG
0D14: D0 49             303          BNE  LOAD1
                        304  *
                        305  * Track + Sektor + Filetyp
                        306  * überspringen: Y = Y + 3
                        307  *
0D16: C8                308          INY
```

```
0D17: C8          309           INY
0D18: C8          310           INY
0D19: AD F4 0C    311           LDA  NUM1
0D1C: 20 DA FD    312           JSR  PRBYTE
0D1F: A9 AD       313           LDA  #"-"
0D21: 20 ED FD    314           JSR  COUT
0D24: A2 1E       315           LDX  #30
0D26: B1 CE       316 DISP4     LDA  (IND),Y
0D28: 20 ED FD    317           JSR  COUT
0D2B: C8          318           INY
0D2C: CA          319           DEX
0D2D: D0 F7       320           BNE  DISP4
0D2F: A2 06       321           LDX  #6
0D31: A9 A0       322           LDA  #$A0
0D33: 20 ED FD    323 DISP5     JSR  COUT
0D36: CA          324           DEX
0D37: D0 FA       325           BNE  DISP5
                  326 *
                  327 * Bei 80-Zeichenkarte Return
                  328 * erst nach jedem zweiten Eintrag
                  329 * ausgeben.
                  330 *
0D39: AD 18 C0    331           LDA  $C018      ;80 Z/Z?
0D3C: 10 06       332           BPL  DISP6      ;nein
0D3E: AD F4 0C    333           LDA  NUM1
0D41: 4A          334           LSR
0D42: 90 05       335           BCC  DISP7
0D44: A9 8D       336 DISP6     LDA  #$8D
0D46: 20 ED FD    337           JSR  COUT
                  338 *
                  339 * Nächster Name
                  340 *
0D49: EE F4 0C    341 DISP7     INC  NUM1
0D4C: 18          342           CLC
0D4D: A5 CE       343           LDA  IND
0D4F: 69 23       344           ADC  #$23       ;Offset
0D51: 85 CE       345           STA  IND
0D53: D0 B2       346           BNE  DISP3
0D55: E6 CF       347           INC  IND+1
0D57: A5 CF       348           LDA  IND+1
0D59: C9 21       349           CMP  #>CATEND
0D5B: D0 A6       350           BNE  DISP2
0D5D: 38          351           SEC             ;nein
0D5E: 60          352           RTS
                  353 *
                  354 * Gewünschte Dateinummer mit
                  355 * momentaner Dateinummer identisch?
                  356 *
0D5F: AD F4 0C    357 LOAD1     LDA  NUM1
0D62: CD F5 0C    358           CMP  NUM2
0D65: D0 E2       359           BNE  DISP7
0D67: 18          360           CLC             ;ja
0D68: 60          361           RTS
                  362 *
                  363 * Flag = 1 = Catalog durchsuchen
```

```
                        364  *
0D69: A9 01             365  LOAD2    LDA  #1
0D6B: 8D F3 0C          366           STA  FLAG
0D6E: 20 F7 0C          367           JSR  DISP1
0D71: 90 05             368           BCC  LOAD3
0D73: A9 01             369           LDA  #1          ;fehlt
0D75: 4C 20 0C          370           JMP  ERR0        ;Exit
                        371  *
                        372  * Track + Sektor + Filetyp
                        373  *
0D78: B1 CE             374  LOAD3    LDA  (IND),Y
0D7A: 8D C0 0B          375           STA  TRACK
0D7D: C8                376           INY
0D7E: B1 CE             377           LDA  (IND),Y
0D80: 8D C1 0B          378           STA  SECTOR
                        379  *
                        380  * 00 = T Textfile      04 = TXT
                        381  * 02 = A Applesoftfile FC = BAS
                        382  * 04 = B Binärfile     06 = BIN
                        383  *
0D83: C8                384           INY
0D84: B1 CE             385           LDA  (IND),Y
0D86: 29 7F             386           AND  #$7F
0D88: D0 07             387           BNE  FILETYP1
0D8A: A9 04             388           LDA  #$04
0D8C: 8D D2 0E          389  FILETYP0 STA  FILETYP
0D8F: D0 21             390           BNE  LOAD4
0D91: C9 02             391  FILETYP1 CMP  #2
0D93: D0 04             392           BNE  FILETYP2
0D95: A9 FC             393           LDA  #$FC
0D97: D0 F3             394           BNE  FILETYP0
0D99: C9 04             395  FILETYP2 CMP  #$04
0D9B: D0 04             396           BNE  FILETYP3
0D9D: A9 06             397           LDA  #$06
0D9F: D0 EB             398           BNE  FILETYP0
0DA1: A9 02             399  FILETYP3 LDA  #2
0DA3: D0 03             400           BNE  ERR4
                        401  *
0DA5: 60                402  ERR2     RTS              ;Exit
0DA6: A9 03             403  ERR3     LDA  #3
0DA8: 8D C4 0B          404  ERR4     STA  ERROR
0DAB: 4C 20 0C          405           JMP  ERR0        ;Exit
0DAE: A9 04             406  ERR5     LDA  #4
0DB0: D0 F6             407           BNE  ERR4
                        408  *
                        409  * TSL-Sektor einlesen nach $1200
                        410  *
0DB2: A9 12             411  LOAD4    LDA  #>DATPUF
0DB4: 8D C3 0B          412           STA  SECTPUF
0DB7: 20 C5 0B          413           JSR  SLDR1
0DBA: AD C4 0B          414           LDA  ERROR
0DBD: D0 E6             415           BNE  ERR2
                        416  *
                        417  * Zeiger zur nächsten.TSL leer,
                        418  * d.h. insgesamt nur 1 TSL?
```

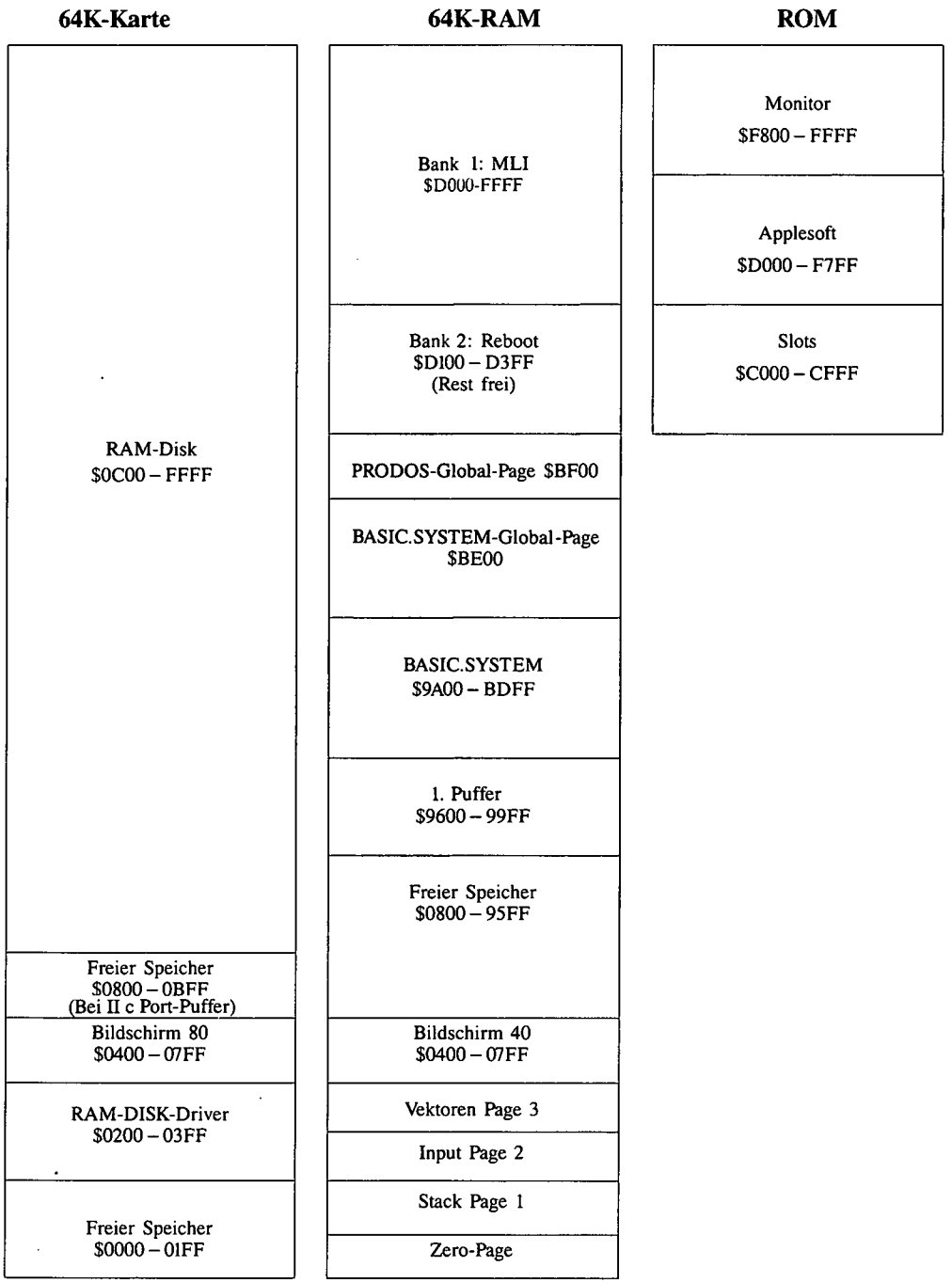
```
                        419  *
0DBF: A0 00             420          LDY  #0
0DC1: 84 CE             421          STY  IND
0DC3: A9 12             422          LDA  #>DATPUF
0DC5: 85 CF             423          STA  IND+1
0DC7: C8                424          INY             ;Y=1
0DC8: B1 CE             425          LDA  (IND),Y
0DCA: D0 DA             426          BNE  ERR3
0DCC: C8                427          INY             ;Y=2
0DCD: B1 CE             428          LDA  (IND),Y
0DCF: D0 D5             429          BNE  ERR3
                        430  *
                        431  * Random-"Loch"-Datei?
                        432  *
0DD1: A0 0C             433          LDY  #$0C       ;1.Track
0DD3: B1 CE             434  LOAD5   LDA  (IND),Y
0DD5: F0 06             435          BEQ  LOAD6
0DD7: C8                436          INY
0DD8: C8                437          INY             ;2.Track
0DD9: D0 F8             438          BNE  LOAD5
0DDB: F0 08             439          BEQ  LOAD7
0DDD: B1 CE             440  LOAD6   LDA  (IND),Y
0DDF: D0 CD             441          BNE  ERR5
0DE1: C8                442          INY
0DE2: C8                443          INY
0DE3: D0 F8             444          BNE  LOAD6
0DE5: A0 0C             445  LOAD7   LDY  #$0C
0DE7: B1 CE             446          LDA  (IND),Y
0DE9: F0 C3             447          BEQ  ERR5       ;leer!
                        448  *
                        449  * Datei einlesen ab $1300
                        450  *
0DEB: 8C F6 0C          451          STY  YSAVE
0DEE: AC F6 0C          452  LOAD8   LDY  YSAVE
0DF1: F0 1A             453          BEQ  TYP0
0DF3: B1 CE             454          LDA  (IND),Y
0DF5: F0 16             455          BEQ  TYP0       ;fertig
0DF7: 8D C0 0B          456          STA  TRACK
0DFA: C8                457          INY
0DFB: B1 CE             458          LDA  (IND),Y
0DFD: 8D C1 0B          459          STA  SECTOR
0E00: C8                460          INY
0E01: 8C F6 0C          461          STY  YSAVE
0E04: EE C3 0B          462          INC  SECTPUF    ;$1300
0E07: 20 C5 0B          463          JSR  SLDR1
0E0A: 90 E2             464          BCC  LOAD8
0E0C: 60                465  ERR6    RTS             ;Exit
                        466  *
                        467  * Anfangsadresse + Länge suchen
                        468  *
0E0D: A2 03             469  TYP0    LDX  #3
0E0F: A9 00             470          LDA  #0
0E11: 9D 00 03          471  TYP1    STA  ADDRESS,X  ;auf 0
0E14: 9D D3 0E          472          STA  AUXTYP,X
0E17: CA                473          DEX
```

```
0E18: 10 F7      474           BPL  TYP1
                 475  *
0E1A: A0 12      476           LDY  #>DATPUF    ;$1200
0E1C: C8         477           INY              ;$1300
0E1D: 8C 01 03   478           STY  ADDRESS+1   ;HH
0E20: 84 CF      479           STY  IND+1
0E22: A0 00      480           LDY  #0
0E24: 84 CE      481           STY  IND
                 482  *
0E26: AD D2 0E   483           LDA  FILETYP
0E29: C9 04      484           CMP  #$04
0E2B: F0 3C      485           BEQ  TXT1
0E2D: C9 06      486           CMP  #$06
0E2F: F0 1B      487           BEQ  BIN1
                 488  *
                 489  * Applesoftfile:
                 490  * LL HH Länge + Programm
                 491  *
0E31: A9 02      492  BAS1     LDA  #$02        ;+2
0E33: 8D 00 03   493           STA  ADDRESS     ;LL
0E36: B1 CE      494           LDA  (IND),Y
0E38: 8D 02 03   495           STA  LENGTH
0E3B: C8         496           INY
0E3C: B1 CE      497           LDA  (IND),Y
0E3E: 8D 03 03   498           STA  LENGTH+1
0E41: A9 01      499           LDA  #$01        ;$0801
0E43: 8D D3 0E   500           STA  AUXTYP
0E46: A9 08      501           LDA  #$08
0E48: 8D D4 0E   502           STA  AUXTYP+1
0E4B: 60         503           RTS              ;Exit
                 504  *
                 505  * Binärfile:
                 506  * LL HH Adresse + LL HH Länge
                 507  * + eigentliche Daten
                 508  *
0E4C: A9 04      509  BIN1     LDA  #$04        ;+4
0E4E: 8D 00 03   510           STA  ADDRESS     ;LL
                 511  *
0E51: B1 CE      512           LDA  (IND),Y     ;Adresse
0E53: 8D D3 0E   513           STA  AUXTYP
0E56: C8         514           INY
0E57: B1 CE      515           LDA  (IND),Y
0E59: 8D D4 0E   516           STA  AUXTYP+1
0E5C: C8         517           INY
0E5D: B1 CE      518           LDA  (IND),Y
0E5F: 8D 02 03   519           STA  LENGTH
0E62: C8         520           INY
0E63: B1 CE      521           LDA  (IND),Y
0E65: 8D 03 03   522           STA  LENGTH+1
0E68: 60         523           RTS              ;Exit
                 524  *
                 525  * Textfile:
                 526  * ASCII-Daten, jedoch Bit 7 off.
                 527  * Länge muß anhand von Hex 00
                 528  * erst ermittelt werden.
```

```
                        529  *
0E69: A9 00             530  TXT1     LDA  #0
0E6B: 8D 02 03          531           STA  LENGTH
0E6E: 8D 03 03          532           STA  LENGTH+1
0E71: B1 CE             533  TXT2     LDA  (IND),Y
0E73: F0 1A             534           BEQ  TXT3
0E75: 29 7F             535           AND  #$7F        ;Bit 7
0E77: 91 CE             536           STA  (IND),Y
0E79: E6 CE             537           INC  IND
0E7B: EE 02 03          538           INC  LENGTH
0E7E: D0 F1             539           BNE  TXT2
0E80: E6 CF             540           INC  IND+1
0E82: EE 03 03          541           INC  LENGTH+1
0E85: A5 CF             542           LDA  IND+1
0E87: C9 8D             543           CMP  #>DATEND
0E89: 90 E6             544           BCC  TXT2
0E8B: A9 04             545           LDA  #4
0E8D: D0 01             546           BNE  ERR7
0E8F: 60                547  TXT3     RTS
                        548  *
0E90: 4C 20 0C          549  ERR7     JMP  ERR0        ;Exit
                        550  *
                        551  *-------------------------------
                        552  *
                        553  * Get/Set File-Info
                        554  *
0E93: A9 0A             555  GETINFO  LDA  #$0A
0E95: 8D C0 0E          556           STA  CNTA
0E98: 20 00 BF          557           JSR  MLI
0E9B: C4                558           HEX  C4          ;Getinfo
0E9C: C0 0E             559           DA   CNTA
0E9E: B0 F0             560           BCS  ERR7
                        561  *
                        562  * Filetyp + Auxtyp poken
                        563  *
0EA0: AD D2 0E          564           LDA  FILETYP
0EA3: 8D C4 0E          565           STA  FILETYPA
0EA6: AD D3 0E          566           LDA  AUXTYP
0EA9: 8D C5 0E          567           STA  AUXTYPEA
0EAC: AD D4 0E          568           LDA  AUXTYP+1
0EAF: 8D C6 0E          569           STA  AUXTYPEA+1
                        570  *
0EB2: A9 07             571  SETINFO  LDA  #$07
0EB4: 8D C0 0E          572           STA  CNTA
0EB7: 20 00 BF          573           JSR  MLI
0EBA: C3                574           HEX  C3          ;Setinfo
0EBB: C0 0E             575           DA   CNTA
0EBD: B0 D1             576           BCS  ERR7
0EBF: 60                577           RTS              ;ENDE!
                        578  *
0EC0: 00                579  CNTA     HEX  00
0EC1: 04 03             580  NAMEA    DA   NAME
0EC3: 00                581  ACCESSA  HEX  00
0EC4: 00                582  FILETYPA HEX  00
```

```
0EC5: 00 00      583  AUXTYPEA HEX  0000
0EC7: 00         584  STORAGEA HEX  00
0EC8: 00 00      585  BLOCKSA  HEX  0000
0ECA: 00 00      586  MDATEA   HEX  0000
0ECC: 00 00      587  MTIMEA   HEX  0000
0ECE: 00 00      588  CDATEA   HEX  0000
0ED0: 00 00      589  CTIMEA   HEX  0000
                 590  *
0ED2: 00         591  FILETYP  HEX  00
0ED3: 00 00      592  AUXTYP   HEX  0000        ;LLHH
0ED5: 00 00      593               HEX  0000

799 Bytes
```


64K-Karte
64K-RAM
ROM
RAM-Disk
$0C00 – FFFF
Freier Speicher
$0800 – 0BFF
(Bei II c Port-Puffer)
Bildschirm 80
$0400 – 07FF
RAM-DISK-Driver
$0200 – 03FF
Freier Speicher
$0000 – 01FF
Bank 1: MLI
$D000-FFFF
Bank 2: Reboot
$D100 – D3FF
(Rest frei)
PRODOS-Global-Page $BF00
BASIC.SYSTEM-Global-Page
$BE00
BASIC.SYSTEM
$9A00 – BDFF
1. Puffer
$9600 – 99FF
Freier Speicher
$0800 – 95FF
Bildschirm 40
$0400 – 07FF
Vektoren Page 3
Input Page 2
Stack Page 1
Zero-Page
Monitor
$F800 – FFFF
Applesoft
$D000 – F7FF
Slots
$C000 – CFFF

MLI-Befehle

READ BLOCK ($80)
Parameter-Count = $03 – 1 Byte
Unit-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes
Block-Number – LL HH – 2 Bytes

WRITE BLOCK ($81)
Parameter-Count = $03 – 1 Byte
Unit-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes
Block-Number – LL HH - 2 Bytes

ALLOCATE INTERRUPT ($40)
Parameter-Count = $02 – 1 Byte
Interrupt-Reference-Number = 1 Byte; gepokt
Interrupt-Code-Pointer – LL HH – 2 Bytes

DEALLOCATE INTERRUPT ($41)
Parameter-Count = $01 – 1 Byte
Interrupt-Reference-Number – 1 Byte

GET TIME ($82)
(hat keine Parameter!)

REBOOT ($65)
Parameter-Count = $04 – 1 Byte
(sonst keine Parameter!)

CREATE ($C0)
Parameter-Count = $07 – 1 Byte
Name-Pointer – LL HH – 2 Bytes
Access – 1 Byte
File-Type – 1 Byte
Auxiliary Type – 2 Bytes
Storage-Type – 1 Byte
Creation-Date – 2 Bytes
Creation-Time – 2 Bytes

DESTROY ($C1)
Parameter-Count = $01 – 1 Byte
Name-Pointer – LL HH – 2 Bytes

RENAME ($C2)
Parameter-Count = $02 – 1 Byte
Name-Pointer zum alten Namen – LL HH – 2 Bytes
Name-Pointer zum neuen Namen – LL HH – 2 Bytes

SET FILE INFO ($C3)
Parameter-Count = $07 – 1 Byte
Name-Pointer – LL HH – 2 Bytes
Access – 1 Byte
File-Type – 1 Byte
Auxiliary Type – 2 Bytes
Null-Field – 3 Bytes
Modification-Date – 2 Bytes
Modification-Time – 2 Bytes

GET FILE INFO ($C4)
Parameter-Count = $0A – 1 Byte
Name-Pointer – LL HH – 2 Bytes
Access – 1 Byte, gepokt
File-Type – 1 Byte; gepokt
Auxiliary Type – 2 Bytes; gepokt
Storage-Type – 1 Byte, gepokt
Blocks used – LL HH – 2 Bytes; gepokt
Modification-Date – 2 Bytes; gepokt
Modification-Time – 2 Bytes; gepokt
Creation-Date – 2 Bytes, gepokt
Creation-Time – 2 Bytes; gepokt

ON LINE ($C5)
Parameter-Count = $02 – 1 Byte
Unit-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes

SET PREFIX ($C6)
Parameter-Count = $01 – 1 Byte
Name-Pointer – LL HH – 2 Bytes

GET PREFIX ($C7)
Parameter-Count = $01 – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes

OPEN ($C8)
Parameter-Count = $03 – 1 Byte
Name-Pointer – LL HH – 2 Bytes
I/O-Buffer-Pointer – LL HH – 2 Bytes
File-Reference-Number – 1 Byte; gepokt

NEWLINE ($C9)
Parameter-Count = $03 – 1 Byte
File-Reference-Number – 1 Byte
Enable-Mask – 1 Byte
Newline-Character – 1 Byte

READ ($CA)
Parameter-Count = $04 – 1 Byte
File-Reference-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes
Request-Count – LL HH – 2 Bytes
Transfer-Count – LL HH – 2 Bytes; gepokt

WRITE ($CB)
Parameter-Count = $04 – 1 Byte
File-Reference-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes
Request-Count – LL HH – 2 Bytes
Transfer-Count – LL HH – 2 Bytes; gepokt

CLOSE ($CC)
Parameter-Count = $01 – 1 Byte
File-Reference-Number – 1 Byte

FLUSH ($CD)
Parameter-Count = $01 – 1 Byte
File-Reference-Number – 1 Byte

SET MARK ($CE)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
Position – LL MM HH – 3 Bytes

GET MARK ($CF)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
Position – LL MM HH – 3 Bytes; gepokt

SET EOF ($D0)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
EOF – LL MM HH – 3 Bytes

GET EOF ($D1)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
EOF – LL MM HH – 3 Bytes; gepokt

SET BUF ($D2)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
I/O-Buffer-Pointer – LL HH – 2 Bytes

GET BUF ($D3)
Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
I/O-Buffer-Pointer – LL HH – 2 Bytes, gepokt

1.5.5. Fehlernummern

Die MLI-Fehlernummern liegen bei ProDOS Version 1.0 im Bereich $00-$5A, wobei $00 = kein Fehler bedeutet.

$00: kein Fehler
$01: falsche MLI-Befehlsnummer
$04: falsche Parameter-Anzahl
$25: Interrupt-Vektor-Tabelle voll (nur 4 Interrupts möglich)
$27: I/O-Fehler (auch ersatzweise für unbekannte Fehler benutzt)
$28: kein Laufwerk angeschlossen
$2B: Diskette schreibgeschützt
$2E: Disketten vertauscht (WRITE, CLOSE usw. können nicht ausgeführt werden)
$40: Dateiname-Syntax stimmt nicht
$42: Dateieintragblock (File Control Block) voll, d.h. mehr als 8 Files
$43: ungültige File-Verweisnummer
$44: Vollständiger Dateiname (Pathname) oder Subdirectory nicht gefunden
$45: Volume-Directory nicht gefunden
$46: Unvollständiger Dateiname (Partial Pathname) nicht gefunden
$47: Doppelter Dateiname (vgl. $57)
$48: Diskette voll oder EOF bei WRITE überschritten
$49: Volume-Directory voll (mehr als 51 Namen)
$4A: Illegaler Dateityp
$4B: Illegaler Speichertyp
$4C: EOF bei READ überschritten
$4D: Positionszeiger größer als EOF
$4E: keine Zugriffserlaubnis (z.B. WRITE bei LOCKED File)
$50: Datei noch/bereits offen (bei erneutem OPEN, ferner bei RENAME und DESTROY)
$51: Anzahl der aktiven Dateieinträge falsch (fehlerhafte Directory-Struktur)
$52: keine ProDOS-Diskette (sondern z.B. DOS 3.3-Diskette)
$53: ungültiger oder unzulässiger Wert in Parameter-Liste
$55: Volume Control Block (VCB) voll, d.h. mehr als 8 Files über mehr als 8 Peripherie-Anschlüsse geöffnet (z.B. 5 Slots mit je 2 Drives)
$56: Datenpuffer oder I/O-Puffer kollidiert mit System Bit Map
$57: Doppelter Volume-Name (vgl. $47)
$5A: Volume Bit Map und Blockgesamtanzahl stimmen nicht überein (fehlerhafte File-Struktur)

BASIC.SYSTEM-Fehlernummern

```
 0 Kein Fehler
 1 Nicht benutzt
 2 RANGE ERROR (Ungültige Parameter)
 3 NO DEVICE CONNECTED (Slot nicht belegt)
 4 WRITE PROTECTED (Diskette schreibgeschützt)
 5 END OF DATA (Dateiende erreicht)
 6 PATH NOT FOUND (Pfadname nicht gefunden)
 7 Nicht benutzt
 8 I/O ERROR (Laufwerkfehler)
 9 DISK FULL (Diskette voll)
10 FILE LOCKED (Datei gelockt)
11 INVALID PARAMETER (Ungültige Parameter)
12 NO BUFFERS AVAILABLE (Alle Puffer belegt)
13 FILE TYPE MISMATCH (Falscher Dateityp)
14 PROGRAM TOO LARGE (Programm zu groß)
15 NOT DIRECT COMMAND (Nicht als Direktbefehl erlaubt)
16 SYNTAX ERROR (Syntaxfehler)
17 DIRECTORY FULL (Volume-Directory voll)
18 FILE NOT OPEN (Datei nicht geöffnet)
19 DUPLICATE FILENAME (Dateiname doppelt)
20 FILE BUSY (Datei bereits offen)
21 FILE(S) STILL OPEN (Datei noch nicht geschlossen)
```

Applesoft-Fehlernummern

```
  0 NEXT WITHOUT FOR (Next ohne For)
 16 SYNTAX ERROR (Syntaxfehler)
 22 RETURN WITHOUT GOSUB (Return ohne Gosub)
 42 OUT OF DATA (Mehr READs als DATA)
 53 ILLEGAL QUANTITY (Illegale Größe)
 69 OVERFLOW (Überlauf)
 77 OUT OF MEMORY (Speicher erschöpft)
 90 UNDEF'D STATEMENT (Nicht-existente Befehlszeile)
107 BAD SUBSCRIPT (Falscher Array-Index)
120 REDIM'D ARRAY (Array erneut dimensioniert)
133 DIVISION BY ZERO (Division durch 0)
163 TYPE MISMATCH (Falscher Variablentyp)
176 STRING TOO LONG (String zu lang)
191 FORMULA TOO COMPLEX (Formel zu komplex)
224 UNDEF'D FUNCTION (Undefinierte Funktion)
254 Falsche Eingabe nach INPUT
255 Programmunterbrechung mit Ctrl-C
```

Fehlerbehandlung

```
 100 ONERR GOTO 1000
 110 REM Ab hier weiteres Programm

 500 POKE 216, 0: REM ONERR abstellen
 510 END: REM Programmende

1000 PRINT "Fehler-Nr.: " PEEK (222)
1010 PRINT "in Programmzeile: " PEEK (218) + PEEK (219) * 256
1020 POKE 512, 104: POKE 513, 168: POKE 514, 104:
     POKE 515, 166: POKE 516, 223: POKE 517, 154:
     POKE 518,  72: POKE 519, 152: POKE 520,  72:
     POKE 521,  96: CALL 512: REM Initialisiert den Stack.
     Danach sind alle For-Next-Schleifen und Gosubs "vergessen".
     Die Variablenwerte bleiben jedoch erhalten.
1030 GOTO xxx: REM Zurück zum Hauptmenü o.ä.
```

BASIC.SYSTEM-Befehle

```
CAT (/n, Ss, Dd, Tt)
CATALOG (/n, Ss, Dd, Tt)
PREFIX (/n, Ss, Dd)
CREATE /n, Tt (, Ss, Dd)
RENAME /alt.n, /neu.n (, Ss, Dd)
DELETE /n (, Ss, Dd)
LOCK /n (, Ss, Dd)
UNLOCK /n (, Ss, Dd)
VERIFY /n (, Ss, Dd)
BYE (ohne Parameter)
NOMON (wird ignoriert!)

PR#s (, Aa) oder
PR# Aa
IN#s oder IN# Aa

- /n (, Ss, Dd)
RUN /n (,  §z, Ss, Dd)
LOAD /n (, Ss, Dd)
SAVE /n (, Ss, Dd)
CHAIN /n (,  §z, Ss, Dd)
STORE /n (, Ss, Dd)
RESTORE /n (, Ss, Dd)
FRE (ohne Parameter)

BSAVE /n, Aa, Ll (, Bb, Tt, Ss, Dd) oder
BSAVE /n, Aa, Ee (, Bb, Tt, Ss, Dd)
BLOAD /n (, Aa, Bb, Ll, Tt, Ss, Dd) oder
BLOAD /n (, Aa, Bb, Ee, Tt, Ss, Dd)
BRUN /n (, Aa, Bb, Ll, Ss, Dd) oder
BRUN /n (, Aa, Bb, Ee, Ss, Dd)
```

```
OPEN /n (, Ll, Tt, Ss, Dd)
READ /n (, Rr, Ff, Bb)
WRITE /n (, Rr, Ff, Bb)
POSITION /n, Ff oder
POSITION /n, Rr
APPEND /n (, Tt, Ll, Ss, Dd)
FLUSH /n
CLOSE /n
EXEC /n (, Ff, Ss, Dd) oder
EXEC /n (, Rr, Ss, Dd)
```

Parameter

```
/n = Name (/Präfix/Dateiname; max. 64 Zeichen)
s  = Slot (0-7)
d  = Drive (1-2)
t  = Typ der Datei (SYS, BAS, BIN, TXT usw.)
a  = Anfangsadresse (theoretisch $0000-$BFFF, praktisch $0300-$99FF)
e  = Endadresse (theoretisch $0000-$BFFF, praktisch $0300-$99FF)
l  = Länge (theoretisch $0000-$FFFF = 0-65535)
b  = Byte-Offset (theoretisch $000000-$FFFFFF = 0-16777215)
f  = Feld-Offset (theoretisch 0-65535)
r  = Recordnummer (theoretisch 0-65535)
z  = Zeilennummer (0-63999)
v  = Volume-Nummer (0-254); wird ignoriert!
```

Dateitypen

```
$01 = BAD = Bad-Blocks-Datei (noch nicht implementiert)
$04 = TXT = Textdatei (ASCII mit Bit 7 off)
$06 = BIN = Binärdatei (Maschinenprogramm, Hiresbild usw.)
$0F = DIR = Subdirectory (Unterinhaltsverzeichnis)
$19 = ADB = Appleworks-Datenbank (früher SOS)
$1A = AWP = Appleworks-Textdatei (früher SOS)
$1B = ASF = Appleworks-Tabellendatei (früher SOS)
$EF = PAS = Pascal-Programm (nicht implementiert)
$F0 = CMD = Befehlsdatei (noch nicht implementiert)
$F1 - $F8 = selbstdefinierte Dateien
$FA = INT = Integer-Basic-Programm (nicht implementiert)
$FB = IVR = Integer-Variablendatei (nicht implementiert)
$FC = BAS = Applesoftprogramm
$FD = VAR = Applesoft-Variablendatei
$FE = REL = Relokatives Maschinenprogramm (EDASM)
$FF = SYS = Systemdatei

(Andere Dateinummern entweder Apple-III-SOS
oder noch nicht für ProDOS definiert.)
```

ASCII-Tabelle

NUL	@ §	00	00000000	000	@ §	40	01000000	064	@ §	80	10000000	128	@ §	C0	11000000	192
	A	01	00000001	001	A	41	01000001	065	A	81	10000001	129	A	C1	11000001	193
	B	02	00000010	002	B	42	01000010	066	B	82	10000010	130	B	C2	11000010	194
	C	03	00000011	003	C	43	01000011	067	C	83	10000011	131	C	C3	11000011	195
EOT	D	04	00000100	004	D	44	01000100	068	D	84	10000100	132	D	C4	11000100	196
	E	05	00000101	005	E	45	01000101	069	E	85	10000101	133	E	C5	11000101	197
	F	06	00000110	006	F	46	01000110	070	F	86	10000110	134	F	C6	11000110	198
BELL	G	07	00000111	007	G	47	01000111	071	G	87	10000111	135	G	C7	11000111	199
←	H	08	00001000	008	H	48	01001000	072	H	88	10001000	136	H	C8	11001000	200
TAB	I	09	00001001	009	I	49	01001001	073	I	89	10001001	137	I	C9	11001001	201
↓	J	0A	00001010	010	J	4A	01001010	074	J	8A	10001010	138	J	CA	11001010	202
↑	K	0B	00001011	011	K	4B	01001011	075	K	8B	10001011	139	K	CB	11001011	203
FF	L	0C	00001100	012	L	4C	01001100	076	L	8C	10001100	140	L	CC	11001100	204
RTN	M	0D	00001101	013	M	4D	01001101	077	M	8D	10001101	141	M	CD	11001101	205
SO	N	0E	00001110	014	N	4E	01001110	078	N	8E	10001110	142	N	CE	11001110	206
SI	O	0F	00001111	015	O	4F	01001111	079	O	8F	10001111	143	O	CF	11001111	207
	P	10	00010000	016	P	50	01010000	080	P	90	10010000	144	P	D0	11010000	208
XON	Q	11	00010001	017	Q	51	01010001	081	Q	91	10010001	145	Q	D1	11010001	209
	R	12	00010010	018	R	52	01010010	082	R	92	10010010	146	R	D2	11010010	210
XOFF	S	13	00010011	019	S	53	01010011	083	S	93	10010011	147	S	D3	11010011	211
	T	14	00010100	020	T	54	01010100	084	T	94	10010100	148	T	D4	11010100	212
→	U	15	00010101	021	U	55	01010101	085	U	95	10010101	149	U	D5	11010101	213
	V	16	00010110	022	V	56	01010110	086	V	96	10010110	150	V	D6	11010110	214
	W	17	00010111	023	W	57	01010111	087	W	97	10010111	151	W	D7	11010111	215
	X	18	00011000	024	X	58	01011000	088	X	98	10011000	152	X	D8	11011000	216
	Y	19	00011001	025	Y	59	01011001	089	Y	99	10011001	153	Y	D9	11011001	217
	Z	1A	00011010	026	Z	5A	01011010	090	Z	9A	10011010	154	Z	DA	11011010	218
ESC	[Ä	1B	00011011	027	[Ä	5B	01011011	091	[Ä	9B	10011011	155	[Ä	DB	11011011	219
	\ Ö	1C	00011100	028	\ Ö	5C	01011100	092	\ Ö	9C	10011100	156	\ Ö	DC	11011100	220
	] Ü	1D	00011101	029	] Ü	5D	01011101	093	] Ü	9D	10011101	157	] Ü	DD	11011101	221
	↑	1E	00011110	030	↑	5E	01011110	094	↑	9E	10011110	158	↑	DE	11011110	222
	_	1F	00011111	031	_	5F	01011111	095	_	9F	10011111	159	_	DF	11011111	223
		20	00100000	032	'	60	01100000	096		A0	10100000	160	'	E0	11100000	224
	!	21	00100001	033	a	61	01100001	097	!	A1	10100001	161	a	E1	11100001	225
	"	22	00100010	034	b	62	01100010	098	"	A2	10100010	162	b	E2	11100010	226
	#	23	00100011	035	c	63	01100011	099	#	A3	10100011	163	c	E3	11100011	227
	$	24	00100100	036	d	64	01100100	100	$	A4	10100100	164	d	E4	11100100	228
	%	25	00100101	037	e	65	01100101	101	%	A5	10100101	165	e	E5	11100101	229
	&	26	00100110	038	f	66	01100110	102	&	A6	10100110	166	f	E6	11100110	230
	'	27	00100111	039	g	67	01100111	103	'	A7	10100111	167	g	E7	11100111	231
	(	28	00101000	040	h	68	01101000	104	(	A8	10101000	168	h	E8	11101000	232
	)	29	00101001	041	i	69	01101001	105	)	A9	10101001	169	i	E9	11101001	233
	*	2A	00101010	042	j	6A	01101010	106	*	AA	10101010	170	j	EA	11101010	234
	+	2B	00101011	043	k	6B	01101011	107	+	AB	10101011	171	k	EB	11101011	235
	,	2C	00101100	044	l	6C	01101100	108	,	AC	10101100	172	l	EC	11101100	236
	–	2D	00101101	045	m	6D	01101101	109	–	AD	10101101	173	m	ED	11101101	237
	.	2E	00101110	046	n	6E	01101110	110	.	AE	10101110	174	n	EE	11101110	238
	/	2F	00101111	047	o	6F	01101111	111	/	AF	10101111	175	o	EF	11101111	239
	0	30	00110000	048	p	70	01110000	112	0	B0	10110000	176	p	F0	11110000	240
	1	31	00110001	049	q	71	01110001	113	1	B1	10110001	177	q	F1	11110001	241
	2	32	00110010	050	r	72	01110010	114	2	B2	10110010	178	r	F2	11110010	242
	3	33	00110011	051	s	73	01110011	115	3	B3	10110011	179	s	F3	11110011	243
	4	34	00110100	052	t	74	01110100	116	4	B4	10110100	180	t	F4	11110100	244
	5	35	00110101	053	u	75	01110101	117	5	B5	10110101	181	u	F5	11110101	245
	6	36	00110110	054	v	76	01110110	118	6	B6	10110110	182	v	F6	11110110	246
	7	37	00110111	055	w	77	01110111	119	7	B7	10110111	183	w	F7	11110111	247
	8	38	00111000	056	x	78	01111000	120	8	B8	10111000	184	x	F8	11111000	248
	9	39	00111001	057	y	79	01111001	121	9	B9	10111001	185	y	F9	11111001	249
	:	3A	00111010	058	z	7A	01111010	122	:	BA	10111010	186	z	FA	11111010	250
	;	3B	00111011	059	{ ä	7B	01111011	123	;	BB	10111011	187	{ ä	FB	11111011	251
	<	3C	00111100	060	| ö	7C	01111100	124	<	BC	10111100	188	| ö	FC	11111100	252
	=	3D	00111101	061	} ü	7D	01111101	125	=	BD	10111101	189	} ü	FD	11111101	253
	>	3E	00111110	062	~ ß	7E	01111110	126	>	BE	10111110	190	~ ß	FE	11111110	254
	?	3F	00111111	063	DEL	7F	01111111	127	?	BF	10111111	191	DEL	FF	11111111	255

6502 / 65C02-Tabelle

Function	Mnemon.	Akku		Immedi.		Zero-P.		Zero, X		Zero, Y		Absol.		Abs, X		Abs, Y		(Ind, X)		(Ind), Y		Implizit		Relativ		(Ind)		Status
		Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	Op	Cy	N Z C I D V
Load	LDA			A9	2	A5	3	B5	4			AD	4	BD	4*	B9	4*	A1	6	B1	5*					*B2*	*5*	N Z - - - -
	LDX			A2	2	A6	3			B6	4	AE	4			BE	4*											N Z - - - -
	LDY			A0	2	A4	3	B4	4			AC	4	BC	4*													N Z - - - -
Store	STA					85	3	95	4			8D	4	9D	5	99	5	81	6	91	6					*92*	*5*	- - - - - -
	STX					86	3			96	4	8E	4															- - - - - -
	STY					84	3	94	4			8C	4															- - - - - -
Transfer	TAX																					AA	2					N Z - - - -
	TXA																					8A	2					N Z - - - -
	TAY																					A8	2					N Z - - - -
	TYA																					98	2					N Z - - - -
Stack Ptr	TSX																					BA	2					N Z - - - -
	TXS																					9A	2					- - - - - -
Stack	PHA																					48	3					- - - - - -
	PLA																					68	4					N Z - - - -
Status R.	PHP																					08	3					- - - - - -
	PLP																					28	4					N Z C I D V
Flags	CLC																					18	2					- - 0 - - -
	SEC																					38	2					- - 1 - - -
	CLI																					58	2					- - - 0 - -
	SEI																					78	2					- - - 1 - -
	CLD																					D8	2					- - - - 0 -
	SED																					F8	2					- - - - 1 -
	CLV																					B8	2					- - - - - 0
Jump	JMP											4C	3					*7C*	*6'*							*6C*	*5/6'*	- - - - - -
	JSR											20	6															- - - - - -
Return	RTS																					60	6					- - - - - -
	RTI																					40	6					N Z C I D V
Compare	CMP			C9	2	C5	3	D5	4			CD	4	DD	4*	D9	4*	C1	6	D1	5*					*D2*	*5*	N Z C - - -
	CPX			E0	2	E4	3					EC	4															N Z C - - -
	CPY			C0	2	C4	3					CC	4															N Z C - - -
	BIT			*89*	2	24	3	*34*	*4*			2C	4	*3C*	*4*													- Z - - - -
Branch	BMI																							30	2+			- - - - - -
N = 0	BPL																							10	2+			- - - - - -
	BEQ																							F0	2+			- - - - - -
Z = 0	BNE																							D0	2+			- - - - - -
	BCS																							B0	2+			- - - - - -
C = 0	BCC																							90	2+			- - - - - -
	BVS																							70	2+			- - - - - -
V = 0	BVC																							50	2+			- - - - - -
Increment	INC	*1A*	*2*					E6	5	F6	6			EE	6	FE	*7/6*											N Z - - - -
	INX																					E8	2					N Z - - - -
	INY																					C8	2					N Z - - - -
Decrement	DEC	*3A*	*2*					C6	5	D6	6			CE	6	DE	*7/6*											N Z - - - -
	DEX																					CA	2					N Z - - - -
	DEY																					88	2					N Z - - - -
Add /Subt.	ADC			69	2	65	3	75	4			6D	4	7D	4*	79	4*	61	6	71	5*					*72*	*5*	N Z C - - V
	SBC			E9	2	E5	3	F5	4			ED	4	FD	4*	F9	4*	E1	6	F1	5*					*F2*	*5*	N Z C - - V
Boolean	AND			29	2	25	3	35	4			2D	4	3D	4*	39	4*	21	6	31	5*					*32*	*5*	N Z - - - -
	ORA			09	2	05	3	15	4			0D	4	1D	4*	19	4*	01	6	11	5*					*12*	*5*	N Z - - - -
	EOR			49	2	45	3	55	4			4D	4	5D	4*	59	4*	41	6	51	5*					*52*	*5*	N Z - - - -
Shift	ASL	0A	2			06	5	16	6			0E	6	1E	*7/6*													N Z C - - -
	LSR	4A	2			46	5	56	6			4E	6	5E	*7/6*													0 Z C - - -
Rotate	ROL	2A	2			26	5	36	6			2E	6	3E	*7/6*													N Z C - - -
	ROR	6A	2			66	5	76	6			6E	6	7E	*7/6*													N Z C - - -
Sonstiges	.NOP																					EA	2					- - - - - -
	BRK																					00	7					- - - 1 - -
65C02	*BRA*																							*80*	*3+*			- - - - - -
	PHX																					*DA*	*3*					- - - - - -
	PHY																					*5A*	*3*					- - - - - -
	PLX																					*FA*	*4*					N Z - - - -
	PLY																					*7A*	*4*					N Z - - - -
	STZ					*64*	*3*	*74*	*4*			*9C*	*4*	*9E*	*5*													- - - - - -
	TRB					*14*	*5*					*1C*	*6*															- Z - - - -
	TSB					*04*	*5*					*0C*	*6*															- Z - - - -
Byteanzahl		1		2		2		2		2		3		3		3		2		2		1		2		2		

* = 1 Takt mehr bei Seitenübergang.
\+ = 2 Takte bei Nicht-Verzweigung, 3 Takte bei Verzweigung, 4 Takte bei Verzweigung mit Seitenübergang.
' = JMP ($HHLL) und JMP ($HHLL, X) je 3 Bytes; passen nicht in Systematik
kursiv = 65C02-Änderungen

Applesoft-Tabelle

1. Allgemeines

Arbeitsspeicher: $0800-$95FF, nach DOS-Mover $0800-$BEFF; (1) Programm, (2) LOMEM, (3) einfache Variablen, (4) Feld-Variablen, (5) String-Pool, (6) HIMEM.

Programmzeile: Länge: 239 Bytes; Bereich 0-63999.

Variablenname: (1) Buchstabe, (2) Buchstabe / Ziffer, (3) Zusatz **%** oder $.

Real-Variable: A oder AB oder A1, 9stellige Mantisse, Bereich +/−1 · 10↑38, max. 15 Stellen −1.23456789E+12, interne Binärkodierung, 5 Bytes.

Integer-Variable: A% oder AB% oder A1%, Bereich +/−32767 (−32768 = $8000 illegal!), max. 6 Stellen −12345, interne Hexadezimalkodierung, 2 Bytes.

String-Variable: A$ oder AB$ oder A1$, Länge 0-255 Zeichen (Strings > 239 nur durch Verkettung A$ + B$ erreichbar), Strings stets ab HIMEM abwärts abgelegt.

Feld-Variable: Array oder dimensionierte Variable (Real, Integer, String) mit Index in Klammern; theoretisch 88 Dimensionen; AB (30, 20) 1. Dimension 0-30 und 2. Dimension 0-20.

Algebraische Operatoren: Zuweisung LET X = Y oder X = Y; Negation X = −Y; Plus A = B + C; Minus A = B − C; Mal X = Y · Z; Geteilt X = Y / Z; Potenz Y = X ↑ Z; Verkettung A$ = B$ + C$.

Relationale Operatoren: =, <>, <, >, <=, >=.

Boolesche Operatoren: NOT, AND, OR; IF NOT (A = B) AND (C = D) OR (E < F) THEN GOTO 1000. W = (A = B); W wird 1, wenn ja, und 0, wenn nein.

2. System-Befehle

NEW: Löscht Programm- und Variablenzeiger (nicht Programm selbst, so daß es rekonstruiert werden könnte).

CLEAR: Löscht Variablenzeiger.

HIMEM/LOMEM: HIMEM: definiert Speicherobergrenze; HIMEM: 38400: CLEAR (CLEAR normalisiert Zeiger und entfernt Strings unterhalb von HIMEM); LOMEM: definiert Variablenuntergrenze; CLEAR: LOMEM: 24567 (legt Variablen nach HGR Seite 2). HIMEM mit FP normalisieren (nicht bei ProDOS!).

DIM: dimensioniert Feldvariable; DIM AB$ (100).

FRE: F = FRE (0); Garbage-Collection ausführen und freien String-Speicher ermitteln; IF PEEK (112) − PEEK (110) < 4 THEN F = FRE (0).

RUN: RUN PROGRAMM (von Diskette); RUN (allein, wenn Programm bereits im Speicher).

END/STOP/CONT: END beendet Programmlauf; bei STOP wird zusätzlich Zeilennummer angezeigt; CONT bewirkt Fortsetzung des Programms nach END/STOP.

TRACE/NOTRACE: Zeilennummern während des laufenden Programms anzeigen / nicht anzeigen.

LOAD/SAVE: LOAD PROGRAMM oder SAVE PROGRAMM (von/auf Diskette übertragen); LOAD oder SAVE (ohne Name von/auf Kassette).

STORE/RECALL: STORE AB oder RESTORE AB (dimensionierte Variable AB auf/von Kassette übertragen); Unter ProDOS überträgt STORE VARIABLEN oder RESTORE VARIABLEN alle Variablen auf/von Diskette.

SLOAD: Shape-Tabelle von Kassette laden.

LIST: Programm listen; LIST 100, 200 listet nur diesen Bereich.

DEL: Programmbereich löschen; DEL 100, 200 löscht diesen Bereich; einzelne Zeile mit 100 Return, gesamtes Programm mit NEW löschen.

REM: Kommentar nach REM wird vom Interpreter ignoriert. Programme ohne REMs laufen schneller.

PEEK/POKE: W = PEEK (S) weist W den Wert der Speicherstelle S (0-65535) zu; POKE S, W weist Speicherstelle S den Wert W (0-255) zu.

WAIT: Wartebefehl (vermeiden!).

CALL: CALL S ruft Maschinenprogramm ab Speicherstelle S auf.

&: bewirkt Sprung nach $03F5, wo ein JMP zum eigenen Maschinenprogramm stehen muß.

USR: W2 = USR (W1) weist dem FAC $009D-$00A3 (Real-Zahl-Zwischenspeicher) den Wert W1 zu, springt dann nach $000A, wo ein JMP zum eigenen Maschinenprogramm stehen muß, und überträgt nach RTS den (jetzigen) FAC-Wert nach W2.

3. Verzweigungen

IF/THEN: bedingte Verzweigung; IF X = Y THEN Z oder IF X = Y GOTO Z (Z = Zeilennummer 0-63999); IF X = Y THEN A = 1.

GOTO: GOTO Z; unbedingte Verzweigung zur Zeile Z.

ON GOTO: ON X GOTO Z1, Z2 usw.; wenn X = 1 dann GOTO Z1 usw.

ONERR GOTO: ONERR GOTO 10000; für Fehlerbehandlung; sollte am Programmanfang definiert werden.

RESUME: vermeiden!

GOSUB/RETURN/POP: GOSUB Z ... RETURN; Unterprogrammaufruf. POP beseitigt letztes RETURN vom Stapel; sparsam anwenden!

ON GOSUB: ON X GOSUB Z1, Z2 ...

FOR/STEP/NEXT: Schleife: FOR X = 1 TO 1000 : FOR Y = 1 TO 100 STEP 10: ... : NEXT Y: NEXT X (oder NEXT Y, X oder NEXT: NEXT)

4. Eingabe/Ausgabe

HOME: löscht Bildschirm; bei 80 Z/Z PRINT CHR$ (12).

VTAB/HTAB: VTAB V (1-24); Vertikaltabulator; HTAB H (1-40); Horizontaltabulator; bei II+ 80 Z/Z POKE 36, H − 1, bei IIe/IIc POKE 1403, H − 1 (wobei H = 1-80).

TAB/SPC: PRINT TAB (H), wobei H = 1-40; PRINT SPC (S) „AB HIER"; S = 1-255 Leerzeichen.

POS: H = POS (0); Horizontalposition 0-39.

NORMAL/INVERSE/FLASH: Zeichenmodifikation; FLASH nicht bei 80 Z/Z.

SPEED: Ausgabegeschwindigkeit; SPEED = S, wobei S = 1-255; bei Grafik unwirksam!

PRINT: PRINT „AAA"; (Semikolon unterdrückt Return); PRINT „AAA", (Komma tabuliert; nur 40 Z/Z.

INPUT: INPUT A$ oder INPUT „HINWEIS"; A$; Eingabe max. 239 Zeichen; bei Strings weder Komma noch Doppelpunkt.

GET: GET A$; Tastaturabfrage; übernimmt in 40 Z/Z alle ASCII-Codes; bei IIe/IIc 80 Z/Z nicht ESC, nicht Rechtspfeil usw.; direkte Tastaturabfrage ohne Cursor mit T = PEEK (49152): IF T > 127 THEN POKE 49168, 0.

DATA/READ/RESTORE: Konstantenliste; DATA 10, AAA; mit READ A, B$ den Variablen zuweisen; RESTORE setzt DATA-Zeiger zurück.

IN#S/PR#S: Eingabe/Ausgabe über Slot S (1-7) aktivieren; PR#6 bootet; PR#3 bewirkt 80 Z/Z usw.

5. Funktionen

SIN: X = SIN (100); analog COS, TAN, ATN, SQR, EXP, LOG

INT/ABS: X = INT (123.456) ergibt X = 123 (Ganzzahl); X = ABS (−123) ergibt X = 123 (absoluter Betrag).

RND: Random- oder Zufallszahl; R = INT (D · RND (1)) + U ergibt Random R im Bereich Untergrenze U bis Obergrenze = Untergrenze + Differenz D.

SGN: Signum; (−X) = −1, (+X) = +1; X = SGN (X) · INT (100 · ABS (X) + .5) / 100 rundet X auf 2 Stellen nach Komma.

DEF: DEF FN L (V) = LOG (V) / LOG (10) definiert Funktion für dekadischen Logarithmus; später mit LL = FN L (V) aufrufen, wobei V variabel.

6. Grafik

TEXT: zurück in Textmodus (von Lores oder HGR).

GR: Lores-Grafik mit 4 Textzeilen; danach mit POKE 49234, 0 Vollgrafik.

COLOR: COLOR = C (= 0-15); Lores-Farbe.

PLOT: PLOT X, Y; Kästchen plotten (X-Achse 0-39, Y-Achse 0-39 oder 0-47).

SCRN: C = SCRN (X, Y) ergibt Farbe des Kästchens.

HLIN: HLIN X1, X2 AT Y; horizontaler Balken.

VLIN: VLIN Y1, Y2 AT X; vertikaler Balken.

HGR: Hires-Grafik Seite 1 ($2000-$3FFF) mit 4 Textzeilen; danach mit POKE 49234, 0 Vollgrafik.

HGR2: Hires-Grafik Seite 2 ohne Textzeilen ($4000-$5FFF).

HCOLOR: HCOLOR = C (= 0-7); Hires-Farbe.

HPLOT: HPLOT X, Y Punkt plotten (X-Achse 0-279, Y-Achse 0-159 oder 0-191); HPLOT X1, Y1 TO X2, Y2 usw.

DRAW/XDRAW: DRAW S AT X, Y; Shape S (0-255). Vorher Shape-Tabelle laden und Adresse poken (POKE 232, L: POKE 234, H)! XDRAW plottet mit invertierter Farbe (meist löschen).

ROT: ROT = R (0-255); Shape-Rotation; ROT = 16 entspricht 90 Grad.

SCALE: SCALE = S (1-255); Shape-Maßstab (Größe).

PDL: Paddle; W = PDL (P), wobei W = 1-255 und P = 0-3.

7. Strings

LEN: L = LEN (A$); Länge L (0-255).

STR$: S$ = STR$ (Z); verwandelt Zahl Z in String S$.

VAL: Z = VAL (S$); verwandelt S$ in Z; bei Nicht-Zahl 0.

CHR$: S$ = CHR$ (A); ASCII-Code A (1-255; 0 vermeiden) dem String S$ zuweisen.

ASC: A = ASC (S$); 1. Zeichen von S$ (darf kein Null-String sein!) als ASCII-Code A zuweisen.

LEFT$: PRINT LEFT$ („AAAMMBBB", 3); zeigt „AAA" an.

RIGHT$: PRINT RIGHT$ („AAAMMBBB", 3); zeigt „BBB" an.

MID$: PRINT MID$ („AAAMMBBB", 4, 2); zeigt „MM" an, d. h. 2 Zeichen ab 4. Position.

Begleitdiskette zu ProDOS für Aufsteiger, Band 2

```
Volume-Directory (Utilities): /STIEHL/

NAME            TYPE  BLOCKS        ENDFILE SUBTYPE

AUFSTEIGER.BD2  BAS        1            321
DIR             DIR        2           1024
EINTRAG.SUCHER  BAS        1            333
EINTRAG.ANALYSE BAS        6           2145
CAT.ARRAY       BAS        1            385
CAT.SAVER       BAS        1            345
FILE.READ.ASC   BAS        3            802
ASC             BIN        1            104 A=$0300
T.ASC           TXT        3            659 R=     0
FILE.READ.HEX   BAS        3            859
HEX             BIN        1            147 A=$0300
T.HEX           TXT        4           1253 R=     0
PRODOS.FID.A    BAS        6           2430
PRODOS.FID.O    BIN        3            582 A=$1380
T.PRODOS.FID.O  TXT       11           4960 R=     0
PROFID.DEMO     BAS        1            319
PROFID          BIN        4           1387 A=$0C80
T.PROFID        TXT       28          13489 R=     0
PRODOS.COPY.A   BAS        3            709
PRODOS.COPY.O   BIN        5           1631 A=$0BB8
T.PRODOS.COPY.O TXT       10           4359 R=     0
QUICKCOPYPUFFER BAS        1            360
QUICKCOPY       BIN        5           1881 A=$08AE
T.QUICKCOPY     TXT       14           6457 R=     0
FORMAT.A        BAS        3            758
FORMAT.O        BIN        4           1470 A=$79FA
T.FORMAT.O      TXT        7           2992 R=     0
DISKCOMPARER    BIN        1            464 A=$0BB8
T.DISKCOMPARER  TXT        8           3559 R=     0
BAD.BLOCKS      BIN        1            184 A=$0300
T.BAD.BLOCKS    TXT        4           1269 R=     0
BLOCKEDIT       BIN        1            208 A=$0300
T.BLOCKEDIT     TXT        6           2190 R=     0
DOSTOPRO.A      BAS        3            570
DOSTOPRO.O      BIN        3            799 A=$0BB8
T.DOSTOPRO.O    TXT       17           7749 R=     0
DATA.MAKER      BAS        4           1104
LIST.MAKER      TXT        1            427 R=     0
TXTTAB          BAS        1            220
INALL.LOADER    BAS        1            280
ZAHLEN.ARRAY    BAS        3            935
MON.COMMANDS    BIN        1            366 A=$20B6
T.MON.COMMANDS  TXT        9           3783 R=     0
FP.COMMAND      BIN        1            160 A=$2000
T.FP.COMMAND    TXT        9           3618 R=     0
BITMAP          BIN        1             93 A=$0300
T.BITMAP        TXT        3            861 R=     0
U.STIEHL.1985   BAS        1            321
```

```
Subdirectory (Demos): /STIEHL/DIR/

 NAME            TYPE  BLOCKS        ENDFILE SUBTYPE

 HAUPTPROGRAMM   BAS        1            229
 UNTERPROGRAMM   BAS        1            166
 HAUPTVARIABLEN  BAS        1            217
 UNTERVARIABLEN  BAS        1             87
 TEXTDEMO        BAS        4           1448
 ONERR.DEMO      BAS        1            276
 FRE.HGR1        BAS        1            164
 FRE.HGR2        BAS        1            164
 OPEN.TEST1      BAS        1            456
 OPEN.TEST2      BAS        1            499
 OPEN.TEST3      BAS        1            440
 INPUT.LOADER    BAS        1            132
 INALL.LOADER    BAS        1            272
 STRING.SAVER.1  BAS        1            255
 STRING.SAVER.2  BAS        1            268
 STRING.SAVER.3  BAS        1            242
 ARRAY.ALT       BAS        3            559
 ARRAY.NEU       BAS        3            801
 MON.DEMO        BAS        3            955
 PBITS           BAS        3            908
 FIELD.BUG       BAS        1            419
 ARRAY1          TXT       13           6000 R=     0
 ARRAY2          TXT       13           6000 R=     0
 XXX             TXT        3            956 R=     0

BLOCKS FREE: 1  BLOCKS USED: 279  TOTAL BLOCKS: 280
```

Register

Anfangsadresse 40, 45
APPEND 53
Array 87
Arrays 41
AUXTYPE 30
Bad-Block-Routine 177
BAS 26
BHIMEM 78
Bibliographie 31
BIENTRY 65
BIN 26
BLOAD 44
Blockeditor 179
Blocks 17
Booten 14
BRUN 44
BSAVE 44
BYE 16, 48
CAT 10, 27
CATALOG 11, 19, 27
CHAIN 39
CHRLAST 73
CLOSE 53
CONVERT 183
COPY.A 147
COUT 59, 71
CREATE 27
Ctrl-D 21
DATA.MAKER 50
Dateieintrag 107
Dateitypen 201
Default-Wert 34
DEFDRV 72
DEFSLT 72
DELETE 36
DEVNUM 72
Dez-Hex-Konverter 76
DIR 26
Directory 28, 109
Direkter Befehl 19
Disk-Driver 16
Diskettenvergleich 171
Doppelpunkt 84
DOS 3.3 183
DOSCMD 65
Endadresse 40, 45
ERRCODE 69
ERROUT 68
EXEC 49
Externer Befehl 66
EXTRNCMD 65
FBITS 74
Fehlernummer 198
Feld 54
FID 116
FILER 116
FLUSH 53
FORMAT 165
FP 98
FRE 39, 79
FREEBUFR 78
FVALS 76
Garbage-Collection 43
GETBUFR 78, 82
GETLN 60
GOSYSTEM 77
Hello-Programm 11
Hex-Dump 113
HGR 79
Hierarchische Dateistruktur 31
HIMEM 18, 81
IN# 52
INALL 85
Indirekter Befehl 20
Input 84
INVECT 69
I/O-Puffer 55, 81
Kaltstart 16
Komma 84
Konvertierungsprogramm 183
LIST.MAKER 50
LOAD 39
Loch-Datei 62
LOCK 36
LOMEM 79, 81
MLI 197
MLI-Aufruf 77
MON 91
MONSTER.MAKER 47
Name 31
NOMON 23
OPEN 53
Open-Datei 82
OUTVECT 69
Parameter 23, 201
PBITS 67, 74, 92
Pfadname 12
POSITION 53
Positionszeiger 56
PR# 52
Präfix 12, 35
PREFIX 27
PRINTERR 68
Random-Textfile 57
RDKEY 60, 71
READ 53
Record 57
RENAME 36
RESTORE 39
RUN 39
RUN-Modus 20
SAVE 39
Sequentieller Textfile 54
SOS 25

STARTUP 11, 14
Sternchen 37
STORE 39
Strichbefehl 48
Subdirectory 30, 34
SUBTYPE 30
SYS 25
System Bit Map 82
Textfile 53
Trace 64
TSYS 46
TXT 26, 53
TXTTAB 80, 98
UNLOCK 36
Urlader 14
VAR 26
VDOSIO 71
VECTIN 70
VECTOUT 70
VERIFY 36
Volume-Directory 30, 34
VPATH 77
VSYSIO 71
Warmstart 16
WRITE 53
XCNUM 74
XLEN 73
XTRNADR 73
Zusatzinfo 30

Hüthig

Jürgen Kehrel

Apple-Assembler lernen

Band 1: Einführung in die Assembler-Programmierung des 6502

1985, ca. 180 S., kart.,
ca. DM 38,—
ISBN 3-7785-1151-3

in Vorbereitung:
Band 2, Nutzung besonderer APPLE-Eigenschaften

Das zweibändige Werk „Apple Assembler lernen" ist ein kompletter Kurs in der Assemblerprogrammierung des 6502 oder 65C02 auf dem Apple II, der nicht da aufhört, wo andere Einführungen auf „weiterführende Literatur" verweisen. Starkes Gewicht wird auf die praktische Anwendung gelegt. Deshalb gehört zum Kurs ein vollwertiger 2-Pass Assembler, der als einer von wenigen die erweiterten Befehle der neuen IIc und IIe Prozessoren 65C02 verarbeitet und der auch lange Programme von mehr als 1 000 Zeilen in wenigen Sekunden übersetzt. Zu seinen professionellen Eigenschaften gehören neben 14 Pseudo-Opcodes, die die Arbeit mit Strings und Tabellen zu einem Kinderspiel werden lassen, ein Zeileneditor mit viel Komfort und die Fähigkeit, Quellcode in verschiedenen Formaten zu schreiben und zu lesen. Ein interaktiver Debugger und Simulator hilft Ihnen, eigene und fremde Maschinenprogramme zu verstehen. Mit seinen vielfältigen und mächtigen Möglichkeiten läßt er Sie hinter die Kulissen Ihres Rechners schauen. Sie können „sehen", was abläuft, Ihre Vorstellungskraft wird angeregt und nicht nur einfach Ihr Gedächtnis strapaziert.
Alle Programme sind 100 % kompakter Maschinencode, nicht einfach compiliertes Basic. Selbstverständlich lernt der Leser sämtliche Maschinenbefehle des Apple und die wichtigen Grundalgorithmen. Aber auch der Umgang mit den eingebauten ROM-Routinen wird ausführlich geübt. Grafik, Sound Stringverwaltung, Fließkommaarithmetik werden ebenso behandelt oder das Zusammenwirken von Applesoft und Assemblerprogrammen.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Hüthig

Arne Schäpers

ProDOS-Analyse

Versionen 1.0.1, 1.0.2, 1.1.1

1985, ca. 450 S., kart.,
DM 68,—
ISBN 3-7785-1134-3

„Die ProDOS Analyse" ist die umfangreichste und detaillierteste Darstellung, die jemals ein Apple-Betriebssystem erfahren hat. Wer die „Innereien" von ProDOS bis zum letzten Byte, z. T. bis ins letzte Bit kennenlernen möchte, braucht dieses Buch. Das komplette Betriebssystem (Urlader, MLI, Disk-Driver, RAM-Disk-Driver und Uhr-Routine) mit Ausnahme des BASIC-SYSTEM wird mit umfangreichen Kommentaren und Übersichtstabellen disassembliert. Dabei werden alle bisherigen Versionen von 1.0.1 bis 1.1.1 berücksichtigt. „Die ProDOS Analyse" beschreibt erstmals auch mehrere Programmierfehler, die bis in die neueste Version zu finden sind. Auch die nicht im „Technical Reference Manual" aufgeführten Eigenschaften von ProDOS werden analysiert und beschrieben, z. B. die vertrackten eingebauten Testroutinen zur Identifikation der verschiedenen Apple II Modelle und eventueller Nachbaugeräte. Programmierer, die ProDOS versionsabhängig „patchen" möchten, erhalten hier den genauen Überblick, wo was geändert werden muß, damit dies keine negativen Konsequenzen hat. Durch die minutiöse theoretische Sezierung von ProDOS eröffnen sich völlig neue programmierpraktische Perspektiven.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1

Arne Schäpers

Hüthig

Bewegte Apple-Graphik

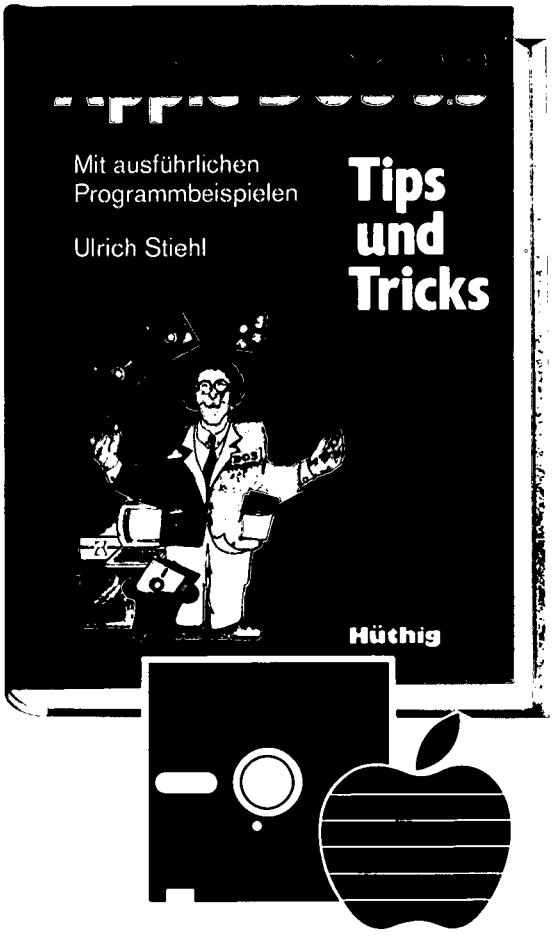
DOS Toolkit-Erweiterungen

1985, ca. 300 S., zahlr. Abb.,
kart., DM 58,—
ISBN 3-7785-1150-5

„Bewegte Grafik, Apple DOS Toolkit Erweiterungen" wendet sich als lehrbuchhafter Kurs an alle, die professionelle hochaufgelöste Grafiken auf dem Apple erzeugen wollen. Der erste Teil beginnt mit einem Abriß des Aufbaus der HGR-Seiten aus der Sicht des Programmierers. Danach wird das Programm HRCG (HI-RES Character Generator, Apple, Inc.) eingehend analysiert und sinnvolle Ergänzungen werden vorgestellt. Schrittweise wird die Nutzung des HRCG erarbeitet bis hin zur beliebigen Bewegung eines statischen Objekts auf einer der HGR-Seiten.

Der zweite Teil baut auf dem ersten auf und führt über die Definition mehrerer Objekte und simultaner Bewegung hin zu einem Arcade-Spiel, das für die meisten käuflichen Action-Spiele in der meisterhaften Grafik als Vorbild dienen kann. Zielgruppe sind Programmierer, die mit Basic keine großen Schwierigkeiten mehr haben und zumindest über praktische Grundkenntnisse in 6502-Assembler verfügen sollten.

Dr. Alfred Hüthig Verlag
Im Weiher 10
6900 Heidelberg 1



Apple DOS 3.3 — Tips und Tricks

von U.Stiehl
1984, 216 S., mit zahlreichen, ausführlich kommentierten Programmlistings,
kart., DM 28,—
2. Auflage 1984
ISBN 3-7785-1049-5

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 eingehend 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. Neben einer detaillierten Beschreibung der DOS-Interna enthält dieser Teil elf vollständige RWTS-Anwendungsprogramme — z.B. CPM-Refiner, DOS-lose Datendisk, TSL-Maker, File-Reader, Pseudo-Disk-Driver und Fastbrun-Routine —, die Techniken enthüllen, die bislang noch niemals publiziert worden sind. So ist z.B. die Fastbrun-Routine 15mal schneller als das normale BRUN. Mit dem File-Reader lassen sich Textfiles beliebiger Größe mit ca. 6000 Zeichen/Sekunde einlesen. Der Pseudo-Disk-Driver ist der erste Apple IIe-64K-Karte-Disk-Emulator für DOS in der Language Card.

Dieses DOS-Buch ist deshalb der unentbehrlich Begleiter für jeden Apple-Programmierer.

DOS 3.3 für Applesoft- und Assembler-Programmierer

Begleitdiskette

Jetzt auch Begleitdiskette zu „Apple DOS 3.3" erhältlich, DM 28,–
ISBN 3-7785-1033-9

Der zweite Band von „ProDOS für Aufsteiger" ist wieder ein typisches Tips- und Tricks-Buch, das viele nützliche Utilities und Hilfsroutinen enthält, die dem Applesoft- und Assembler-programmierer den Umgang mit ProDOS erleichtern sollen.

Zunächst wird eine bewußt leichtverständliche Einführung in die Applesoft-Programmierung unter ProDOS gegeben, die sich insbesondere an diejenigen Leser wendet, die das alte DOS 3.3 nur noch „am Rande" kennengelernt haben.

Im Anschluß daran wird gezeigt, wie man einige Unzulänglichkeiten des BASIC-SYSTEMs durch Tricks beheben kann, z.B. Einlesen von Strings mit Komma und Doppelpunkt, Laden und Speichern von Zahlen als Binärdateien, Simulierung des fehlenden MON-Befehls u.a.

Den Hauptteil des Buches bilden sofort einsatzfähige Utilities, die alle klassischen „Pflichtübungen" abdecken, z.B. Dateilese-programme mit ASCII- und Hex-Dump, Datei-kopierprogramme, Diskettenformatier- und -kopierprogramme, Diskettenvergleichspro-gramme, Konvertierungsprogramme (DOS 3.3 nach ProDOS), Bad-Block-Programme u.a.. Zur Anwendung dieser Utilities sind keinerlei Assemblerkenntnisse erforderlich.