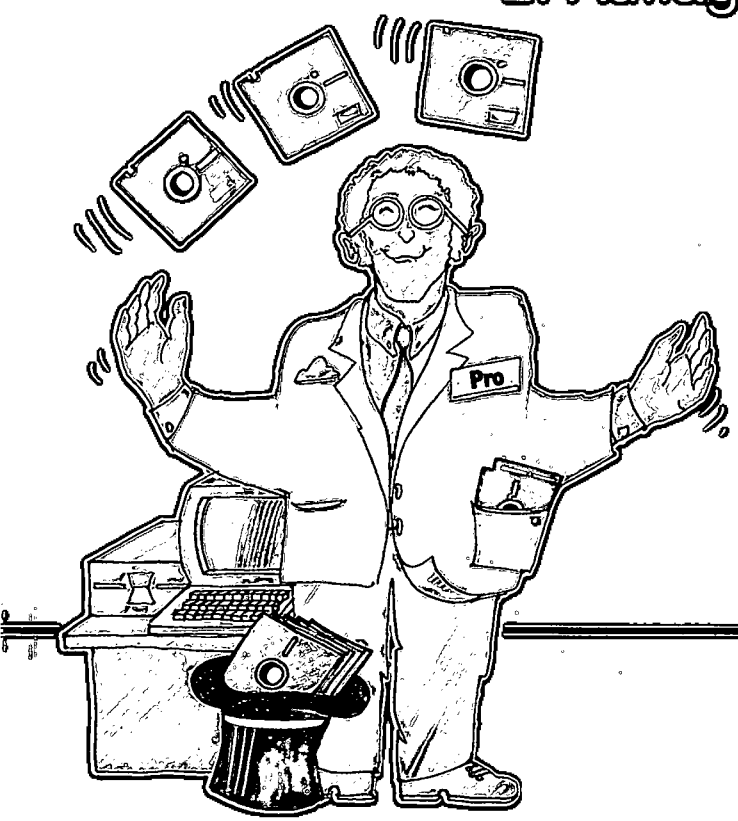
Apple ProDOS

für Aufsteiger

Mit ausführlichen Programmbeispielen

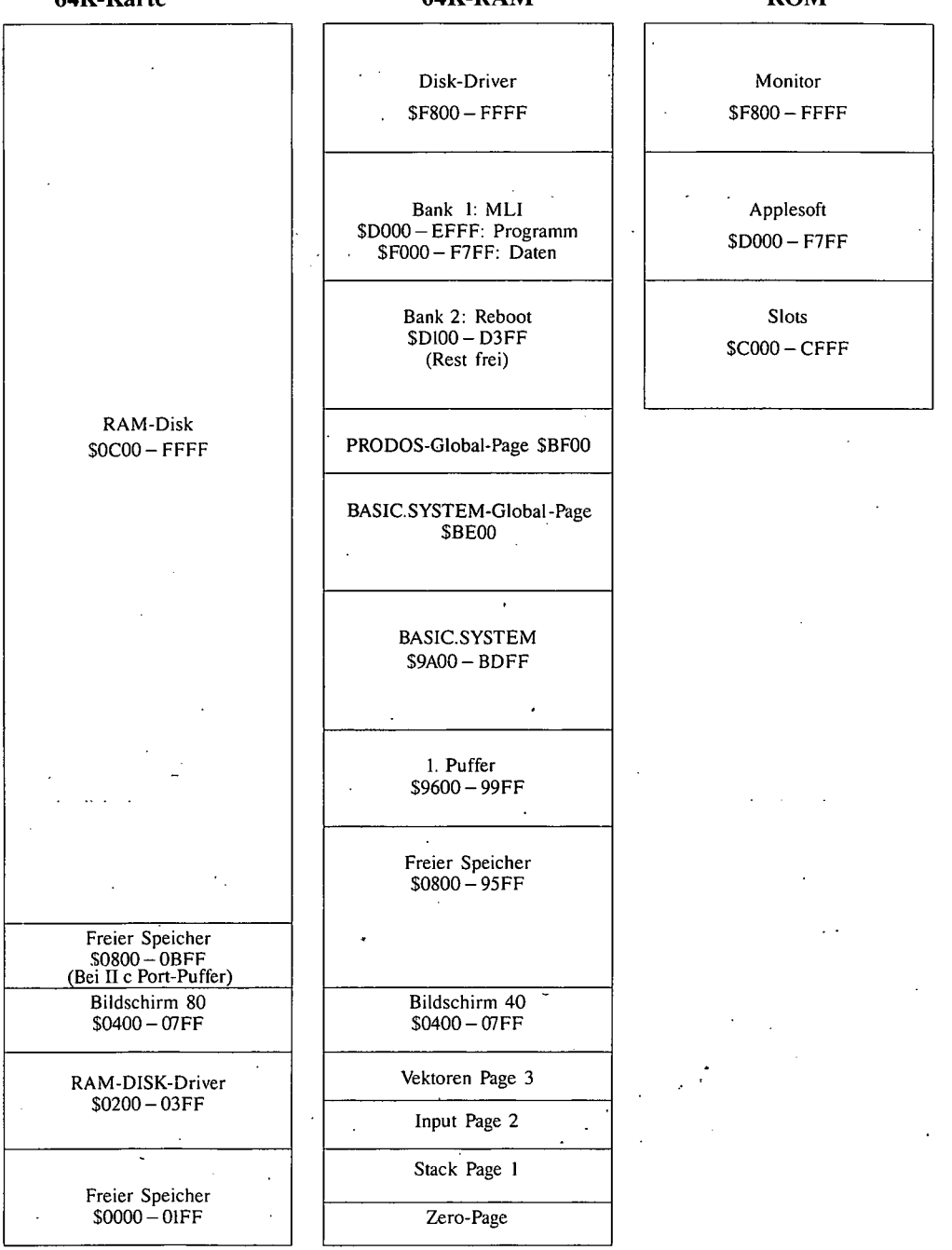
Ulrich Stiehl

2. Auflage



1

Hüthig


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
Disk-Driver
$F800 – FFFF
Bank 1: MLI
$D000 – EFFF: Programm
$F000 – F7FF: Daten
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

Ulrich Stiehl · ProDOS für Aufsteiger

Ulrich Stiehl

ProDOS für Aufsteiger

Mit ausführlichen Programmbeispielen

Band 1

2., ergänzte Auflage

Dr. Alfred Hüthig Verlag Heidelberg

Als Ergänzung zu diesem Buch ist gesondert lieferbar:
„Begleitdiskette zu den ProDOS für Aufsteiger, Band 1"
ISBN 3-7785-1032-0

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Stiehl, Ulrich
ProDOS für Aufsteiger, Band 1
Mit ausführlichen Programmbeispielen - 2., erg. Aufl.
– Heidelberg: Hüthig, 1985
ISBN 3-7785-1098-3

Vorwort

Im Anschluß an mein vor zwei Monaten erschienenes ,,Apple DOS 3.3 – Tips und Tricks'' (nachfolgend zitiert als ,,DOS-Buch'') wird mit dem zweibändigen Werk ,,ProDOS für Aufsteiger'' eine entsprechende Darstellung von ProDOS vorgelegt. Wegen der Komplexität dieses Betriebssystems mußte der Stoff auf zwei Bände verteilt werden.

ProDOS ist das neue ,,professionelle DOS'' (Professional Disk Operating System) für den Apple IIe sowie den mit einer Language Card ausgestatteten Apple II Plus. (Das Original-ProDOS läuft weder auf einem 48 K-Apple noch auf einem 64 K-Apple-Nachbau.) Reine Applesoft-Programmierer, die bereits unter DOS 3.3 programmiert haben, werden sich schneller an ProDOS gewöhnen, da die diesbezüglichen Unterschiede zwischen DOS 3.3 und ProDOS weniger gravierend sind. Sinngemäß liegt das Schwergewicht in dem vorliegenden ersten Band von ,,ProDOS für Aufsteiger'' auf der Assembler-Programmierung, da Assembler-Programmierer unter ProDOS völlig umdenken müssen. Insbesondere sind alle früheren Assemblerprogramme unter ProDOS nicht mehr lauffähig und bedürfen einer intensiven Überarbeitung. Dies gilt selbstverständlich nur für file-orientierte Programme und nicht etwa für Spiele oder sonstige Anwendungen, bei denen nach dem Starten des Programms nicht mehr auf externe Datenträger zugegriffen wird.

Die von der Firma Apple herausgegebenen Bände über ProDOS, insbesondere das ,,ProDOS Technical Reference Manual'' (nachfolgend zitiert als ,,Technical Manual''), sind für weit fortgeschrittene Assembler-Programmierer ausgezeichnet geeignet, doch enthält das ,,Technical Manual'' fast keine Anwendungsbeispiele und kaum Angaben von Systemadressen, so daß die Einarbeitung in dieses neue Betriebssystem zeitaufwendig ist. Deshalb wird man die in dem vorliegenden Band enthaltenen grundlegenden Programmbeispiele sowie die zahlreichen Disketten-Dumps (Block-Dumps), die die Diskettenstruktur bis zum letzten Byte auseinanderpflücken,

begrüßen. Es sei jedoch darauf hingeweisen, daß dieses ,,ProDOS-Buch'' vom Schwierigkeitsgrad her über das ,,DOS-Buch'' hinausgeht, da die dort geschilderten Grundlagen – insbesondere für Apple-Neulinge – hier nicht noch einmal wiederholt werden können. Alle drei Bände gehören deshalb sachlich zusammen. Begriffe, die in diesem Buch nicht erklärt werden – z.B. Skewing, Booten usw. –, schlage man im Register des ,,DOS-Buches'' nach. Im einzelnen unterscheiden sich die drei Bände wie folgt:

– ,,Apple DOS 3.3'' enthält neben einer ausführlichen Darstellung des gesamten Betriebssystems anhand zahlreicher praktischer Beispielprogramme sowohl für Applesoft- wie auch Assembler-Programmierer eine kurze Einführung in den Umgang mit Disketten für reine Anwender und Apple-Neulinge.

– ,,ProDOS für Aufsteiger, Band 1'' befaßt sich überwiegend mit den theoretischen Grundlagen von ProDOS, der internen und externen Speicherorganisation und enthält grundlegende Beispielprogramme für Assembler-Programmierer sowie generelle Untersuchungen zum BASIC.SYSTEM. Da ProDOS über erheblich vielfältigere und leistungsfähigere, zugleich jedoch erheblich kompliziertere Dateistrukturen verfügt, sind theoretische Kenntnisse von ProDOS unabdingbar, wenn man die Features von ProDOS voll ausschöpfen will.

,,ProDOS für Aufsteiger, Band 2'' beschreibt ausführlich die Anwendung des ProDOS-BASIC.SYSTEM für Applesoft-Programmierer und enthält darüber hinaus zahlreiche größere Assembler-Anwenderprogramme, die aus Platzgründen in dem ersten Band nicht mehr untergebracht werden konnten.

Dieser zweite Band wird im ersten Quartal 1985 auf den Markt kommen.

Heidelberg, im November 1984

Ulrich Stiehl

Inhalt

1. ProDOS für Assembler-Programmierer

1. ProDOS für Assembler-Programmierer

1.1. Ein erster Überblick

ProDOS – entwickelt von der Firma Apple Computer – ist das neue Diskettenbetriebssystem für den Apple IIe und den Apple II Plus mit Language Card, das in den USA Anfang 1984 und in der Bundesrepublik im April 1984 (zur Hannover-Messe) auf den Markt kam. Testversionen wurden bereits seit etwa März 1983 an Software-Entwickler verteilt.

Anmerkung: Unter „ProDOS" verstehen wir das gesamte Betriebssystem und unter „PRODOS" den in die Language Card geschobenen Teil des Betriebssystems, der auf der ProDOS-Diskette unter dem Namen „PRODOS" abgelegt ist. ProDOS umfaßt die Files „PRODOS" und „BASIC.SYSTEM".

Als erster Überblick seien stichwortartig einige markante Merkmale von ProDOS aufgezählt:

1.1.1. ProDOS, SOS und Apple Pascal

ProDOS für den Apple IIe hat dieselbe Dateistruktur wie SOS für den Apple III, so daß ein Datenaustausch zwischen Apple II und III vorgenommen werden kann. ProDOS-Befehle sind eine Teilmenge der SOS-Befehle. Während SOS auch andere Peripherie-Geräte (Drucker usw.) überwacht, dient ProDOS ausschließlich dem Disketten-Handling. Wenn auch Apple III Programme nicht ohne weiteres auf dem Apple II laufen und umgekehrt, können jetzt bootfähige, kombinierte Programm-Disketten hergestellt werden, die je nach Maschine das ProDOS- oder SOS-Anwenderprogramm laden.
ProDOS ist mit Apple Pascal nicht kompatibel, doch haben beide eine gleichartige

Diskettenorganisation. Dies betrifft zum einen das Skewing (siehe ,,DOS-Buch'', S. 111ff.) sowie zum anderen die Blockorganisation (1 Block bei ProDOS und Apple Pascal = 2 Sektoren bei DOS 3.3).

1.1.2. Interner Speicherbedarf

ProDOS setzt einen 64K Apple voraus, d.h. entweder einen Apple IIe oder einen Apple II Plus mit Language Card. Das Betriebssystem ist quasi zweigeteilt: Zum einen gibt es das ,,eigentliche'' PRODOS, das auf der ProDOS-Betriebssystemdiskette (,,User's Disk'') unter dem Dateinamen ,,PRODOS'' abgespeichert ist und nach dem Booten die oberen 16K belegt. Zum anderen befindet sich auf derselben Diskette das Systemprogramm namens ,,BASIC.SYSTEM'', das als Schnittstelle zwischen PRODOS und dem Applesoft-Interpreter fungiert. Dieses BASIC.SYSTEM nimmt in den unteren 48K denselben Speicherraum ein wie das alte, *gesamte* DOS 3.3.

1.1.3. Disketten- und Festplattenlaufwerke

Während bei DOS 3.3 die sog. RWTS (siehe ,,DOS-Buch'') ursprünglich nur für normale Diskettenlaufwerke bestimmt war, ist ProDOS umgekehrt primär für Festplattenlaufwerke und erst sekundär für Diskettenlaufwerke gedacht. Dies spiegelt sich in der Dateiorganisation wider: Unter ProDOS kann ein einzelner externer Datenträger (,,Volume'') insgesamt 32 Megabytes und eine einzelne Datei insgesamt 16 Megabytes umfassen (1 Byte = 1 Zeichen; 1 Kilobyte = 1024 Bytes; 1 Megabyte = 1024 · 1024 = 1.048.576 Bytes; 16 Megabytes = 16.777.216 Bytes).

DOS-Kenner beachten, daß man unter ,,Volume'' jetzt einen externen Datenträger (z.B. Diskette) sowie den Namen, d.h. Volume-Namen, des externen Datenträgers versteht. Die sog. Volume-Nummer im Sinne von DOS 3.3 spielt bei ProDOS überhaupt keine Rolle mehr. Unter ProDOS formatierte Disketten haben stets die Volume-Nummer 1 ($01), doch ist die Volume-Nummer willkürlich und damit ignorierbar, da Datenträger mit dem Volume-Namen angesprochen werden müssen.

1.1.4. Blocks und Sektoren

Der Zugriff auf externe Datenträger erfolgt bei ProDOS stets auf Block-Ebene, wobei ein Block als $0200 oder 512 Bytes definiert wird. Genauer gesagt ist das Betriebssystem PRODOS in zwei Teile gegliedert: Der erste Teil (Command Dispatcher und Block File Manager) arbeitet auf Block-Ebene, während der zweite Teil (,,Disk Driver'') auf Spuren-Sektoren-Ebene fungiert. Bei physischen Diskettenlaufwerken (im Gegensatz zu RAM-Disks usw.) entspricht

1 512-Byte-Block = 2 256-Byte-Sektoren (linke und rechte Block-Hälfte).

ProDOS ist insoweit mit dem Apple-Pascal-Betriebssystem identisch, das ebenfalls auf Block-Ebene auf externe Datenträger zugreift.

1.1.5. Hierarchische Catalog-Struktur

Das auf den ersten Blick auffälligste Merkmal von ProDOS ist die hierarchische, quasi dezimalklassifikatorische Struktur des Disketteninhaltsverzeichnisses. Zwischen dem Inhaltsverzeichnis eines Buches und dem Inhaltsverzeichnis einer ProDOS-Diskette gibt es eine direkte Analogie: Ein Buch-Inhaltsverzeichnis gliedert sich in Teile, die Teile in Hauptkapitel, die Hauptkapitel in Unterkapitel, die Unterkapitel in Abschnitte usw. Sinngemäß kann man das Hauptinhaltsverzeichnis („Volume-Directory") bei ProDOS in Unterinhaltsverzeichnisse („Subdirectories") und diese wiederum in Unter-Unterinhaltsverzeichnisse („Sub-Subdirectories") untergliedern usw. Beispiel:

1. Volume-Directory VOLUME
   a) PRODOS
   b) BASIC.SYSTEM
   c) STARTUP
1.1. Subdirectory PROGRAMME
1.1.1. Sub-Subdirectory UTILITIES
   a) BIG MAC
   b) MACROEDITOR
1.1.2. Sub-Subdirectory ANWENDUNGEN
   a) APPLEWRITER
   b) VISICALC
1.2. Subdirectory DATEN
   a) BRIEF1
   b) KALKULATION1

Wenn man eine ProDOS-Diskette neu formatiert, erhält sie einen Namen, z.B. „VOLUME", und ein Volume-Directory, das 51 File-Namen umfassen kann, z.B. „PRODOS", „BASIC.SYSTEM" und „STARTUP". (Die eigentlichen File-Namen sind bei dem obigen Beispiel mit den Buchstaben a, b, c gekennzeichnet, während die Directory-Namen dezimalklassifikatorisch mit 1., 1.1., 1.1.1. usw. hervorgehoben werden.)

Mit dem CATALOG-Befehl

CATALOG/VOLUME

sieht man dann diese drei File-Namen „PRODOS", „BASIC.SYSTEM" und „STARTUP". Das Volume-Directory kann jedoch anstelle der oder neben den eigentlichen File-Namen auch Directory-Namen enthalten, z.B. die Subdirectory-Namen „PROGRAMME" und „DATEN". Diese Unterinhaltsverzeichnisse werden mit den CREATE-Befehlen

CREATE/VOLUME/PROGRAMME und
CREATE/VOLUME/DATEN

erzeugt, wobei darauf hingewiesen sei, daß ein Subdirectory beliebig viele File-Namen umfassen kann. Ferner beachte man, daß die Befehle und Teilnamen durch Schrägstriche abgegrenzt werden.

Aber auch z.B. das Unterinhaltsverzeichnis „VOLUME/PROGRAMME" läßt sich seinerseits wiederum in Unter-Unterinhaltsverzeichnisse aufspalten, z.B. mit dem Create-Befehl

CREATE/VOLUME/PROGRAMME/UTILITIES

Gibt man danach etwa den Befehl

CATALOG/VOLUME/PROGRAMME/UTILITIES

dann sieht man nur noch die unter den Utilities eingeordneten eigentlichen File-Namen „BIG MAC" und „MACROEDITOR".

Mit den PREFIX-Befehlen

PREFIX/VOLUME/PROGRAMME/UTILITITES oder
PREFIX/VOLUME/PROGRAMME/ANWENDUNGEN oder
PREFIX/VOLUME/DATEN usw.

kann man alle nachfolgenden Diskettenbefehle (CATALOG, LOAD, SAVE usw.) gezielt auf die gewünschten Subdirectories einstellen und mit etwa

PREFIX/VOLUME

allein kann man dann wieder auf das Volume-Directory zurückschalten.

Bei ProDOS sind Unterinhaltsverzeichnisse selbst Dateien, wie im übrigen auch die Betriebssystemteile „PRODOS" und „BASIC.SYSTEM" auf der Diskette als Dateien gespeichert sind. (Anders bei DOS 3.3, wo es eine DOS-Catalog-Spur sowie drei DOS-Betriebsystem-Spuren gibt.) Lediglich das Volume-Directory sowie einige andere „System-Blocks" (Boot-Programm, Volume Bit Map usw.) haben auf der Diskette einen festen Platz. Denn das Volume-Directory enthält die Zeiger bzw. Block-Nummern der Subdirectories des ersten Grades, weil die Suche nach einem bestimmten „Sub-Sub-Subdirectory-File-Namen" naturgemäß an einem bestimmten, festdefinierten Ausgangspunkt beginnen muß, und dies ist stets der erste Block des Hauptinhaltsverzeichnisses.

1.1.6. Interrupt

Die Firma Apple betont als besonderes Merkmal von ProDOS die Interrupt-Fähigkeit. Insgesamt können bis zu 4 Interrupt-Driver installiert werden, wobei für eine bestimmte Uhr („Thunderclock") ein Driver bereits im ProDOS enthalten ist. Die Interrupt-Schilderungen von Apple sind jedoch dahingehend mißverständlich, daß z.B. während des Diskettenzugriffs ein Interrupt stattfinden könnte. Dies ist indes weder vorgesehen noch möglich, da bekanntlich das Lesen von und Schreiben auf physische Disketten extrem zeitkritisch ist.

Technischer Hinweis: Exakt in dem Moment, da ProDOS den Sektorvorspann $D5 $AA $96 gefunden hat, wird bei $FBC3 in der Language Card mit dem Befehl SEI vorübergehend jeder Interrupt unterbunden. Insoweit unterscheidet sich ProDOS interruptmäßig nicht von DOS 3.3.

1.2. ProDOS und DOS 3.3

Obgleich das „DOS-Buch" einige Schwächen des alten DOS 3.3 aufzeigte, wurde gleichzeitig die Leistungsfähigkeit und Flexibilität dieses Betriebssystems für die normalen Apple-Diskettenlaufwerke unterstrichen, insbesondere wenn erstens anstelle des Original-DOS 3.3 die fortgeschrittenen DOS-Varianten wie Diversi-DOS – entwickelt von Diversified Software Research – eingesetzt werden und wenn zweitens der Diskettenzugriff in Spezialfällen durch Assembler-Routinen beschleu-

nigt wird. Um es gleich vorwegzunehmen: Gegenüber dem alten DOS 3.3 ist ProDOS erheblich professioneller, wie die Abkürzung ,,ProDOS" (= Professional Disk Operating System = professionelles Diskettenbetriebssystem) bereits nahelegt. Anders sieht es demgegenüber aus, wenn man ProDOS mit einer fortgeschrittenen DOS-Variante, eben z.B. mit Diversi-DOS, vergleicht. Dann ergeben sich folgende Gemeinsamkeiten und Unterschiede:

1.2.1. Basic

Im Gegensatz zu dem Apple Pascal UCSD Betriebssystem sind ProDOS und Diversi-DOS für die Programmiersprachen Assembler und Basic gedacht, doch während Diversi-DOS nicht nur unter Applesoft-, sondern auch unter Integer-Basic lauffähig ist, ist ProDOS nur für Applesoft implementiert. Dies ist kein gewichtiger Nachteil, zumal Integer-Basic ohnehin meist nur für Spielprogramme verwendet wurde.

1.2.2. Freier Speicherraum

Im Idealfall läßt sich Diversi-DOS mit Hilfe des DOS-Movers in die Language Card Bank 2 verschieben. Erweitert man Diversi-DOS noch um einen RAM-Disk-Driver für die 64K-Karte für den Apple IIe (der in dem ,,DOS-Buch" gelistet wurde), dann ergibt sich folgender freier Speicherraum bei MAXFILES 5 (dies gilt speziell für die Version 2-C, da die neueste Version 4-C nur 3 Puffer in der Bank 2 zuläßt):

$0800-$BDFF	frei
$D000-$DFFF	Bank 1 der LC frei

d.h. insgesamt $C600 = 50.688 Bytes Arbeitsspeicher, und davon wieder $B600 = 46.592 Bytes als homogener Block.

Bootet man ProDOS auf einem Apple IIe mit 64K Karte, dann wird ProDOS automatisch in die Language Card geschoben, und zwar komplett in die Bank 1 sowie teilweise in die Bank 2. Ferner wird ein RAM-Disk-Driver installiert und das BASIC.SYSTEM, d.h. derjenige Teil des Betriebssystems, der als Schnittstelle zwischen Applesoft und ProDOS dient, in den Bereich $9600-$BFFF gelegt, und zwar mit zunächst nur einem einzigen geöffneten Puffer. Für jeden zusätzlichen Puffer werden $400 Bytes benötigt. (Ein I/O-Puffer unter ProDOS ist fast doppelt so groß wie unter DOS 3.3.) Bei etwa 3 belegten Puffern, was in der Praxis häufig vorkommt, liegt damit HIMEM bei $8E00, mithin ergibt sich folgender freier Speicherbereich

$0800-$8DFF frei unter PRODOS + BASIC.SYSTEM

d.h. $8600 = 34.304 Bytes. Damit hat man also bei ProDOS exakt 16K = 16.384 Bytes weniger als Arbeitsspeicher für Programme und Daten. Dies ist zweifelsohne ein gravierender Nachteil, der auch verständlich macht, weshalb Integer-Basic nicht unter ProDOS implementiert wurde oder werden kann, denn Applesoft liegt im ROM, nimmt also insoweit keinen zusätzlichen Speicherraum weg. Integer-Basic oder eine andere Programmiersprache im RAM würde ihrerseits etwa 16K beanspruchen, womit unter ProDOS und einer anderen RAM-gespeicherten Programmiersprache der freie Speicher nicht einmal mehr für kurze „Spielchen" ausreichen würde.

1.2.3. Datenübertragungsrate

Die nächste Frage, die uns interessiert, ist die Geschwindigkeit des Diskettenzugriffs. Hier zeigt sich, das ProDOS gegenüber Diversi-DOS (und im übrigen auch gegenüber DOS 3.3) auf der untersten Track-Sektor-Ebene etwa 18,5% schneller ist, weil bei ProDOS ähnlich wie bei Apple Pascal grundsätzlich nur 512-Byte-Blocks, d.h. je zwei 256-Byte-Sektoren auf einmal, eingelesen werden, d.h. ein reines Assemblerprogramm, das auf Block-Ebene auf externe Speicher zugreift, ist stets fast 20% schneller als DOS/Diversi-DOS, wenn und nur dann wenn die einzulesenden Informationen größer/gleich 1 Block sind. Beispiel: Wenn die Records eines Random-Files mehr als 256 Bytes umfassen, ist ProDOS schneller, und wenn die Records weniger als 256 Bytes umfassen, ist DOS 3.3 schneller. Oder anders formuliert: Je größer die einzulesenden oder zu speichernden Dateien, desto relativ schneller ist Prodos.

Da üblicherweise nicht nur auf dieser elementaren Block-Ebene programmiert wird, müssen auch die anderen Zugriffsgeschwindigkeiten in Betracht gezogen werden. Im einzelnen gilt:

1.2.3.1. Speichern und Laden von Binärfiles

(Binärfiles = Maschinenprogramme, Applesoftprogramme, Hires-Bilder usw.)

Es wurde ein 32K großer Speicherbereich wiederholt mit BSAVE gespeichert (FOR X = 1 TO 20 : PRINT CHR$ (4) ,,BSAVE XXX, A3000, L 32767": NEXT) und wiederholt mit BLOAD geladen (FOR X = 1 TO 20 : PRINT CHR$ (4) ,,BLOAD XXX": NEXT). Als Mittelwerte ergaben sich:

Befehl	ProDOS-Disk II	Diversi-DOS
BSAVE	22 s	10 s
BLOAD	6 s	7 s

Die Tabelle zeigt, daß der BSAVE-Befehl (und sinngemäß der SAVE-Befehl) bei Diversi-DOS mehr als doppelt so schnell ist wie bei ProDOS, während umgekehrt der BLOAD-Befehl (und sinngemäß der LOAD-Befehl) bei ProDOS geringfügig schneller als bei Diversi-DOS ausgeführt wird. Die Begründung für die langsamere BSAVE-Ausführung trotz der Größe der Datei (32K) ergibt sich aus der weiter unten geschilderten ,,Buchhaltung".

Derselbe BSAVE-BLOAD-Test wurde auch auf dem Profile-Festplattenlaufwerk und der 64K-RAM-Disk ausgeführt, wobei allerdings hier der 32K-File insgesamt je 100mal gespeichert und geladen wurde, um genauere Ergebnisse zu erzielen:

Befehl	ProDOS-Profile	ProDOS-RAM-Disk	Diversi-DOS-RAM-Disk
BSAVE	4.05 s	1.88 s	0.83 s
BLOAD	1.70 s	0.58 s	0.77 s

Aus der Tabelle ist ersichtlich, daß eine RAM-Disk mehr als doppelt so schnell wie die Profile ist. Hinzu kommt, daß sich die 64K-RAM-Karte wegen des umständlichen Bank-Selecting als RAM-Disk nicht so gut eignet wie die auf dem Markt erhältlichen 128K- und 256K-RAM-Karten, bei denen nach eigenen Tests die Datenübertragungsrate bei einem normalen 1-Megahertz-6502-Prozessor fast 64K/s beträgt gegenüber ca. 50K/s bei der 64K-Karte. Damit ist bei 128K/256K-RAM-Karten die Datenübertragung im Mittel dreimal schneller als bei der Profile.

1.2.3.2. Speichern und Laden von Strings (Textfiles)

(Strings = Zeichenketten, z.B. ,,xxxxx")

Es wurden 500 Strings mit je 99 Zeichen Länge (+ Return) mit WRITE gespeichert und mit READ geladen. Die so entstandene Datei hatte eine Länge von 50.000 Zeichen:

Befehl	ProDOS	Diversi-DOS
WRITE	34 s	60 s
READ	28 s	48 s

Das Applesoft-Programm sah bei dem Write-Test so aus:

```
10 PRINT CHR$ (4) ,,OPEN TEST" : PRINT CHR$ (4) ,,WRITE TEST"
20 FOR X = 1 TO 500: PRINT ,,123456789 123456789 123456789 123456789
   123456789 123456789 123456789 123456789 123456789 123456789" : NEXT
30 PRINT CHR$ (4) ,,CLOSE"
```

Hier zeigt sich, daß ProDOS beim WRITE-Befehl fast doppelt so schnell wie Diversi-DOS ist, während der READ-Befehl immerhin noch etwa ein Drittel schneller als bei Diversi-DOS abgearbeitet wird. Verfeinerte Untersuchungen ergaben, daß die relative Geschwindigkeit von ProDOS mit der String-Länge steigt, da sich bei langen Strings der größere Input-Output-Puffer günstiger auswirkt.

1.2.3.3. Speichern und Laden von Zahlen (Textfiles)

Zahlen werden auch unter ProDOS normalerweise als Textfiles, also quasi als Ziffernfolgen und nicht binär verschlüsselt, gespeichert. Zu Testzwecken wurden 5.000 neunstellige Zahlen mit WRITE gespeichert und mit READ eingelesen. Die so entstandene Datei hatte eine Länge von 50.000 Zeichen:

Befehl	ProDOS	Diversi-DOS
WRITE	72 s	90 s
READ	86 s	77 s

Hier ist ProDOS beim WRITE-Befehl schneller, dagegen bei READ-Befehl langsamer als Diversi-DOS, was wohl mit der Kürze der Zahlen-String-Länge zusammenhängt.

1.2.3.4. Speed Test 1: Block-Lesen bei Disk II Laufwerk

Der nachfolgend gelistete zweiteilige ,,Speed Test 1" – der später programmtechnisch erläutert wird – zeigt zweierlei:

Erstens beträgt die Datenübertragungsrate ca. 9100 Bytes/Sekunde bei ProDOS gegenüber ca. 7400 Bytes/Sekunde bei DOS 3.3, d.h. ProDOS ist ca. 18,5% schneller als DOS 3.3.

Zweitens ist die optimale Übertragungsrate bei ProDOS nur dann zu erzielen, wenn die Blocks – und sinngemäß die Sektoren – aufsteigend, d.h. von $0000-$0117 eingelesen werden, während bei DOS 3.3 die Sektoren absteigend, d.h. von $0F-$00 gelesen werden müssen, falls optimaler Diskettenzugriff gewährleistet sein soll. ProDOS arbeitet also – ähnlich wie Pascal – nach dem Ascending Skew Verfahren (gegenüber dem Descending Skew Verfahren bei DOS 3.3).

1.2.3.5. Speed Test 2: Block-Lesen bei 64K RAM-Disk

Der „Speed Test 2" liest von der Apple IIe RAM-Disk ca. 1 Megabyte ein. Dies entspricht einer Datenübertragungsrate von ca. 52,5 K/Sekunde und ist in etwa auch die maximale Zugriffsgeschwindigkeit, die bei Festplattenlaufwerken (Profile, Corvus) theoretisch erzielt werden kann. (Bei RAM-Disks und Festplattenlaufwerken besteht darüber hinaus die Möglichkeit, durch Verwendung eines schneller getakteten 6502-Prozessors – z.B. Accelerator-Karte mit 3.5 Megahertz – den Zugriff entsprechend zu steigern. Dies ist bei den zeitkritischen physischen Diskettenzugriffen nicht möglich.)

```
                        1                ORG  $6000
                        2    *
                        3    * Speed Test 1: Disk II
                        4    * ======================
                        5    *
                        6    * 3 X Lesen von 35 Tracks
                        7    * ------------------------
                        8    *
                        9    * AUFSTEIGEND
                        10   * -----------
                        11   * Von Block 00-FF aufsteigend
                        12   * = 46.5 Sekunden
                        13   *
                        14   * Das sind  ca. 9100 Bytes/Sek.
                        15   * bei Prodos
                        16   * gegenüber ca. 7400 Bytes/Sek.
                        17   * bei DOS 3.3.
                        18   *
                        19   * Prodos ist also ca. 18,5%
                        20   * schneller als DOS 3.3 auf
                        21   * unterster Track-Sektor-
                        22   * Ebene bei normalen Disk II
                        23   * Laufwerken.
                        24   *
6000: 20 06 60          25            JSR  LOOP1A
6003: 20 06 60          26            JSR  LOOP1A
6006: A2 00             27   LOOP1A   LDX  #0
6008: 8E 3E 60          28   LOOP1    STX  BLOCK
600B: 20 2C 60          29            JSR  RDBLOCK
600E: AE 3E 60          30            LDX  BLOCK
6011: E8                31            INX                ;00-FF
6012: D0 F4             32            BNE  LOOP1
6014: 60                33            RTS
                        34   *
                        35   * ABSTEIGEND
                        36   * ----------
                        37   * Von Block FF-00 absteigend
                        38   * = 111.5 Sekunden
                        39   *
                        40   * Im Gegensatz zu DOS 3.3 müssen
                        41   * also Blocks stets aufsteigend
                        42   * gelesen werden, wenn ein
                        43   * optimaler Zugriff gegeben sein
                        44   * soll.
                        45   *
6015: 20 1B 60          46            JSR  LOOP2A
6018: 20 1B 60          47            JSR  LOOP2A
601B: A2 FF             48   LOOP2A   LDX  #$FF
601D: 8E 3E 60          49   LOOP2    STX  BLOCK
6020: 20 2C 60          50            JSR  RDBLOCK
6023: AE 3E 60          51            LDX  BLOCK
6026: CA                52            DEX                ;FF-00
6027: E0 FF             53            CPX  #$FF
```

```
6029: D0 F2     55              BNE   LOOP2
602B: 60        56              RTS
                57    *
602C: 20 00 BF  58    RDBLOCK   JSR   $BF00        ;MLI
602F: 80        59    READ      HEX   80
6030: 3A        60              DFB   #<CMDLIST
6031: 60        61              DFB   #>CMDLIST
6032: F0 05     62              BEQ   EXIT
6034: 20 DA FD  63    ERROR     JSR   $FDDA        ;HEXOUT
6037: 68        64              PLA
6038: 68        65              PLA
6039: 60        66    EXIT      RTS
                67    *
603A: 03        68    CMDLIST   HEX   03
603B: 60        69    UNITNO    HEX   60          ;S6,D1
603C: 00 10     70    BUFFER    HEX   0010        ;$1000
603E: 00 00     71    BLOCK     HEX   0000        ;LO-HI

--End assembly--

64 bytes

Errors: 0
```

Hätten Sie's gewußt?

Wenn man unter dem BASIC.SYSTEM Textfiles mit RDKEY (JRS $FD0C) einzulesen versucht, bricht ProDOS zusammen oder liest nur „Schrott".
Wenn man BASIC.SYSTEM-Befehle über COUT (JSR $FDED) ausgeben will, werden sie nicht erkannt und nur am Bildschirm angezeigt.
(Grund: Pseudo-Trace)

```
                      1                ORG  $6000
                      2     *
                      3     * Speed Test 2: 64K RAM-Disk
                      4     * ==========================
                      5     *
                      6     * 20 X Lesen von 100 Blocks
                      7     * entspricht ca. 1 Megabyte
                      8     *
                      9     * ca. 19 Sek. = ca. 52.5 K/Sek.
                      10    *
                      11    *
6000: A9 14           12               LDA  #20          ;20 MAL
6002: 8D 1F 60        13               STA  COUNT
6005: 20 0E 60        14    MAINLOOP   JSR  LOOP0
6008: CE 1F 60        15               DEC  COUNT
600B: D0 F8           16               BNE  MAINLOOP
600D: 60              17               RTS
                      18    *
600E: A2 0A           19    LOOP0      LDX  #10          ;10-110
6010: 8E 32 60        20    LOOP1      STX  BLOCK
6013: 20 20 60        21               JSR  RDBLOCK
6016: AE 32 60        22               LDX  BLOCK
6019: E8              23               INX               ;00-FF
601A: E0 6E           24               CPX  #110
601C: D0 F2           25               BNE  LOOP1
601E: 60              26               RTS
                      27    *
601F: 00              28    COUNT      HEX  00
                      29    *
6020: 20 00 BF        30    RDBLOCK    JSR  $BF00        ;MLI
6023: 80              31    READ       HEX  80
6024: 2E              32               DFB  #<CMDLIST
6025: 60              33               DFB  #>CMDLIST
6026: F0 05           34               BEQ  EXIT
6028: 20 DA FD        35    ERROR      JSR  $FDDA        ;HEXOUT
602B: 68              36               PLA
602C: 68              37               PLA
602D: 60              38    EXIT       RTS
                      39    *
602E: 03              40    CMDLIST    HEX  03
602F: B0              41    UNITNO     HEX  B0           ;S3,D2
6030: 00 10           42    BUFFER     HEX  0010         ;$1000
6032: 00 00           43    BLOCK      HEX  0000         ;LO-HI

--End assembly--

52 bytes

Errors: 0
```

1.2.3.6. Disketten-,,Buchhaltung"

Entscheidend für die Gesamtbeurteilung des Diskettenzugriffs ist der Zeitaufwand für die Disketten-,,Buchhaltung", da üblicherweise über die Blocks einer Datei in dem Volume-Directory, den Subdirectories sowie in der Volume Bit Map und den Index-Blocks ,,Buch geführt" werden muß. (Die erwähnten Begriffe werden später erläutert.) Hier zeigt sich leider, daß ProDOS dem alten DOS 3.3 hoffnungslos unterlegen ist. Im einzelnen wurden 2 Test in unterschiedlichen Varianten mit den normalen Apple-Diskettenlaufwerken durchgeführt:

Beim ersten Test wurden mit dem folgenden Applesoft-Programm

```
10 FOR X = 1 TO 51 : PRINT CHR$ (4) ,,SAVE XXX"; X: NEXT
20 FOR X = 1 TO 51 : PRINT CHR$ (4) ,,DELETE XXX"; X: NEXT
```

51 Applesoft-Files mit einer Länge von jeweils weniger als 1 Block erzeugt und im Anschluß daran wieder gelöscht, wobei bei ProDOS in 2 Teiltests die Speicherung sowohl in dem Volume-Directory als auch in einem Subdirectory vorgenommen wurde:

Befehl	Diversi-DOS	Volume-Directory	Subdirectory
SAVE + DELETE	207 s	297 s	371 s

Hier ist Diversi-DOS immerhin schon 50-90% schneller als ProDOS. Das relativ ,,günstige" Abschneiden von ProDOS rührt daher, daß bei 1-Block-Dateien der sog. Index-Block entfällt.

Beim zweiten Test wurden mit dem folgenden Applesoft-Programm

```
10 FOR X = 1 TO 51 : PRINT CHR$ (4) ,,BSAVE XXX"; X ,,,A 5000, L 550"
   : NEXT
20 FOR X = 1 TO 51 : PRINT CHR$ (4) ,,DELETE XXX"; X : NEXT
```

51 Binärfiles mit einer Länge von nunmehr 2 Blocks zunächst erzeugt und dann wieder gelöscht. Da bei ProDOS erst ab 2 Datenblocks ein gesonderter Index-Block erforderlich ist, schlägt erst hier die zeitaufwendige ProDOS-,,Buchführung" voll durch:

Befehl	Diversi-DOS	Volume-Directory	Subdirectory
BSAVE + DELETE	223 s	526 s	692 s

Diversi-DOS ist nunmehr sogar über 300% schneller als ProDOS. Dieser Test zeigt deutlich, daß der Diskettenzugriff auf 2-Sektoren-Ebene (= Block-Ebene) einerseits um bis zu 20% schneller ist, wenn ,,Mammut''-Dateien gelesen oder gespeichert werden müssen, daß aber umgekehrt dieser Geschwindigkeitsvorteil durch die ebenfalls auf Block-Ebene vorgenommene ,,Buchführung'' mehr als aufgehoben wird. Insgesamt lassen sich folgende praktische Schlußfolgerungen ziehen:

a) Die aufwendige Buchführung, die ja dann auch die Anlage von Dateien mit einem Umfang von bis zu 16 Megabytes zuläßt, ist bei Festplattenlaufwerken angemessen, zumal der mit der ,,Buchführung'' zusammenhängende zeitraubende Spurenwechsel – das Volume-Directory befindet sich bei physischen Disketten auf der äußersten Spur 0 – bei Harddisks nicht ins Gewicht fällt. ProDOS ist damit das ideale Betriebssystem für Festplattenlaufwerke.

b) Für normale Apple-Diskettenlaufwerke bringt ProDOS gegenüber Diversi-DOS im großen und ganzen keinerlei Geschwindigkeitsvorteile, und wenn mit Subdirectories gearbeitet wird, ist ProDOS sogar langsamer als Diversi-DOS. Bei normalen Disketten sollte man also auf Subdirectories am besten ganz verzichten, zumal die maximal 51 File-Namen, die das Volume-Directory zuläßt, für 140K-Disketten fast immer ausreichend sind (140K brutto minus ,,System-Blocks'' geteilt durch 51 File-Namen = ca. 2,5K pro File.)

Daß gleichzeitig mit ProDOS zur Hannover-Messe Anfang April 1984 als ,,neue'' Apple-Laufwerke die sog. Duo-Drives (= Doppellaufwerke mit je 140K) vorgestellt wurden, wird wohl als April-Scherz in die Apple-Historie eingehen, denn ein 140K-Laufwerk paßt genausowenig zu ProDOS wie eine Maus zum Elefanten. Für Nur-Besitzer von 140K-Laufwerken dürfte Diversi-DOS nach wie vor das Mittel der Wahl sein. Wenngleich ProDOS speziell für reine Assembler-Programmierer auch bei 140K-Drives erweiterte Möglichkeiten bietet, ist der Einsatz von ProDOS in der Regel nur dann sinnvoll, wenn *neben* den normalen Drives *zusätzlich* Festplattenlaufwerke eingesetzt werden, beispielsweise wenn eine Firma in der einen Abteilung einen Apple IIe mit Profile und in einer weiteren Abteilung einen Apple IIe mit 140K-Laufwerk installiert hat.

1.2.4. Programm-Kompatibilität

Mit dem ProDOS-Betriebssystem in Form der ,,User's Disk'' erhält der Käufer zusätzlich zwei wichtige Programme, nämlich FILER und CONVERT. Während der FILER zum Formatieren von Leerdisketten sowie zum Kopieren ganzer Disketten oder einzelner Dateien dient, lassen sich mit dem CONVERT-Programm DOS 3.3 Dateien auf ProDOS-Disketten überspielen und umgekehrt. Textfiles und Hires-Bilder sind problemlos konvertierbar. Als problematisch erweisen sich lediglich nicht-vorformatierte Random-Files (siehe ,,DOS-Buch'', S. 58 ff.). Ferner lassen sich mit CONVERT auch Applesoft- und Assemblerprogramme überspielen, doch stellt man dann in der Regel fest, daß die konvertierten Programme nicht mehr lauffähig sind. Im einzelnen gilt:

a) ,,Lupenreine'' Applesoft-Programme – ohne Peeks, Pokes und Calls – sind unter ProDOS nach gewissen syntaktischen Änderungen, die später im einzelnen geschildert werden, lauffähig. Der Änderungsaufwand hält sich also in Grenzen.

b) Mit TASC usw. compilierte Applesoft-Programme sind nicht mehr lauffähig, und zwar unter anderem wegen der unter ProDOS geänderten Garbage Collection. Auch hierauf wird später im einzelnen technisch eingegangen.

c) Reine Assemblerprogramme oder gemischte Applesoft-Assembler- bzw. compilierte Applesoft-Assembler-Programme sind nicht mehr lauffähig, da selbstverständlich unter ProDOS die ganzen Systemadressen anders sind. Der Änderungsaufwand ist enorm und impliziert in der Regel ein völliges Umschreiben.

Sehr wichtig ist auch die Erkenntnis, das viele unter DOS 3.3 entwickelte Programme allein deshalb nicht mehr lauffähig sind, weil unter ProDOS 16K weniger an Speicherraum zur Verfügung steht. Viele Utilities und Anwenderprogramme haben beispielsweise die Language Card benutzt, sei es um DOS 3.3 selbst in die Karte zu schieben, sei es um Assemblerroutinen, Daten, Hires-Bilder usw. in die Karte zu verlagern. Unter ProDOS hat man theoretisch nur dann denselben freien Speicherraum, wenn man ausschließlich mit dem in der Karte befindlichen ,,PRODOS'' auskommt und somit auf das ,,BASIC.SYSTEM'' völlig verzichtet. Dies ist jedoch nur bei lupenreinen Assemblerprogrammen möglich. Einige unabdingbare Teile des ,,BASIC.SYSTEM'', z.B. der CATALOG-Befehl, müssen dann selbst programmiert werden, weil ,,PRODOS'' diesen Befehl nicht enthält.

1.3. Interne Speicherorganisation

Das im Einband dieses Buches abgebildete Diagramm veranschaulicht die interne Speicherorganisation unter dem Betriebssystem mit PRODOS, BASIC.SYSTEM und RAM-Disk-Driver für 64K-Karte.

1.3.1. Boot-Vorgang

Wenn ProDOS von einem Diskettenlaufwerk gebootet wird (zum Begriff ,,Booten'' siehe ,,DOS-Buch'', S. 6), passiert etwa folgendes:

a) Nach Ctrl-Offener Apfel-Reset springt die Monitor-Routine ab $FAA6 zu dem erstbelegten DOS-Controller-Slot mit der höchsten Slot-Nummer, z.B. bei Slot 6 zur Adresse $C600 und startet die erste Stufe des Urladers, der den Sektor 0 von Spur 0, d.h. die linke Hälfte von Block 0, in den Speicher ab $0800 einliest.

b) Danach springt das Slot-Controller-Programm $C600-C6FF nach $0801 und lädt die rechte Hälfte von Block 0 (Spur 0, Sektor 1) sowie im Anschluß daran den Block 1 (= Spur 0, Sektoren 2-3) ein.

c) Blocks 0 und 1 oder – in DOS-Terminologie – die Sektoren 0-3 von Spur 0 enthalten also die zweite Stufe des Urladers, der seinerseits entweder bei einem Apple III nunmehr das SOS-Betriebssystem oder bei einem Apple IIe das PRODOS-Betriebssystem in den Speicher ab $2000 einlädt. Dieses Betriebssystem, dem ein Move-Programm vorangestellt ist, verschiebt den eigentlichen PRODOS-Teil in die Bank 1 der Language Card ab $D000, installiert ein Reboot-Programm in der Bank 2 der Language Card ab $D100, legt den RAM-Disk-Driver auf eine möglicherweise vorhandene 64K-Karte, und zwar dort ab $0200, und konfiguriert die PRODOS Global Page $BF00-$BFFF.

d) Danach sucht PRODOS in dem Disketten-Inhaltsverzeichnis das erstgenannte System-Programm mit dem Zusatz ,,SYSTEM'' – in der Regel also das BASIC.SYSTEM – und lädt dieses ebenfalls zunächst ab $2000 in den Speicher, um es im Anschluß daran in den eigentlichen Bereich – beim BASIC.SYSTEM ab $9A00 – zu verschieben.

e) Das BASIC.SYSTEM sucht nunmehr im Disketten-Inhaltsverzeichnis ein Hello-Programm namens ,,STARTUP'', das sowohl ein Applesoft- als auch ein Maschinenprogramm sein kann, und startet es – falls vorhanden. Andernfalls springt das

BASIC.SYSTEM in den Applesoft-Modus.

Selbst dann, wenn das STARTUP-Programm nicht benötigt wird, empfiehlt es sich, ein kurzes Applesoft-Hello-Programm unter diesem Namen anzulegen, da sonst die Zero-Page nicht korrekt initialisiert wird.

Abschreckendes Beispiel:

ProDOS-Diskette ohne STARTUP-Programm booten, dann CALL -151, dann FC58G (= HOME) läßt ProDOS ,,ausflippen'', weil Speicherstelle $0048 nicht initialisiert wurde!

Das STARTUP-Programm muß sich auf der Boot-Diskette befinden, wobei zu beachten ist, daß das Profile-Festplattenlaufwerk kein STARTUP-Volume sein kann, d.h. ein Apple IIe, an den *nur* die Profile angeschlossen ist, ist nicht bootfähig. Findet das BASIC.SYSTEM auf der Boot-Diskette kein STARTUP-Programm, dann gibt es die Suche auf. Anders liegen die Dinge, wenn sich das Betriebssystem nach der Beendigung des Bootvorgangs komplett im Speicher befindet. Dann kann mit

— PRODOS oder
— BASIC.SYSTEM

von jedem Laufwerk einschließlich Profile, das mit PREFIX/VOLUME.NAME oder CATALOG/VOLUME.NAME zuletzt angesprochen wurde, neu gebootet werden. Der Strich-Befehl (-PRODOS) entspricht dann einem Mittelding zwischen Kalt-Booten und Warm-Booten. Er ist Kalt-Booten insofern, als das Betriebssystem gänzlich neu installiert wird, und Warm-Booten insofern, als der Urlader hierzu nicht mehr verwendet wird. Daraus folgt, daß der Strich-Befehl ,,-PRODOS'' nur dann mit Erfolg ausgeführt werden kann, wenn sich PRODOS und BASIC.SYSTEM noch intakt im Speicher befinden. Mit ,,-PRODOS'' werden sowohl PRODOS als auch BASIC.SYSTEM neu installiert, mit ,,-BASIC.SYSTEM'' dagegen wird das BASIC.SYSTEM *allein* neu eingelesen. Der Strich-Befehl hat den Vorteil, daß man ein anderes externes Laufwerk, z.B. Drive 2, oder die Profile (z.B. Slot 6, Drive 2 oder Slot 7, falls sich dort die Profile befindet) aktivieren kann. Das Kalt-Booten ist also nur von Drive 1, das Warm-Booten auch von Drive 2 möglich.

Man beachte, daß ,,-BASIC.SYSTEM'' den Inhalt der ggf. vorhandenen 64K-RAM-Disk nicht zerstört, wohingegen mit ,,-PRODOS'' oder mit PR#6 die RAM-Disk stets neu initialisiert wird. Dieser Sachverhalt ist von der Firma Apple nicht voll durchdacht worden, denn ein gewollter oder auch ungewollter Kaltstart kommt in der

Praxis vergleichsweise häufig vor. Damit ist die Verwendung der RAM-Disk nur als Scratch-Disk zu empfehlen.

Wenn sich das Betriebssystem korrekt im Speicher befindet, wird nach einem noch nicht bekannten Volume-Namen (z.B. CATALOG/VOLUME/NAME.XYZ) lt. ,,Technical Mannual'' in der nachstehenden Reihenfolge gesucht (= Volume Search Order). In Wahrheit scheint jedoch die Suche stets mit der niedrigsten Slot-Nummer zu beginnen.

```
Slot 6,  Drive 1  (normales Disk-Drive angeschlossen)
Slot 6,  Drive 2
Slot 7            (Profile angeschlossen)
Slot 5,  Drive 1  (normales Disk-Drive angeschlossen)
Slot 5,  Drive 2
Slot 4,  Drive 1  (normales Disk-Drive angeschlossen)
Slot 4,  Drive 2  usw.
```

Wenn man den Boot-Prozeß nachvollziehen will, dann verfahre man wie folgt (Die Angaben beziehen sich auf Slot 6 bei normalem Slot-Controller-Programm):

Spur 0, Sektor 0 ab $0801 einlesen:

```
CALL -151
1600<C600.C6FFM
16F8:0
1600G
```

Spur 0, Sektor 0-1 ab $0801 einlesen:

```
CALL -151
1600<C600.C6FFM
16F8: A9 00 8D 31 08 4C 01 08
1600G
```

Urlader Stufe 2 ab $0800 und PRODOS ab $2000 einlesen:

```
CALL -151
1600<C600.C6FFM
16F8: A9 00 8D FC 08 4C 01 08
1600G
```

1.3.1.1. Disk-Controller-Erkennungsbytes

Laut Firma Apple soll ein Disk-Controller-Programm folgende Erkennungsbytes enthalten:

$CX01: $20
$CX03: $00
$CX05: $03
$CX07: $3C

Anhand dieser vier Bytes erkennt der Monitor, daß ein Disk-Controller vorliegt. Diese Werte gelten auch für die bisherigen DOS-Controller. X steht für die Slot-Nummer, d.h. bei Slot 6 bedeutet $CX01 = $C601 usw. (Der ProDOS-Urlader selbst prüft $C65E-$C6EA bei Slot 6, so daß Fremd-Controller nicht booten.)

$CXFC: LL HH

Diese Stelle soll – Low Byte first – die Gesamtanzahl der Blocks einer Diskette enthalten.

$CXFE: 76543210

Das Bit-Muster in dieser Speicherstelle soll den Status charakterisieren, und zwar wie folgt:

7: Datenträger entfernbar (z.B. Diskette, Gegensatz Festplatte)
6: Interrupt möglich
5: Laufwerkanzahl (in Verbindung mit Bit 4)
4: Laufwerkanzahl (z.B. 5:10 bei 2 Drives)
3: Formatierung möglich
2: Beschreibbarkeit
1: Lesbarkeit
0: Status lesbar

$CXFF: LL

Diese Speicherstelle soll das niederwertige Byte der Controller-Einsprung-Adresse enthalten, also $00 bei Einsprung $C600.

Es sei darauf hingeweisen, daß die bisherigen Apple-Controller – mit Ausnahme der erstgenannten 4 Bytes – *nicht* die charakterisierten Erkennungsbytes aufweisen, da dieser ROM-Bereich bislang nur Nullen enthielt und insoweit unbenutzt blieb.

1.3.1.2. Freier Speicher nach Booten

Bisweilen ist es nützlich zu wissen, welche Speicherbereiche durch das Booten nicht zerstört werden. Wenn PRODOS und BASIC.SYSTEM gebootet werden und im Anschluß daran kein STARTUP-Programm geladen wird, dann werden folgende Speicherbereiche *nicht* verändert:

$0100 – $01D7
$0200 – $027F
$0291 – $02FF
$0356 – $036B
$03D6 – $03EF
$0A86 – $0BFF
$1800 – $1DFF
$5C00 – $95FF (ohne BASIC.SYSTEM $5C00-$BEFF)
$9600 – $99FF (externer Puffer)
$BD00 – $BDFF (interner Puffer)

Wenn man beispielsweise unter DOS 3.3 ein Assemblerprogramm ab $6000-$9000 einlädt, bleibt es auch nach dem Booten von ProDOS im Speicher erhalten. Deshalb sind einige Programmbeispiele in diesem Buch mit ORG $6000 assembliert.

1.3.2. Zero-Page

Das eigentliche PRODOS in der Language Card benutzt nur wenige Zero-Page-Adressen, nämlich insbesondere für das MLI

$3A-$3F
$40-$4F

sowie für das Reboot-Programm

$00-$03.

Laut ,,Technical Manual'', S. 28, werden die Speicherstellen und $40-$4E (gemeint ist wohl $4F!) von PRODOS vorübergehend zwischengespeichert und nach Beendigung der Routine wieder zurückgepokt. Diese Angaben sind falsch! Das nachfolgende Testprogramm ,,Zero-Page-Test'', das einen beliebigen Diskettenblock einliest, zeigt, daß mit Ausnahme der Monitor-Speicherstelle $44 für das Processor Status Register die anderen Zero-Page-Adressen, insbesondere $3A, $3E, $40-$46 (außer $44) sowie $4F verändert werden.

Die Speicherstellen $42-$47 haben für den Disk-Driver folgende Bedeutung:

$42: Befehl, z.B. $01 = Lesen

0 = Seek (Suchen)
1 = Lesen
2 = Schreiben
3 = Formatieren (wird nicht unterstützt!)

$43: Geräte-Nummer (UNITNUM), z.B. $60 = S6, D1, als folgendes Bit-Muster:

76543210
01100000 = $60

Bit 7: 0 = Drive 1
1 = Drive 2
Bit 6-4: Slot-Nummer (z.B. 110 = Slot 6)
Bit 3-0: nicht benutzt (0000)

$44-$45: LL HH = 512 Byte-Puffer-Zeiger, z.B. 0070 = $7000

$46-$47: LL HH = Block-Nummer, z.B. AA00 = $00AA = Block 170

Der Disk-Driver läßt sich direkt mit JSR $F800 ansprechen, wenn man $42-$47 zuvor entsprechend initialisiert und die LC einschaltet. Nach JSR $F800 enthält der Akkumulator die Fehlernummer, falls das Carry-Flag gesetzt ist.

```
                        1               ORG   $6000
                        2    *
                        3    * Zero-Page-Test
                        4    * ==============
                        5    *
                        6    * U.Stiehl/30.04.84
                        7    *
6000: 4C 07 60          8               JMP   ZERO1
                        9    *
6003: 00               10    BLOCKLL   HEX   00          ;Low
6004: 00               11    BLOCKHH   HEX   00          ;High
6005: 01               12    DELETE    HEX   01          ;Löschen
6006: 00               13    ERROR     HEX   00          ;1=Error
                       14    *
6007: AD 03 60         15    ZERO1     LDA   BLOCKLL
600A: 8D 68 60         16              STA   BLOCKL
600D: AD 04 60         17              LDA   BLOCKHH
6010: 8D 69 60         18              STA   BLOCKH
6013: A9 00            19              LDA   #0
6015: 8D 06 60         20              STA   ERROR
                       21    *
                       22    * Zero-Page vor  MLI nach $8000
                       23    * Zero-Page nach MLI nach $8100
                       24    * Zero-Blank vor MLI nach $8200
                       25    *
6018: A2 00            26              LDX   #$00
601A: B5 00            27    ZERO2     LDA   $00,X
601C: 9D 00 80         28              STA   $8000,X
601F: AD 05 60         29              LDA   DELETE
6022: 95 00            30              STA   $00,X
6024: 9D 00 82         31              STA   $8200,X
6027: E8               32              INX
6028: D0 F0            33              BNE   ZERO2
                       34    *
602A: 20 00 BF         35    RDBLOCK   JSR   $BF00       ;MLI
602D: 80               36              HEX   80          ;READ
602E: 64 60            37              DA    COUNT
                       38    *
6030: F0 03            39              BEQ   ZERO3
6032: EE 06 60         40              INC   ERROR
6035: A2 00            41    ZERO3     LDX   #$00
6037: B5 00            42    ZERO4     LDA   $00,X
6039: 9D 00 81         43              STA   $8100,X
603C: BD 00 80         44              LDA   $8000,X
603F: 95 00            45              STA   $00,X
6041: E8               46              INX
6042: D0 F3            47              BNE   ZERO4
                       48    *
6044: BD 00 81         49    DISPLAY1  LDA   $8100,X
6047: DD 00 82         50              CMP   $8200,X
604A: F0 14            51              BEQ   DISPLAY2
604C: 8A               52              TXA
604D: 20 DA FD         53              JSR   $FDDA       ;HEXOUT
```

```
6050: A9 BA     55            LDA  #":"
6052: 20 ED FD  56            JSR  $FDED      ;PRINT
6055: BD 00 81  57            LDA  $8100,X
6058: 20 DA FD  58            JSR  $FDDA
605B: A9 8D     59            LDA  #$8D
605D: 20 ED FD  60            JSR  $FDED
6060: E8        61   DISPLAY2 INX
6061: D0 E1     62            BNE  DISPLAY1
6063: 60        63            RTS
                64   *
6064: 03        65   COUNT    HEX  03
                66   *
6065: 60        67   UNITNUM  HEX  60         ;S6,D1
6066: 00 70     68   BUFFER   HEX  0070       ;$7000
6068: 01        69   BLOCKL   HEX  01         ;Block 1
6069: 00        70   BLOCKH   HEX  00

--End assembly--

106 bytes

Errors: 0
```

Hätten Sie's gewußt?

Befindet sich HIMEM nicht auf einer Seitengrenze, z.B. HIMEM: 30000, dann stürzt das BASIC.SYSTEM zusammen, wenn man einen Textfile zu speichern versucht. (Grund: Garbage Collection „dreht durch".)

Wenn jedoch neben PRODOS auch noch das BASIC.SYSTEM gestartet worden ist, sind auf der Zero-Page nur noch wenige Speicherstellen frei, und zwar im einzelnen:

```
$06-$07
$08         (bei Apple IIe nicht frei)
$09
$19-$1E
$1F         (bei Apple IIe nicht frei)
$CE-$CF
$D6-$D7
$E3
$EB-$EF
$FA-$FE
```

Es sei darauf hingewiesen, daß diese wenigen freien Zero-Page-Stellen dieselben wie unter DOS 3.3 sind.

1.3.3. Eingabe-Puffer

Unter ProDOS ist der Eingabe-Puffer (Tastatureingabepuffer) $0200-$02FF nicht mehr frei. In der zweiten Hälfte ab $0280 wird der Dateiname und ab $0200 die errechnete Anzahl der freien Diskettenblocks abgelegt. Dies heißt nicht, daß der Eingabe-Puffer jetzt nicht mehr von der GETLN-Routine benutzt werden könnte. Doch ist es nicht mehr möglich, Hilfsvariablen usw. vorübergehend etwa in der oberen Hälfte des Puffers zu speichern, da diese Daten dann bei einem Diskettenzugriff zerstört würden. Sinngemäß sind auch alle diejenigen Programme nicht mehr lauffähig, bei denen das Startprogramm in den Eingabe-Puffer eingelesen wurde.

1.3.4. ProDOS-Vektoren ab $03D0

Die alte DOS-Vektortabelle ab $03D0 wird unter ProDOS nur noch teilweise belegt. Insbesondere sind die ehemaligen DOS-RWTS- und Connect-Adressen verschwunden. An die Stelle der RWTS ist das MLI getreten, das jedoch nicht mehr über die Vektortabelle erreicht werden kann.

```
$03D0: 4C  00  BE   JMP  $BE00  ProDOS Warmstart
$03D3: 4C  00  BE   JMP  $BE00  ProDOS Warmstart
```

($03D6-$03EF frei; $03ED-$03EE: XFER-JMP-Adresse bei Apple IIe; wird u.a. vom RAM-Disk-Driver benutzt)

$03F0:	59	FA		JMP	$FA59	Break-Adresse (Monitor)
$03F2:	00	BE	1B	JMP	$BE00	Reset-Vektor (ProDOS Warmstart)
$03F5:	4C	03	BE	JMP	$BE03	Ampersand-Adresse (Command-Handler)
$03F8:	4C	00	BE	JMP	$BE00	Ctrl-Y (Monitor; Warmstart)
$03FB:	4C	59	FF	JMP	$FF59	Nicht-maskierbarer Interrupt (alter Reset)
$03FE:	EB	BF		JMP	$BFEB	Interrupt-Entry (ProDOS)

Anmerkungen:

a) Der ProDOS-Warmstart ist ein Mittelding zwischem dem DOS 3.3 Kalt- und Warmstart. Beim DOS-Kaltstart (3D3G) wurde das Applesoft-Programm einschließlich der Variablen gelöscht. Beim DOS-Warmstart (3D0G) blieben Programm *und* Variablen erhalten. Beim ProDOS-Warmstart bleibt zwar das Applesoft-Programm unversehrt, doch werden *alle* Variablen zerstört, weil die Applesoft-Interpreter-CLEAR-Routine $D665 aufgerufen wird. Ein Unterschied zwischen Warm- und Kaltstart gibt es nicht mehr. Man beachte, daß auch Reset und Ctrl-Y diesen ,,lauwarmen Kaltstart'' durchführen.

b) Wenn man & + Return allein eingibt, passiert gar nichts, weil ,,Null''-Befehle vom Command-Handler offenbar ignoriert werden. Gibt man dagegen z.B. &CATALOG + Return ein, erfolgt eine Fehlermeldung.

c) Ein indirekter Sprung zu $03FE aufgrund eines Interrupts bewirkt zunächst einen Sprung zur Adresse $BFEB, wo die Bank 1 der Language Card eingeschaltet wird, und dann einen Sprung nach $D13A (laut ,,Technical Manual'' irrtümlicherweise nach $D131). Wenn keine Interrupt-Routine (z.B. Uhr) existiert, dann erfolgt die Meldung ,,INSERT SYSTEM DISK AND RESTART – ERR XX'', wobei ,,XX'' für eine Fehlernummer steht. Bei $D239 wird ein Endlos-Jump nach $D239 ausgeführt.

d) Wenn die Input-Output-Vektoren mit PR#0 : IN#0 oder mit FF59G ,,abgehängt'' wurden, kann man *nicht* mehr den altvertrauten DOS 3.3 CALL 1002 (3EAG) ausführen, sondern muß mit 9A17G oder ersatzweise mit Reset (falls der Reset-Vektor nicht geändert wurde!) das BASIC.SYSTEM wieder ,,anhängen''. $9A17 ist damit die neue Connect-Adresse. Weitere Details finden sich in der Beschreibung der BASIC.SYSTEM Global Page. Wer will, kann mit

03EA: 4C 17 9A

den alten CALL 1002 wieder installieren.

e) Wenn der Reset-Vektor nicht geändert wurde, dann erfolgt bei Ctrl-Reset über $BE00 ein Sprung nach $AC35:

```
AC35: 20  65  D6   JSR  $D665  (Applesoft-Clear)
AC38: 20  17  9A   JSR  $9A17  (Connect)
AC3B: A9  00       LDA  #$00
AC3D: 85  24       STA  $24    (HTAB 1)
AC3F: 4C  3F  D4   JMP  $D43F  (Monitor-0G-Applesoft-Restart)
```

Dem ProDOS-Benutzer wird auffallen, daß die Programmierer der Firma Apple offenbar jedes zusätzlich Return peinlich vermieden haben. So entsprach unter DOS 3.3 der Monitor-0G-Befehl dem JMP $D43C – 3 Bytes vor $D43F –, womit ein zusätzliches Return am Bildschirm ausgegeben wurde. Die „Return-Einsparung" geht sogar soweit, daß erstens nach jedem BASIC.SYSTEM-Befehl (z.B. CATALOG usw.) ein Ctrl-H, d.h. ein Backspace, ausgegeben wird und daß zweitens nach Ctrl-Reset der Cursor zunächst „einfriert", bis die erste Taste gedrückt wird.

1.3.5. BASIC.SYSTEM-Puffer und Garbage Collection

Der Speicherraum von $0800-$95FF ist frei für Applesoft- und Maschinenprogramme. Danach folgt von $9600-$99FF der erste Puffer, der 1K oder 1024 Bytes lang ist. Beim Einlesen von Dateien (Textfiles, Binärfiles usw.) wird in der ersten Hälfte dieses Puffers ($9600-$97FF) der jeweilige 512-Byte-Daten-Block sowie in der zweiten Hälfte ($9800-$99FF) der momentan benötigte 512-Byte-Index-Block abgelegt. Dagegen wird beim CATALOG-Befehl in der ersten Hälfte des Puffers der jeweilige 512-Byte-Catalog-Block zwischengespeichert. Unter dem BASIC.SYSTEM sind die Puffer und demzufolge HIMEM (= Highest Memory = Obergrenze des freien Arbeitspeichers) dynamisch. Beispiel:

a) Nach dem Booten von PRODOS und BASIC.SYSTEM ist HIMEM auf $9600 initialisiert, der erste Puffer beginnt bei $9600 und das BASIC.SYSTEM selbst beginnt bei $9A00.

b) Wird der CATALOG-Befehl ausgeführt, dann ändern sich die obigen Werte nicht. Das gleiche gilt, wenn ein LOAD-, RUN-, BLOAD- oder BRUN-Befehl gegeben wird.

c) Wird jedoch ein Textfile mit OPEN geöffnet, dann wird HIMEM auf $9200 gesetzt und ein zweiter Puffer ab $9200 angelegt. Nach dem Schließen der Datei mit CLOSE wird HIMEM wieder heraufgesetzt und gleichzeitig der Puffer wieder freigegeben. Da die Strings jedoch immer bei HIMEM beginnen, muß das BASIC.SYSTEM selbst eine Garbage Collection (= Entfernung nicht mehr benötigter Strings; z.B. A$ =,,AAA'': A$ = ,,BBB'': danach ist ,,AAA'' durch Garbage Collection entfernbar) vornehmen und vor Beginn von WRITE den String-Pool um $0400 Bytes nach unten und später nach CLOSE wieder um $0400 Bytes noch oben schieben. Dies hat zwei Folgen:

1) Compilierte Applesoft-Programme (TASC, Speedstar, Hayden, Expediter) sind nicht mehr lauffähig, da das BASIC.SYSTEM mit den Compiler-Pointern nichts anfangen kann. Desgleichen ist das Ändern von HIMEM aus dem Programm heraus nicht mehr ohne weiteres möglich (siehe hierzu die Beschreibung der System Bit Map). Genauer gesagt gibt es drei Formen der Garbage Collection: Erstens die des Applesoft-Interpreters, zweitens die des Compilers und drittens die des BASIC.SYSTEM. Ein Compiler läßt nur seine eigene Garbage Collection zu. Das BASIC.SYSTEM läßt neben seiner eigenen auch die des Applesoft-Interpreters zu.

2) Wenn soviele ,,echte'', d.h. auch durch Garbage Collection nicht mehr entfernbare Strings angelegt wurden, daß der freie Speicherraum weniger als 1K beträgt, besteht keine Möglichkeit mehr, diese Strings unter dem BASIC.SYSTEM abzuspeichern, weil kein 1024-Byte-Puffer mehr angelegt werden kann. Hinzu kommt noch, daß das untere Ende des String-Pools eine Seitengrenze (Page Boundary) nicht überschreiten darf. Mithin werden bis zu 1024 + 256 = 1280 Byte als Freiraum benötigt, damit das BASIC.SYSTEM den Textfile gerade noch abspeichern kann. Das folgende Beispiel, das man in exakt der vorliegenden Form eingeben möge, damit dieselbe Programmlänge erreicht wird, soll den Zusammenhang erläutern:

```
100 ONERR GOTO 170
110 DIM A$ (340) : A = 340
120 F1 = 0: F2 = 0: X = 0
130 F1 = FRE (0)
140 FOR X = 1 TO A: A$ (X) = ,,A" +
    ,,AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAA": NEXT
150 F2 = FRE (0)
160 PRINT CHR$ (4) ,,OPEN XXX": PRINT CHR$ (4) ,,WRITE XXX": FOR
    X = 1 TO A: PRINT A$ (X): NEXT: PRINT CHR$ (4) ,,CLOSE"
170 PRINT F1, F2
```

Wenn man dieses Programm bei HIMEM $9600 laufen läßt, erfolgt *nach* der Eröffnung des Textfiles ,,XXX" ein Sprung zur On-Error-Zeile 170 mit der Anzeige:

```
F1 = -30518      F2 = 1018

65536  -  30518  = 35018
35018  -  1018   = 34000
34000  :  100    = 340 Strings (mit je 100 A's)
```

Da F2 weniger als 1024 Bytes beträgt, ist verständlich, daß kein Puffer mehr angelegt werden kann. Ersetzt man jedoch Zeile 110 durch

```
110 DIM A$ (339) : A = 339
```

erfolgt trotzdem erneut eine Fehlermeldung, obgleich nunmehr F2 = 1115 Bytes beträgt. Gleiches gilt für DIM A$ (338), wobei bereits F2 = 1218 Bytes frei sind. Erst bei DIM A$ (337) mit nunmehr F2 = 1321 freien Bytes entsteht keine Fehlermeldung mehr, da jetzt F2 größer als 5 · 256 = 1280 ist.

Das Beispiel zeigt, daß dynamische Puffer erhebliche Unwägsamkeiten in die Applesoft-Programmierung bringen, da man bei Programmen, bei denen der freie String-Pool nicht erheblich mehr als 1280 Bytes beträgt, stets selbst über den String-Pool ,,Buch führen" muß, da sonst der Versuch der Textfile-Speicherung zu einem Programmabbruch führen kann, wenn die besagte Grenze von 1280 Bytes unterschritten wird. Der Applesoft-Programmierer muß deshalb HIMEM gedanklich stets bei $9100 ansetzen, falls das Programm string-intensiv ist.

Da das BASIC.SYSTEM bei unzureichendem Speicherraum *erst* den Textfile eröffnet und *dann* zusammenbricht, wird klar, daß die Systemprogrammierer von der Firma Apple hier nicht genügend nachgedacht haben.

1.3.6. BASIC.SYSTEM Global Page $BE00

Das BASIC.SYSTEM nimmt den Speicherraum $9A00-$BEFF ein, wobei $BE00-$BEFF als Global Page des BASIC.SYSTEM – im Gegensatz zur Global Page des PRODOS – bezeichnet wird. In dieser 256-Byte-Page sind alle wichtigen System-Daten des BASIC.SYSTEM zusammengefaßt. Zwei Bereiche der Global Page sollen nachfolgend anhand von Beispielprogrammen erläutert werden:

1.3.6.1. Command-Handler: BASIC.SYSTEM-Befehle

Ab $BE00 stehen mehrere Sprungadressen, u.a. der Command-Handler und die Print-Error-Routine.

```
$BE00: JMP  $AC35  Warmstart
$BE03: JMP  $A6B4  Erst-Command-Handler
$BE06: JMP  $BE9E  Zweit-Command-Handler (normal RTS)
$BE09: JMP  $9B22  On-Error-Handler
$BE0C: JMP  $9FBF  Print-Error-Routine
```

Der Command-Handler, genauer Erst-Command-Handler, läßt sich aus einem Assemblerprogramm heraus wie folgt benutzen:

a) Zunächst muß der BASIC.SYSTEM-Befehl, z.B. „CATALOG", „BLOAD XXX" o.ä. im Eingabe-Puffer ab $0200 mit Bit 7 on abgelegt werden.

b) Dann muß ein JSR $BE03 = Command-Handler erfolgen, der den Befehl auszuführen versucht.

c) Tritt kein Fehler auf, kehrt die Routine zum Assemblerprogramm zurück. Andernfalls wird – entgegen der inkorrekten Darstellung im „Technical Manual" – nach Anzeige der Fehlermeldung das Programm abgebrochen.

d) Um dies zu vermeiden, muß die Print-Error-Routine in das Assemblerprogramm umgelenkt werden. Wie dies geschieht, ist dem nachfolgenden Programm „BASIC.SYSTEM-Befehle" zu entnehmen. Man beachte, daß bestimmte Befehle nicht mehr zum Assemblerprogramm zurückkehren, z.B. RUN. In diesem Fall würde die

gepatchte Print-Error-Routine bestehen bleiben. Man berücksichtige ferner, daß man das Demo ,,BASIC.SYSTEM-Befehle'' nur über das Tippen der Del-Taste korrekt verläßt. Solange der ,,:'' als Prompt sichtbar ist, befindet man sich noch im Demo-Programm.

1.3.6.2. Input-Output-Vektoren: Outputvektoränderung

Die Vektoradressen für Slot ,,0''-7 sind in der Global Page wie folgt gespeichert:

$BE10:	HEX	F0FD	$FDF0	= COUT1 = Video-Output
$BE12:	HEX	00C1	$C100	= Slot 1 (z.B. Drucker)
$BE14:	HEX	209B	$9B20	= Slot 2 (falls nicht belegt!)
$BE16:	HEX	00C3	$C300	= Slot 3 (80-Z-Karte) usw. bis Slot 7
$BE20:	HEX	1BFD	$FD1B	= KEYIN = Keyboard-Input
$BE22:	HEX	00C1	$C100	= Slot 1 (z.B. Drucker)
$BE24:	HEX	209B	$9B20	= Slot 2 (falls nicht belegt!)
$BE26:	HEX	00C3	$C300	= Slot 3 (80-Z-Karte) usw. bis Slot 7
$BE30:	HEX	F0FD	$FDF0	= Aktiver Output-Vektor (hier Bildschirm)
$BE32:	HEX	1BFD	$FD1B	= Aktiver Input-Vektor (hier Tastatur)
$BE34:	HEX	4BB8	$B84B	= DOS-Output-Abfangvektor (in $36-$37 CSWL)
$BE36:	HEX	4EB8	$B84E	= DOS-Input-Abfangvektor (in $38-$39 KSWL)
$BE38:	HEX	2F9A	$9A2F	= eigentlicher Output-Handler
$BE3A:	HEX	BA9A	$9ABA	= eigentlicher Input-Handler
$B851:	HEX	2F9A	$9A2F	= Restore Output-Handler
$B853:	HEX	BA9A	$9ABA	= Restore Input-Handler

Anmerkung: $9B20 ist der Einsprung zur ,,NO DEVICE CONNECTED''-Fehlermeldung.

COUT läßt sich z.B. so nachvollziehen:

```
$FDED: 6C 36 00   JMP ($0036)

$0036: 4B B8      JMP $B84B

$B84B: 6C 38 BE   JMP ($BE38)

$BE38: 2F 9A      $9A2F = eigentlicher Output-Handler
```

Das Beispielprogramm ,,Outputvektoränderung" veranschaulicht, wie unter ProDOS der Output- und analog der Inputvektor geändert werden kann.

Es gibt folgende ,,Connect"-Kopier-Routinen im BASIC.SYSTEM:

```
$9A00: Kopiert $BE30-$BE33   nach $36-$39
$9AA3: Kopiert $BE34-$BE37   nach $36-$39
$9A17: Kopiert $B851-$B854   nach $BE38-$BE3B
$9A8D: Kopiert $36-$39       nach $BE30-$BE33
```

$9A17 normalisiert die Vektoren und entspricht dem alten DOS 3.3-Connect $03EA. Wenn man eine eigene Vektorroutine dem BASIC.SYSTEM vorschalten will, dann kopiere man am besten die Adressen direkt in den Bereich $BE30-$BE33 und springe dann in die Routine $9A00, die ihrerseits die Werte in die Zero-Page kopiert.

```
                        1               ORG  $300
                        2    *
                        3    * BASIC.SYSTEM-Befehle
                        4    * ====================
                        5    * aus Assemblerprogramm
                        6    *
                        7    * U.Stiehl/25.04.84
                        8    *
                        9    * Wenn man beispielsweise
                        10   * CATALOG,S6,D1 eingibt
                        11   * erfolgt der Catalog aus
                        12   * dem Assemblerprogramm heraus.
                        13   *
                        14   * Wenn man als erstes die Del-
                        15   * Taste tippt, wird die Schleife
                        16   * beendet.
                        17   * Bei Syntax-Fehlern wird die
                        18   * entsprechende Fehlermeldung
                        19   * und die Fehler-Nr. ausgegeben.
                        20   *
                        21   PUFFER   EQU  $0200
                        22   DOSCMD   EQU  $BE03
                        23   PRINTERR EQU  $BE0C
                        24   GETLN    EQU  $FD6A
                        25   HEXOUT   EQU  $FDDA
                        26   PRINT    EQU  $FDED
                        27   *
                        28   * Alten PRINTERR-Vektor in
                        29   * Assemblerprogramm poken.
                        30   *
                        31   * PRINTERR vorher:4C BF 9F JMP!
                        32   * DOSERROR jetzt: 20 BF 9F JSR!
                        33   *
0300: AD 0D BE          34   POKEN    LDA  PRINTERR+1
0303: 8D 37 03          35            STA  DOSERROR+1
0306: AD 0E BE          36            LDA  PRINTERR+2
0309: 8D 38 03          37            STA  DOSERROR+2
                        38   *
030C: A9 36             39            LDA  #<DOSERROR
030E: 8D 0D BE          40            STA  PRINTERR+1
0311: A9 03             41            LDA  #>DOSERROR
0313: 8D 0E BE          42            STA  PRINTERR+2
                        43
                        44   *
                        45   * Prompt + Status-Register
                        46   *
0316: A9 BA             47   BEFEHL1  LDA  #":"
0318: 85 33             48            STA  $33         ;Prompt
                        49   *
031A: A9 00             50            LDA  #0          ;Status
031C: 85 48             51            STA  $48
031E: 48                52            PHA
031F: 28                53            PLP
```

```
                        55   *
                        56   * Nach GETLN befindet sich
                        57   * der ProDOS-Befehl mit Bit 7 on
                        58   * im Puffer ab $0200.
                        59   *
0320: 20 6A FD          60   BEFEHL2  JSR  GETLN
0323: AD 00 02          61            LDA  PUFFER
0326: C9 FF             62            CMP  #$FF        ;DEL
0328: F0 2F             63            BEQ  EXIT
032A: E0 02             64            CPX  #2
032C: 90 F2             65            BCC  BEFEHL2     ;NUR RTN
                        66   *
                        67   * DOSCMD erwartet den Befehl
                        68   * in dieser Form im Puffer
                        69   *
032E: 20 03 BE          70   EXECUTE  JSR  DOSCMD
                        71   *
                        72   * Falls ein ProDOS-Fehler
                        73   * auftrat, enthält nach DOSCMD
                        74   * der Akkumulator die Fehler-Nr.
                        75   * und das Carry ist gesetzt.
                        76   * Die Fehlermeldung wird
                        77   * angezeigt, indem man mit
                        78   * Akkumulator = Fehler-Nr.
                        79   * in die PRINTERR-Routine
                        80   * springt. Da bei $BE0C
                        81   * jedoch $4C = JMP steht,
                        82   * würde man nach der Fehler-
                        83   * meldung nicht mehr zur
                        84   * Assemblerroutine zurück-
                        85   * kehren. Deshalb muß der
                        86   * PRINTERR-Vektor abgeändert
                        87   * werden.
                        88   *
0331: 90 23             89            BCC  ERNEUT
0333: 8D 66 03          90            STA  ERRNUM
                        91   *
                        92   * DOSERROR+1 + DOSERROR+2 gepokt!
                        93   *
0336: 20 0C BE          94   DOSERROR JSR  PRINTERR   ;JSR!!!
                        95   *
0339: A9 C5             96            LDA  #"E"
033B: 20 ED FD          97            JSR  PRINT
033E: A9 D2             98            LDA  #"R"
0340: 20 ED FD          99            JSR  PRINT
0343: 20 ED FD         100            JSR  PRINT
0346: A9 BA            101            LDA  #":"        ;ERR:
0348: 20 ED FD         102            JSR  PRINT
034B: AD 66 03         103            LDA  ERRNUM
034E: 20 DA FD         104            JSR  HEXOUT
0351: A9 8D            105            LDA  #$8D
0353: 20 ED FD         106            JSR  PRINT
                       107   *
```

```
                    109  * Hier bei Bedarf EA EA EA.
                    110  * falls Routine nur einmal
                    111  * durchlaufen werden soll.
                    112  *
0356: 4C 16 03      113  ERNEUT   JMP  BEFEHL1
                    114  *
                    115  * Ursprünglichen PRINTERR-
                    116  * Vektor wieder herstellen.
                    117  *
0359: AD 37 03      118  EXIT      LDA  DOSERROR+1
035C: 8D 0D BE      119           STA  PRINTERR+1
035F: AD 38 03      120           LDA  DOSERROR+2
0362: 8D 0E BE      121           STA  PRINTERR+2
0365: 60            122           RTS
                    123  *
0366: 00            124  ERRNUM    HEX  00

--End assembly--

103 bytes

Errors: 0
```

Hätten Sie's gewußt?

Das BASIC.SYSTEM läßt sich „abhängen" mit
10 POKE 242, 0: PR#0: IN#0
und wieder anhängen mit
20 CALL 39447

```
                        1               ORG  $0300
                        2    *
                        3    * ProDOS-Outputvektoränderung
                        4    * ===========================
                        5    *
                        6    * U.Stiehl/29.04.84
                        7    *
                        8    * Zweck:
                        9    *
                        10   * A) Umwandlung von Ctrl-Zeichen
                        11   *    in druckbare Zeichen
                        12   *
                        13   * B) Linker Rand bei Listings
                        14   *
                        15   * C) Formfeed bei Listings
                        16   *
                        17   * Vor BRUN Drucker oder
                        18   * 80-Zeichenkarte einschalten
                        19   *
                        20   * Vektor normalisieren mit Reset
                        21   *
                        22   DOSWARM  EQU  $03D0
                        23   VEKTOUT  EQU  $BE30
                        24   VEKTIN   EQU  $BE32
                        25   CONNECT  EQU  $9A00
                        26   SCREEN   EQU  $FDF0
                        27   HOME     EQU  $FC58
                        28   *
0300: 4C 09 03          29               JMP  BEGINN      ;768
                        30   *
                        31   * Outputadresse hier poken!
                        32   *
                        33   * Für Bildschirm: $FDF0  SLOT:0
                        34   * Für Drucker   : $C102  SLOT:1
                        35   * Für 80-Z-Karte: $C307  SLOT:3
                        36   *
0303: 20 F0 FD          37   DRUCK    JSR  $FDF0      ;771
0306: 60                38               RTS
                        39   *
                        40   * Linker Rand 0-255
                        41   *
                        42   * 0 = Kein Rand
                        43   *
0307: 03                44   BREITE   DFB  03         ;0-255
                        45   *
                        46   * Zeilen/Seite 0-255
                        47   *
                        48   * 0 = Kein Formfeed
                        49   *
0308: 3C                50   TIEFE    DFB  60         ;0-255
                        51   *
                        52   * Vektor initialisieren
                        53   *
```

```
0309: A9 1E      55   BEGINN   LDA  #<VEKTOR
030B: 8D 30 BE   56            STA  VEKTOUT
030E: A9 03      57            LDA  #>VEKTOR
0310: 8D 31 BE   58            STA  VEKTOUT+1
0313: 20 00 9A   59            JSR  CONNECT
                 60   *
0316: A9 00      61            LDA  #0
0318: 8D 50 03   62            STA  FFCOUNT
031B: 4C D0 03   63            JMP  DOSWARM
                 64   *
                 65   * Print-Routine
                 66   *
031E: 09 80      67   VEKTOR   ORA  #$80
0320: C9 8D      68            CMP  #$8D        ;RETURN
0322: D0 2D      69            BNE  CTRL
0324: 20 03 03   70            JSR  DRUCK
                 71   *
                 72   * Tiefe
                 73   *
0327: EE 50 03   74            INC  FFCOUNT
032A: AD 08 03   75            LDA  TIEFE
032D: F0 0F      76            BEQ  RANDO       ;KEIN FF
032F: CD 50 03   77            CMP  FFCOUNT
0332: B0 0A      78            BCS  RANDO
0334: A9 8C      79            LDA  #$8C        ;FF
0336: 20 03 03   80            JSR  DRUCK
0339: A9 00      81            LDA  #0
033B: 8D 50 03   82            STA  FFCOUNT
                 83   *
                 84   * Breite
                 85   *
033E: 8A         86   RANDO    TXA
033F: 48         87            PHA
0340: AE 07 03   88            LDX  BREITE
0343: F0 08      89            BEQ  RAND2       ;KEIN R.
0345: A9 A0      90   RAND1    LDA  #$A0
0347: 20 03 03   91            JSR  DRUCK
034A: CA         92            DEX
034B: D0 F8      93            BNE  RAND1
034D: 68         94   RAND2    PLA
034E: AA         95            TAX
034F: 60         96            RTS
                 97   *
                 98   * Formfeed-Counter
                 99   *
0350: 00        100   FFCOUNT  HEX  00
                101   *
                102   * Ctrl-Zeichen
                103   *
0351: C9 A0     104   CTRL     CMP  #$A0
0353: B0 AE     105            BCS  DRUCK       ;>=A0
0355: 18        106            CLC
0356: 69 40     107            ADC  #$40        ;80->C0
0358: 4C 03 03  108            JMP  DRUCK
```

1.3.7. PRODOS Global Page $BF00

Die PRODOS Global Page $BF00-$BFFF – auch System Global Page genannt – enthält alle wichtigen System-Daten für das sich in der Language Card befindliche PRODOS. Diese Page wird nachstehend in Form eines detaillierten Assemblerlistings beschrieben.

1.3.7.1. Language Card Softswitches

Das eigentliche PRODOS – im Gegensatz zum BASIC.SYSTEM befindet sich in der Language Card. Die LC umfaßt bekanntlich 16K, wobei zu einem bestimmten Zeitpunkt immer nur 12K mit Hilfe der Softswitches (Lese-Schreib-Schalter) aktiviert werden können:

ROM:	$D000-$FFFF	(Applesoft-Interpreter)
Bank 1:	$D000-$FFFF	(PRODOS)
Bank 2:	$D000-$FFFF	(enthält ab $D100 das Rebootprogramm)

$D000-$FFFF sind insgesamt 12K. Softswitchmäßig ist die LC wie folgt bestückt:

$D000-$DFFF	Bank 1: 4K
$D000-$DFFF	Bank 2: 4K
$E000-$FFFF	Bank 1 und Bank 2 gemeinsam: 8K

Daraus erhellt, daß der Bereich $E000-$FFFF – gleichviel, welche Bank gewählt wird – stets eingeschaltet ist, während der Bereich $D000-$DFFF zweimal vorkommt, wobei niemals beide Bereiche gleichzeitig aktiv sein können. Die nachstehende Softswitch-Tabelle enthält eine Auswahl der möglichen Softswitch-Schalter, die nach zweimaliger „Schaltung" mit z.B. „LDA $C08B LDA $C08B" den gewünschten R(ead) und/oder W(rite)-Zustand (= Lese-Schreib-Zustand) herbeiführen.

Softswitch	Bank 1	Bank 2	ROM
2 X LDA C08B	R/W	–	–
2 X LDA C083	–	R/W	–
2 X LDA C089	W	–	R
2 X LDA C081	–	W	R

```
                        1               ORG  $BF00
                        2    *
                        3    * PRODOS Global Page $BF00
                        4    * ========================
                        5    *
                        6    SYSERR1  =     $D1DA
                        7    SYSTOD1  =     $D1E6
                        8    SYSTOD2  =     $D1E4
                        9    REBOOT1  =     $EECF
                        10   IRQ1     =     $D13A
                        11   IRQ2?    =     $FFD8
                        12   LCTEST1  =     $D000
                        13   LCTEST2  =     $E000
                        14   RDROM    =     $C082
                        15   RDWRBK2  =     $C083
                        16   RDWRBK1  =     $C08B
                        17   *
                        18   * Machine Language Interface MLI
                        19   * ------------------------------
                        20   *
                        21   * Wird mit JSR angesprungen von
                        22   *
                        23   * $BE82 (BASIC.SYSTEM)
                        24   * $D141 (Bank 2 Reboot-Programm)
                        25   * $D1BF
                        26   * $D24F
                        27   * $D26B
                        28   * $D282
                        29   * $D292
                        30   * $D2B7
                        31   * $D2BE
                        32   *
BF00: 4C B7 BF          33   MLI      JMP  MLI1
                        34   *
                        35   * Reboot-Programm
                        36   * ---------------
                        37   *
                        38   * Starten mit LDA $C08B
                        39   *             LDA $C08B
                        40   *             JMP REBOOT
                        41   *
                        42   * Die eigentliche Reboot-Routine
                        43   * wird durch das Move-Programm
                        44   * bei $EECF von $D100ff (Bank 2)
                        45   *          nach $1000ff gemovt,
                        46   * dann wird der Reset-Vektor
                        47   * auf $1000 gesetzt und im
                        48   * Anschluß daran wird nach
                        49   * $1000 gesprungen.
                        50   * Das Reboot-Programm fragt nun
                        51   * nach dem PREFIX und dem
                        52   * SYSTEM-Programm und versucht
                        53   * es dann einzuladen.
```

```
                        55   *
                        56   * Wird mit JMP angesprungen von
                        57   *
                        58   * $D05D (Bank 1)
                        59   *
BF03: 4C CF EE          60   REBOOT   JMP  REBOOT1
                        61   *
                        62   * Uhr-Routine
                        63   * ------------
                        64   *
                        65   * Die eigene Uhr-Routine muß
                        66   * DATUM und UHRZEIT in die
                        67   * entsprechenden Speicher-
                        68   * stellen ab $BF90-$BF93
                        69   * poken.
                        70   *
                        71   * Wird mit JSR angesprungen von
                        72   *
                        73   * $D060 (Bank 1)
                        74   * $D270
                        75   *
                        76   * Statt RTS NOP NOP
                        77   *       JMP Uhr-Routine
                        78   *
BF06: 60                79   UHR       RTS
BF07: EA                80             NOP
BF08: EA                81             NOP
                        82   *
                        83   * System-Fehler-Routine
                        84   * --------------------
                        85   *
                        86   * Diese Routine springt mit
                        87   * Akkumulator = Fehler-Nummer
                        88   * nach $D1AA, entfernt mit
                        89   * PLA PLA die RTS-Adresse
                        90   * vom Stack und führt dann
                        91   * selbst ein RTS aus.
                        92   *
                        93   * Wird mit JSR angesprungen von
                        94   *
                        95   * $D0A4 (Bank 1)
                        96   * $D0D7
                        97   * $D121
                        98   * $D278
                        99   * $DFB4
                       100   * $EB7C
                       101   * $EC1C
                       102   * $EC4D
                       103   *
BF09: 4C DA D1         104   SYSERR   JMP  SYSERR1
                       105   *
                       106   * System-Tod1
                       107   * -----------
```

```
                    109  *
                    110  * Wenn man mit  LDA $C08B
                    111  *               LDA $C08B
                    112  *               LDA SYSERROR
                    113  *               JMP SYSTOD1
                    114  *
                    115  * nach SYSTOD1 springt, wird
                    116  * die folgende Meldung:
                    117  *
                    118  * INSERT SYSTEM DISK AND RESTART
                    119  *
                    120  * -ERR XY ausgegeben.
                    121  *
                    122  * Springt man dagegen nach
                    123  * SYSTOD2, dann wird der Akku
                    124  * mit $00 geladen und die
                    125  * ERR-Nummer unterdrückt.
                    126  *
                    127  * Systod schaltet die 80-Z-Karte
                    128  * ab, pokt die Meldung ein-
                    129  * schließlich der hexadezimalen
                    130  * Fehler-Nummer in die untere
                    131  * Bildschirmzeile und geht bei
                    132  * $D239 JMP $D239
                    133  * in eine Endlosschleife.
                    134  * Mehr macht Systod nicht!
                    135  * Reset bleibt unverändert!
                    136  *
                    137  * Wird mit JSR angesprungen von
                    138  *
                    139  * $D19C (Bank 1)
                    140  * $D40F
                    141  * $D414
                    142  * $E04C
                    143  * $EC71
                    144  *
BF0C: 4C E6 D1      145  SYSTOD   JMP  SYSTOD1
                    146  *
                    147  * System-Error
                    148  * ------------
                    149  *
                    150  * Hier befindet sich nach MLI
                    151  * die hexadezimale Fehlernummer
                    152  *
BF0F: 00            153  SYSERROR HEX  00
                    154  *
                    155  * Disk-Driver-Vektoren
                    156  * --------------------
                    157  * Slot "0"-7, Drive 1
                    158  *
BF10: A2 D0         159  SL0DR1   DA   $D0A2       ;nein!
BF12: A2 D0         160  SL1DR1   DA   $D0A2
BF14: A2 D0         161  SL2DR1   DA   $D0A2
```

```
BF16: A2 D0      163  SL3DR1   DA   $D0A2
BF18: A2 D0      164  SL4DR1   DA   $D0A2
BF1A: A2 D0      165  SL5DR1   DA   $D0A2
BF1C: 00 F8      166  SL6DR1   DA   $F800      ;belegt
BF1E: A2 D0      167  SL7DR1   DA   $D0A2
                 168  *
                 169  * Slot "0"-7, Drive 2
                 170  *
BF20: A2 D0      171  SL0DR2   DA   $D0A2      ;nein!
BF22: A2 D0      172  SL1DR2   DA   $D0A2
BF24: A2 D0      173  SL2DR2   DA   $D0A2
BF26: 00 FF      174  SL3DR2   DA   $FF00      ;64K-RAM
BF28: A2 D0      175  SL4DR2   DA   $D0A2
BF2A: A2 D0      176  SL5DR2   DA   $D0A2
BF2C: 00 F8      177  SL6DR2   DA   $F800      ;belegt
BF2E: A2 D0      178  SL7DR2   DA   $D0A2
                 179  *
                 180  *
                 181  * Last Device
                 182  * -----------
                 183  *
                 184  * zuletzt angesprochenes Drive
                 185  *
BF30: 60         186  LASTDEV  HEX  60          ;S6,D1
                 187  *
                 188  * Device-Anzahl
                 189  * -------------
                 190  *
                 191  * Anzahl der Drives, minus 1!
                 192  * also 00, 01, 02 = 3 DEVNUM
                 193  *
BF31: 02         194  DEVNUM   HEX  02          ;+1=3
                 195  *
                 196  *
                 197  * Device-Bit-Muster
                 198  * -----------------
                 199  *
                 200  * Bit-Muster 76543210
                 201  *            X        Drive
                 202  *            0        Drive 1
                 203  *            1        Drive 2
                 204  *             XXX     Slot
                 205  *             110     Slot 6
                 206  *             011     Slot 3
                 207  *                XXXX Devicetyp
                 208  *                0000 Laufwerk
                 209  *                0100 Profile
                 210  *                1111 RAM-Karte
                 211  *
BF32: E0         212  DEVICE1  DFB  %11100000  ;S6,D2
BF33: 60         213  DEVICE2  DFB  %01100000  ;S6,D1
BF34: BF         214  DEVICE3  DFB  %10111111  ;S3,D2
BF35: 00         215  DEVICE4  DFB  $00
```

```
BF36: 00          217  DEVICE5  DFB  $00
BF37: 00          218  DEVICE6  DFB  $00
BF38: 00          219  DEVICE7  DFB  $00
BF39: 00          220  DEVICE8  DFB  $00
BF3A: 00          221  DEVICE9  DFB  $00
BF3B: 00          222  DEVICE10 DFB  $00
BF3C: 00          223  DEVICE11 DFB  $00
BF3D: 00          224  DEVICE12 DFB  $00
BF3E: 00          225  DEVICE13 DFB  $00
BF3F: 00          226  DEVICE14 DFB  $00
                  227  *
BF40: 43 4F 50    228  COPYR    ASC  'COPR. APPLE,1983'
                  229  *
                  230  * Wohin1?
                  231  * -------
                  232  *
                  233  * Der nachfolgende Sprung ist
                  234  * im "Technical Manual" nicht
                  235  * dokumentiert und wird auch
                  236  * von nirgendwo angesprungen.
                  237  * Bei $FFD8 wird der Akku in
                  238  * INTAREG gespeichert, WAS1?
                  239  * in $45 übertragen und die
                  240  * Bank 1 eingeschaltet, wonach
                  241  * zur Adresse $BFD3 = INTEXIT?
                  242  * gesprungen wird. Das Ein-
                  243  * schalten der Bank 1 zeigt,
                  244  * daß der Sprung wahrscheinlich
                  245  * von Bank 2 kommen soll, also
                  246  * wohl ein Cross-Bank-Jump ist?
                  247  *
BF50: 8D 8B C0    248  WOHIN1?  STA  RDWRBK1
BF53: 4C D8 FF    249           JMP  IRQ2?      ;???
                  250  *
                  251  * Bei $FF9E: STA WAS1?
                  252  * Bei $FFDB: LDA WAS1?
                  253  *
BF56: 00          254  WAS1?    HEX  00
                  255  *
                  256  * Bei $FFB6: STA WAS2?
                  257  * Bei $FFE6: LDA WAS2?
                  258  *
BF57: 00          259  WAS2?    HEX  00
                  260  *
                  261  * System Bit Map 48K $0000-$BFFF
                  262  * -----------------------------
                  263  *
                  264  * Bit 1 = Page belegt
                  265  * Bit 0 = Page frei
                  266  *
                  267  * 76543210
                  268  * 11001111 $0000-$07FF
                  269  *
```

```
                        271  * Dieses Bitmuster besagt,
                        272  * daß Zero-Page und Stack
                        273  * sowie Textpage 1 für
                        274  * ProDOS tabu sind, während
                        275  * der Input-Puffer und Page 3
                        276  * belegt werden können.
                        277  *
BF58: CF                278  P00_07   DFB  %11001111  ;$0000
                        279  *
                        280  * Nachfolgende Pages alle frei
                        281  *
BF59: 00                282  P08_0F   DFB  %00000000  ;$0800
BF5A: 00                283  P10_17   DFB  %00000000  ;$1000
BF5B: 00                284  P18_1F   DFB  %00000000  ;$1800
BF5C: 00                285  P20_27   DFB  %00000000  ;$2000
BF5D: 00                286  P28_2F   DFB  %00000000  ;$2800
BF5E: 00                287  P30_37   DFB  %00000000  ;$3000
BF5F: 00                288  P38_3F   DFB  %00000000  ;$3800
BF60: 00                289  P40_47   DFB  %00000000  ;$4000
BF61: 00                290  P48_4F   DFB  %00000000  ;$4800
BF62: 00                291  P50_57   DFB  %00000000  ;$5000
BF63: 00                292  P58_5F   DFB  %00000000  ;$5800
BF64: 00                293  P60_67   DFB  %00000000  ;$6000
BF65: 00                294  P68_6F   DFB  %00000000  ;$6800
BF66: 00                295  P70_77   DFB  %00000000  ;$7000
BF67: 00                296  P78_7F   DFB  %00000000  ;$7800
BF68: 00                297  P80_87   DFB  %00000000  ;$8000
BF69: 00                298  P88_8F   DFB  %00000000  ;$8800
BF6A: 00                299  P90_97   DFB  %00000000  ;$9000
                        300  *
                        301  * 76543210
                        302  * 00111111 $9800-$9FFF
                        303  *
                        304  * Das BASIC.SYSTEM beginnt
                        305  * bei $9A00, daher ab hier tabu
                        306  *
BF6B: 3F                307  P98_9F   DFB  %00111111  ;$9800
                        308  *
                        309  * Nachfolgende Pages alle tabu
                        310  *
BF6C: FF                311  PA0_A7   DFB  %11111111  ;$A000
BF6D: FF                312  PA8_AF   DFB  %11111111  ;$A800
BF6E: FF                313  PB0_B7   DFB  %11111111  ;$B000
                        314  *
                        315  * 76543210
                        316  * 11000011 $B800-$BFFF
                        317  *
                        318  * Im Bereich $BA00-BDFF
                        319  * befinden sich interne
                        320  * BASIC.SYSTEM-Puffer
                        321  *
BF6F: C3                322  PB8_BF   DFB  %11000011  ;$B800
                        323  *
```

```
                        325  * Puffer-Adressen
                        326  * ---------------
                        327  *
                        328  * Puffer-Adressen von OPEN-Files
                        329  * Maximal 8 Puffer = Maxfiles 8
                        330  *
BF70: 00 00             331  PUFFER1  HEX  0000        ;LLHH
BF72: 00 00             332  PUFFER2  HEX  0000
BF74: 00 00             333  PUFFER3  HEX  0000
BF76: 00 00             334  PUFFER4  HEX  0000
BF78: 00 00             335  PUFFER5  HEX  0000
BF7A: 00 00             336  PUFFER6  HEX  0000
BF7C: 00 00             337  PUFFER7  HEX  0000
BF7E: 00 00             338  PUFFER8  HEX  0000
                        339  *
                        340  * Interrupt-Adressen
                        341  * ------------------
                        342  *
                        343  * Maximal 4 Interrupts möglich
                        344  *
BF80: 00 00             345  INTRUPT1 HEX  0000        ;LLHH
BF82: 00 00             346  INTRUPT2 HEX  0000
BF84: 00 00             347  INTRUPT3 HEX  0000
BF86: 00 00             348  INTRUPT4 HEX  0000
                        349  *
                        350  * Prozessor-Register
                        351  * ------------------
                        352  *
BF88: 00                353  INTAREG  HEX  00
BF89: 00                354  INTXREG  HEX  00
BF8A: 00                355  INTYREG  HEX  00
BF8B: 00                356  INTSREG  HEX  00
BF8C: 00                357  INTPREG  HEX  00
                        358  *
                        359  * LC-Bank1, LC-Bank2, ROM Status
                        360  * -----------------------------
                        361  *
BF8D: 01                362  INTBANK  HEX  01
                        363  *
                        364  * Interrupt-Return-Adresse
                        365  * -----------------------
                        366  *
BF8E: 00 00             367  INTADR   HEX  0000        ;LLHH
                        368  *
                        369  * Datum und Uhrzeit
                        370  * -----------------
                        371  *
                        372  * Werte sind auch ohne Hardware-
                        373  * Uhr durch Poken änderbar
                        374  *
                        375  * Datum als Doppelbyte
                        376  * --------------------
                        377  *
```

```
                 379  * Beachte, daß die Monats-Bits
                 380  * im linken UND rechten Byte
                 381  * enthalten sind.
                 382  *
                 383  * FEDCBA9876543210
                 384  * JJJJJJJMMMMTTTTT Jahr/Monat/Tag
                 385  *
BF90: 00 00      386  DATUM    HEX  0000
                 387  *
                 388  * Uhrzeit als getrennte Bytes
                 389  * --------------------------
                 390  *
                 391  * FEDCBA98 76543210
                 392  * ---sssss --mmmmmm Stunde/Minute
                 393  *
BF92: 00 00      394  UHRZEIT  HEX  0000
                 395  *
                 396  *
                 397  * Level ist das "Niveau"
                 398  * der geöffneten Files und
                 399  * gibt u.a. Auskunft über
                 400  * die Anzahl der OPEN-Files.
                 401  * Spielt eine Rolle bei
                 402  * OPEN, FLUSH und CLOSE
                 403  * Nach CATALOG wird hier
                 404  * beispielsweise 0F abgelegt.
                 405  * Das MLI selbst ändert LEVEL
                 406  * jedoch offenbar nicht, wenn-
                 407  * gleich bei
                 408  * $E152: LDA $BF94 und
                 409  * $E689: CMP $BF94
                 410  * auf diese Speicherstelle
                 411  * zugegriffen wird.
                 412  * LEVEL spielt also wohl nur
                 413  * für das BASIC.SYSTEM eine
                 414  * Rolle.
                 415  *
BF94: 0F         416  LEVEL    HEX  0F
                 417  *
                 418  * Die BACKUP-Stelle zeigt an,
                 419  * ob eine Datei zu duplizieren
                 420  * ist, d.h. nach Änderungen
                 421  * infolge von CREATE, RENAME,
                 422  * CLOSE, WRITE und SET FILE
                 423  * INFO. Bei der Rückkehr vom
                 424  * MLI wird bei $D078 mit
                 425  * LDA #$00 STA $BF95 das
                 426  * BACKUP zurückgesetzt.
                 427  * Bei $E9D9 wird mit
                 428  * LDA $BF95
                 429  * EOR #%00100000 = $20
                 430  *       76543210
                 431  * Bit 5, d.h. das Backup-Bit,
```

```
                433  * geprüft.
                434  * $E9D9 liegt innerhalb der
                435  * SET FILE INFO Routine.
                436  *
BF95: 00        437  BACKUP   HEX  00
                438  *
BF96: EA        439          NOP
BF97: EA        440          NOP
                441  *
                442  * Machine Identification Code
                443  * --------------------------
                444  *
                445  * 76543210
                446  * XX       Apple-Typ
                447  * 00        II
                448  * 01        II+
                449  * 10        IIe
                450  * 11        III Emulation
                451  *   XX     Speicherkapazität
                452  *   01       48K
                453  *   10       64K
                454  *   11      128K
                455  *     XX   nicht benutzt
                456  *     00
                457  *       X  80-Zeichen-Karte
                458  *       1   ja
                459  *       0   nein
                460  *        X Uhr
                461  *        1  ja
                462  *        0  nein
                463  *
                464  * 10110010 = $B2 - Beispiel:
                465  *          = IIe mit 128K, 80Z,
                466  *            ohne Uhr
                467  *
                468  *
BF98: B2        469  MACHID   DFB  %10110010  ;Beisp.
                470  *
                471  * Slot-Bitmuster
                472  * --------------
                473  *
                474  *           76543210
                475  * Beispiel %01001010 = $4A
                476  *
                477  * Slot 6, 3 und 1 enthalten ROM
                478  *
BF99: 4A        479  SLOTBIT  DFB  %01001010  ;Beisp.
                480  *
                481  * PREFIX
                482  * ------
                483  *
                484  * Prefix  =  0 = nicht definiert
                485  * Prefix < > 0 = definiert
```

```
                        487  *
BF9A: 00                488  PREFIX   HEX  00
                        489  *
                        490  * MLI-Aktivzustand
                        491  * ----------------
                        492  *
                        493  * MLI aktiv, falls Bit 7 = 1
                        494  * Durch ROR-Befehl alternierend
                        495  *
BF9B: 55                496  MLIAKTIV DFB  %01010101  ;$55
                        497  *
                        498  * MLI-Return-Adresse
                        499  * ------------------
                        500  *
                        501  * Diese Adresse wird beim
                        502  * Beginn des MLI hier ab-
                        503  * gespeichert und vor der
                        504  * Rückkehr vom MLI auf den
                        505  * Stack geschoben.
                        506  *
BF9C: 88 BE             507  MLIRTS   DA   $BE88      ;Beisp.
                        508  *
                        509  * X- und Y-Register
                        510  * -----------------
                        511  *
BF9E: 07                512  XSAVEMLI HEX  07         ;Beisp.
BF9F: 1D                513  YSAVEMLI HEX  1D         ;Beisp.
                        514  *
                        515  * MLI-Exit
                        516  * --------
                        517  *
                        518  * Rückkehr des MLI von Bank 1
                        519  * von $D09F
                        520  *
                        521  * Für den Test, welche 16K
                        522  * - ROM, LCBANK1, LCBANK2 -
                        523  * vor dem MLI-Sprung aktiv war,
                        524  * gibt es folgende Erkennungs-
                        525  * bytes:
                        526  *
                        527  *          ROM     BANK1   BANK2
                        528  *
                        529  *  $D000   6F      D8      EE
                        530  *  $E000   4C      20
                        531  *
                        532  *          C082    C081    C08B
                        533  *                  C081    C08B
                        534  *
                        535  *          RDROM   RD/WR   RD/WR
                        536  *
BFA0: 4D 00 E0          537  EXIT     EOR  LCTEST2
BFA3: F0 05             538           BEQ  EXIT1
BFA5: 8D 82 C0          539           STA  RDROM
```

```
BFA8: D0 0B     541           BNE  EXIT2
BFAA: AD F5 BF  542  EXIT1    LDA  BANKB2
BFAD: 4D 00 D0  543           EOR  LCTEST1
BFB0: F0 03     544           BEQ  EXIT2
BFB2: AD 83 C0  545           LDA  RDWRBK2
BFB5: 68        546  EXIT2    PLA
BFB6: 40        547           RTI
                548  *
                549  * MLI-Entry
                550  * ---------
                551  *
                552  * Eigentlicher Sprung zum MLI
                553  * auf der Bank 1 der LC $D000
                554  * Bank 1 ist nach Entry
                555  * read-write-enabled.
                556  * Wird nur von MLI = $BF00
                557  * angesprungen.
                558  *
BFB7: 38        559  MLI1     SEC
BFB8: 6E 9B BF  560           ROR  MLIAKTIV
BFBB: AD 00 E0  561           LDA  LCTEST2
BFBE: 8D F4 BF  562           STA  BANKB1
BFC1: AD 00 D0  563           LDA  LCTEST1
BFC4: 8D F5 BF  564           STA  BANKB2
BFC7: AD 8B C0  565           LDA  RDWRBK1
BFCA: AD 8B C0  566           LDA  RDWRBK1
BFCD: 4C 00 D0  567           JMP  LCTEST1
                568  *
                569  * Interrupt Exit
                570  * --------------
                571  *
                572  * wird mit JMP von $D1CB (Bank 1)
                573  * und mit  JSR von $E1F2
                574  * angesprungen.
                575  *
BFD0: AD 8D BF  576  INTEXIT  LDA  INTBANK
                577  *
                578  * Second Exit?
                579  * ------------
                580  *
                581  * Wird mit JMP von $FFE9
                582  * angesprungen.
                583  *
BFD3: F0 0D     584  INTEXIT? BEQ  INTEXIT2
BFD5: 30 08     585           BMI  INTEXIT1
BFD7: 4A        586           LSR
BFD8: 90 0D     587           BCC  INTEXIT3
BFDA: AD 82 C0  588           LDA  RDROM
BFDD: B0 08     589           BCS  INTEXIT3
BFDF: AD 83 C0  590  INTEXIT1 LDA  RDWRBK2
BFE2: A9 01     591  INTEXIT2 LDA  #$01
BFE4: 8D 8D BF  592           STA  INTBANK
BFE7: AD 88 BF  593  INTEXIT3 LDA  INTAREG
```

```
BFEA: 40          595           RTI
                  596  *
                  597  * Interrupt Entry
                  598  * ---------------
                  599  *
                  600  * Wird über Vektor $03FE-$03FF
                  601  * angesprungen.
                  602  *
BFEB: AD 8B C0    603  INTENTRY LDA  RDWRBK1
BFEE: AD 8B C0    604           LDA  RDWRBK1
BFF1: 4C 3A D1    605           JMP  IRQ1
                  606  *
                  607  * Bank-Bytes
                  608  * ----------
                  609  *
                  610  * Anhand dieser Bytes erkennt
                  611  * ProDOS, ob ROM, LC1, oder LC2
                  612  * eingeschaltet waren.
                  613  *
BFF4: 4C          614  BANKB1   HEX  4C
BFF5: 6F          615  BANKB2   HEX  6F
                  616  *
                  617  * System-Tod2
                  618  * -----------
                  619  *
                  620  * System-Tod, Second Entry.
                  621  * Beschreibung siehe oben.
                  622  * Diese Routine wird von
                  623  * nirgendwo angesprungen.
                  624  *
BFF6: AD 8B C0    625  SYSTOD1A LDA  RDWRBK1
BFF9: 4C E4 D1    626           JMP  SYSTOD2
                  627  *
                  628  * ProDOS-Versionen
                  629  * ----------------
                  630  *
                  631  * Z.Zt. bei ProDOS Version 1.01
                  632  * enthalten diese Stellen nur
                  633  * Nullen, d.h. Versionen $00
                  634  *
BFFC: 00          635  EARLIEST HEX  00
BFFD: 00          636  BASICSYS HEX  00
BFFE: 00          637  MINIMUM  HEX  00
BFFF: 00          638  IDENTNO  HEX  00

--End assembly--

256 bytes

Errors: 0
```

1.3.7.2. Language Card Speicherorganisation: MLI, Puffer, Interrupt, Disk-Driver und Reboot-Programm

a) Machine Language Interface (MLI)

Das MLI nimmt den Bereich $D000-$EFFF der Bank 1 der Language Card ein. Das MLI sollte stets über $BF00 angesprungen werden, wenngleich sich der Eingang bei $D000 wohl nicht ändern wird. Genaue interne Kenntnisse des MLI sind weder für Applesoft- noch Assembler-Programmierer erforderlich, da – im Gegensatz zu DOS 3.3 – die Global-Page-Schnittstelle kaum Wünsche offenläßt. Für diejenigen, die sich trotzdem mit den PRODOS-Interna beschäftigen wollen, seien nachstehend die wichtigsten Systemadressen genannt (die indes die Kenntnis des MLI-Kapitels voraussetzen). Zunächst soll jedoch der PRODOS-Command-Handler kurz charakterisiert werden.

Nach dem Eingang ab $D000 holt sich PRODOS über den Stack den Pointer zu dem MLI-Command (Befehlsnummer im Bereich $40-$D3) und speichert ihn in der Zero-Page $40-$41. Gleichzeitig wird die Rücksprungadresse (MLI-JSR + 4) ermittelt und in $BF9C-$BF9D abgelegt. Außerdem wird der „Syserror" $BF0F auf Null gesetzt. Dann wird nach einem komplizierten Umrechnungsverfahren über das X-Register geprüft, ob der vorgegebene MLI-Command in der Befehlsnummer-Tabelle $EF25-$EF44 vorkommt. Wenn nein, wird das MLI über $D0A7 mit Syserror = 1 verlassen. Wenn ja, wird als nächstes anhand der Parameteranzahl-Tabelle $EF45-$EF64 geprüft, ob der Parameter-Count stimmt. Wenn nein, wird das MLI über $D0AB mit Syserror = 4 verlassen. Vorher wird jedoch noch gecheckt, ob der Parameter-Count = 0 ist. Dies gilt für den im „Technical Manual" nicht-dokumentierten Datum-Uhrzeit-Call (MLI-Command $82 bei Parameter-Count $00), der zum Aufruf der Uhr-Routine ab $D060 führt, die lediglich einen JSR $BF06 macht. Liegt weder die Uhr-Routine noch ein falscher Parameter-Count vor, dann wird erneut der MLI-Command aus der Befehlsnummer-Tabelle in den Akkumulator geladen und mit dem ebenfalls nicht-dokumentierten Reboot-Call (MLI-Command $65 bei Parameter-Count $04) verglichen. Wenn Übereinstimmung vorliegt, wird dieser Reboot-Befehl ab $D05D ausgeführt, der lediglich nach $BF03 und von dort nach $EECF springt (siehe hierzu Global Page Assemblerlisting). Andernfalls wird anhand von Bit 7 und 6 des MLI-Commands geprüft, ob der Befehl in die Kategorie $40-$41, $80-$81 oder $C0-$D3 gehört. Im Falle von $40-$41 wird der Alloc-Dealloc-Befehl ab $D054 über $D0F3 ausgeführt. Im Falle von $80-$81 wird der Read/Write-Block-Befehl ab $D066 über $D0B2 weiter bearbeitet. Hingegen wird bei der Kategorie $C0-$D3 über $D071, dann über $D23C – wo wiederum

komplizierte Prüfroutinen anhand der Check-Tabelle $EF8D-$EFA0 stattfinden – bis schließlich über $D27C aufgrund eines indirekten Sprungs, der der Sprungtabelle $EF65-$EF8C entnommen ist, der entsprechende Befehl ausgeführt. Wenn im weiteren Verlauf der Befehlsausführung keine Fehler mehr auftreten, wird das MLI über $D078, wo die Rücksprungadresse auf den Stack geschoben wird, und dann endgültig über JMP $BFA0 (bei $D09F) verlassen.

Es sei an dieser Stelle darauf hingewiesen, daß über den Hüthig Software Service der ,,ProDOS-Roh-Source'' als unkommentierter Assembler-Quellcode auf Diskette erschienen ist. Bei entsprechender Nachfrage soll zu einem späteren Zeitpunkt ggf. auch ein kommentiertes ProDOS-Listing folgen.

Command-Adressen

$40 = $D0F8
$41 = $D124

$65 = $D05D (Reboot)

$80 = $D0B2
$81 = $D0B2 (dito)

$82 = $D060 (Uhr)

$C0 = $D4F7
$C1 = $EB50
$C2 = $EA0B
$C3 = $E9D4
$C4 = $E96F
$C5 = $D42A
$C6 = $D32A
$C7 = $D382
$C8 = $E0B9
$C9 = $E95D
$CA = $E1F7
$CB = $E47C
$CC = $E677
$CD = $E6E8
$CE = $DE1E
$CF = $DE08

$D0 = $E807
$D1 = $E94B
$D2 = $EEA7
$D3 = $EE98

Die Adressen gelten nur für ProDOS Version 1.01, also die ab Januar 1984 erhältliche, geringfügig modifizierte Erstversion. Man beachte ferner, daß man nicht wahllos in diese Adressen hineinspringen kann. Vielmehr dient die Aufstellung lediglich dem Zweck, im Bedarfsfall die gewünschten Stellen schnell aufzufinden, falls bestimmte Befehle gepatcht werden sollen. Im übrigen sollten die MLI-Befehle stets über $BF00 aufgerufen werden.

b) Interrupt

Der Interrupt-Handler liegt im Bereich $D13A-$D1D7, wobei ab $D1CE die 4 möglichen Interrupt-Sprungadressen über $BF80-$BF87 vektormäßig umgelenkt werden. Danach folgt übrigens die ,,Syserr1''-Routine ab $D1DA und danach ab $D1E4 die ,,Systod''-Routine (zu beiden siehe das Global Page Assemblerlisting). Falls eine Thunderclock-Uhr angeschlossen ist, dann ist die Uhr-Interrupt-Routine ab $F142 aktiv.

c) Puffer und Daten innerhalb PRODOS

PRODOS unterhält eine Reihe interner Puffer, Zwischenspeicher, Sprungtabellen und Systemdaten, und zwar im einzelnen:

ab $EF25 = 541 Bytes
ab $F1AC = 1620 Bytes
ab $F996 = 495 Bytes
ab $FEBC = 66 Bytes
ab $FF7F = 28 Bytes

$EF25-$EFA0 enthält die oben geschilderten Prüf- und Sprungtabellen. Ab $F2A0 werden die Online-Volume-Namen zusammengefaßt. Im Bereich $F400-$F5FF wird die jeweils benötigte Volume Bit Map abgelegt. Von $F400 bis $F5FF wird der momentan benötigte Catalog-Block gespeichert. In der Page $FA00-$FAFF befindet sich die Nibble-Übersetzungstabelle.

Der Name ,,HUSTON'' – wohl der Programmierer von ProDOS – steht ab $EFA7. der Text ,,INSERT SYSTEM DISK AND RESTART'' beginnt bei $EFDF.

d) Normaler Disk-Driver: PRODOS.INIT

PRODOS unterhält neben dem gesonderten Driver für die Profile und dem RAM-Disk-Driver den ,,normalen" Disk-Driver. Der für die normalen Apple-Laufwerke gedachte Disk-Driver befindet sich im Bereich $F800-$F995 – dazwischen Puffer – und $FB85-$FEBD. Bevor das MLI $F800 per JSR aufruft, ist bereits folgendes geschehen. In $40-$41 wurden bereits bei $D036 die Adresse der MLI-Parameter-Liste gespeichert. Ferner wurde bei $D066 der Command $80 bzw. $81, der durch ASL und später LSR zu $00 bzw. $01 wurde, um 1 erhöht und in $42 (Command) gespeichert. Das MLI läßt bei den Block-Befehlen nur Read (= 1) und Write (= 2) zu. Daneben sind 0 = Seek und 3 = Format als Parameter des Disk-Drivers – wenn auch nicht über das MLI – denkbar. Des weiteren wurde bei $D0B2 die Unit-Nummer (Geräte-Nummer) nach $43, der Puffer-Pointer (LL-HH) nach $44-$45, die Block-Nummer nach $46-$47 und schließlich das High Byte (HH) des Puffer-Pointers nach $4F übertragen. Gleichzeitig wurde durch JSR $EE89 überprüft, ob die Puffer-Adresse mit der System Bit Map kollidiert. Und schließlich wurde die Driver-Adresse des entsprechenden Slots ab $BF10, also $F800, nach $F0B5-$F0B6 übertragen und dann indirekt nach $F800 gesprungen.

Beim Eingang zum Disk-Driver ist also die Zero-Page wie folgt verändert:
(Ferner benutzt der Disk-Driver $3A-$3F.)

$40 = Low Byte MLI-Parameter-Liste
$41 = High Byte MLI-Parameter-Liste
$42 = Disk-Command (01 = Read, 02 = Write)
$43 = Unit-Nummer (Bit-Muster DSSS0000)
$44 = Low Byte Puffer-Adresse
$45 = High Byte Puffer-Adresse
$46 = Low Byte Block-Nummer
$47 = High Byte Block-Nummer
$4F = High Byte Puffer-Adresse

Zunächst wird geprüft, ob die zulässige Gesamtblockanzahl von 280 ($0118) – genauer 279, da Zählung bei 0 beginnt – überschritten wurden. Danach wird nach einem Umrechnungsverfahren, das auch dem Programm ,,ProDOS Skewing-Tabelle" zugrunde liegt, die Block-Nummer in die entsprechende Spur-Nummer sowie die erste Sektor-Nummer umgerechnet. Man beachte, daß die zwei Sektoren eines Blocks stets auf derselben (!) Spur liegen. Die Spur wird in $FB56 und der jeweilige Sektor in $FB57 zwischengespeichert. Die eigentliche Sektor-Lese-Schreib-Routine ab $F83A wird zweimal aufgerufen, da ja 1 Block=2 Sektoren entspricht.

Der Disk-Driver eignet sich auch für Laufwerke mit mehr als 35 Spuren. Zu diesem Zweck ist zweierlei erforderlich. Erstens muß die Stelle

$F809: CA D0 2A

abgeändert werden in

$F809: 18 90 04

damit die Routine, die die Block-Gesamtzahl von 280 überprüft, übersprungen wird. Zweitens muß eine eigene Formatierungsroutine geschrieben werden, da der ,,FILER" auf der ,,ProDOS User's Disk" nur 35 Spuren initialisiert. Das folgende Programm ,,PRODOS.INIT", von dem in dieser Auflage von ,,ProDOS für Aufsteiger" eine neue Version als Hex-Dump veröffentlicht wird, formatiert Disketten bis zu 80 Spuren und mehr nach einer modifizierten DOS-RWTS-Routine.

Beim Start von ,,PRODOS.INIT" wird nach dem Volume-Namen gefragt, der bis zu 15 Zeichen lang sein kann. ,,PRODOS.INIT" kann sowohl von einer DOS- wie auch von einer ProDOS-Diskette gestartet werden, da dieses Programm in sich völlig geschlossen ist und Zero-Page-Adressen und versteckte Bildschirmadressen nicht verändert. ,,PRODOS.INIT" setzt jedoch 40-Zeichen-Darstellung voraus. Das Programm wird verlassen, indem man die Del-Taste anstelle des Volume-Namens eingibt.

e) RAM-Disk-Driver

Der RAM-Disk-Driver für die 64K-Karte liegt zum einen im Bereich $FF00-FF7E sowie auf der Karte selbst von $0200 bis $03FD. Die 64K-Karte ist nach dem Strich-Befehl ,,-PRODOS" wie folgt initialisiert:

Der Bereich $0000-$01FF ist weitgehend unbenutzt. Ausnahmen: $3D-$43 sowie das obere Ende des Stacks ab etwa $01CA. Wegen dieser Ausnahmen empfiehlt es sich nicht, diesen Bereich zur eigenen Datenspeicherung zu benutzen.
Der Bereich $0200-$03FF enthält den eigentlichen RAM-Disk-Driver.
Der Bereich $0400-$07FF ist die zweite Bildschirmhälfte bei 80-Zeichen-Darstellung. Bei 40 Z/Z ist dieser $0400-Byte-Speicher frei.
Auch der Bereich $0800-$0BFF, d.h. der zweite 40-Zeichen-Bildschirmspeicher, ist frei, weil der Apple IIc $0800-$08FF als Puffer für die V24-Schnittstelle benutzt.

Die Volume Bit Map ist im Bereich $0C00-$0DFF und der Catalog-Block im Bereich $0E00-$0FFF untergebracht.
Insgesamt ist damit der Bereich $0000-$0FFF (= 8 Blocks) belegt, wohingegen der Bereich $1000-$FFFF (= 120 Blocks) für die eigentlichen Files zur Verfügung steht.

Es empfiehlt sich normalerweise nicht, auf die 64K-RAM-Disk mit Block-Read-Write-Routinen zuzugreifen, da bei den Blocks $00-$01 und $04-$07 nur ,,Schrott'' gelesen wird, weil die Umrechnungen und das Bank-Switching hier nicht funktionieren. Lediglich gegen das Block-Read ab Block $80 (dezimal 128) ist der Driver abgesichert.

Mit Hilfe des nachfolgend gelisteten Programms ,,64K RAM-Disk abstellen'' kann die 64K-Karte gänzlich ,,abgehängt'' werden. Sie wird dann frei für eigene Datenspeicherung und andere Verwendungszwecke (siehe hierzu die ,,MMU Memory Management Utilities für die 64K Karte'', die über den Hüthig Software Service erhältlich sind.)

f) Reboot-Programm

Das Reboot-Programm befindet sich in der Bank 2 der LC ab $D100 bis $D3FF. Der restliche Bereich $D000-$D0FF sowie $D400-$DFFF sind zur Zeit noch nicht von PRODOS benutzt, doch könnte dieser Bereich von zukünftigen ProDOS-Versionen belegt werden (laut ,,Technical Manual'' ,,reserved for future use'').

1.3.8. PRODOS-Diskettenblock-Zuordnung

Um PRODOS zu modifizieren (,,zu patchen''), gibt es drei Möglichkeiten:

a) PRODOS wird nach erfolgtem Laden und Verschieben im RAM gepatcht.
b) PRODOS wird als Binärfile nach $2000 geladen, dann gepatcht und schließlich durch JMP $2000 in die Language Card geschoben.
c) PRODOS wird auf der Diskette selbst geändert.

Da der File ,,PRODOS'' auf einer Diskette keinen spur-sektor-mäßig festen Platz einnehmen muß (im Gegensatz zum DOS 3.3, das sich stets auf den ersten 3 Spuren der Diskette befindet), werden nachstehend relative Block-Nummern angegeben. PRODOS Version 1.0 belegt insgesamt 30 Programm-Blocks (von 0-29) ein. Wenn

man auf eine frisch formatierte Diskette mit dem ,,FILER'' PRODOS kopiert, dann nimmt es die Blocks $0007, $0009-$0025 ein (Der Index-Block befindet sich in Block $0008).

Move-Programm

Block 0-4 = $2000-29FF = Verschiebeprogramm

RAM-Disk-Driver

Block 5 = $2A00-2BFF = RAM-Disk-Driver ($0200-03FF 64K-Karte)
Block 6, linke Hälfte = $2C00-2C7E = RAM-Disk-Driver ($FF00-FF7E Bank 1)
Block 6, linke Hälfte = $2C7F-2CFF = nur Nullen

MLI

Block 6, rechte Hälfte = $2D00-2DFF = Beginn MLI ($D000-D0FF Bank 1)
Block 7-21 = $2E00-4BFF = MLI ($D100-EEFF Bank 1)
Block 22, linke Hälfte = $4C00-4CFF = Ende MLI ($EF00-EFFF Bank 1)
Block 22, rechte Hälfte = $4D00-4D05 = Rest MLI ($F000-F005 Bank 1)
Block 22, rechte Hälfte = $4D06-4DFF = nur Nullen

Global Page

Block 23, linke Hälfte = $4E00-4EFF = Global Page für LC-ProDOS ($BF00-BFFF)
Block 23, rechte Hälfte = $4F00-4FFF = Global Page 48K-ProDOS ($BF00-BFFF)

Interrupt

Block 24, linke Hälfte = $5000-5069 = $F142-F1AB (Uhr-Programm)
Block 24, linke Hälfte = $506A-507C = $F1AC-F20E (Daten)
Block 24, linke Hälfte = $507D-509A = nur Nullen
Block 24, linke Hälfte = $509B-50EB = $FF9B-FFEB (Interrupt)
Block 24, linke Hälfte = $50EC-50F9 = nur Nullen
Block 24, linke Hälfte = $50FA-50FF = $FFFA-FFFF (NMI, Reset, IRQ)
Block 24, rechte Hälfte = $5100-51FF = nur Nullen

Disk-Driver

Block 25, linke Hälfte = $5200-5395 = Disk-Driver ($F800-F995)
Block 25, linke Hälfte = $5396-53FF = Disk-Driver Daten ($F996-F9FF)
Block 26, linke Hälfte = $5400-54FF = Nibble-Tabelle ($FA00-FAFF)
Block 26, rechte Hälfte = $5500-5572 = nur Nullen
Block 26, rechte Hälfte = $5573-5584 = Disk-Driver Daten ($FB73-FB84)
Block 26, rechte Hälfte = $5585-55FF = Disk-Driver ($FB85-FBFF)
Block 27 = $5600-57FF = Disk-Driver ($FC00-FDFF)
Block 28, linke Hälfte = $5800-58BD = Disk-Driver ($FE00-FEBD)
Block 28, linke Hälfte = $58BE-58FF = $FEBE-FEFF = nur Nullen

Reboot-Programm

Block 28, rechte Hälfte = $5900-59FF = Beginn Reboot ($D100-D1FF Bank 2)
Block 29 = $5A00-5BFF = Reboot-Programm ($D200-D3FF Bank 2)

1.3.9. Datum-Eingabe für PRODOS

Da nicht jeder über eine Hardware-Uhr verfügt, stellt sich das Problem, wie das Datum eingegeben und insbesondere gepokt werden kann, da das Datum infolge der bitmäßigen Verschlüsselung (im Gegensatz zur Uhrzeit) für Applesoft-Programmierer komplizierte Umrechnung-Pokes erforderlich macht. Deshalb wird das nachfolgende Programm „Datum-Eingabe für PRODOS" vorgestellt, das nicht nur die Richtigkeit der Eingabe (mit Ausnahme des Schaltjahrs) überprüft, sondern auch die entsprechenden Pokes vornimmt.

PRODOS.INIT für DOS 3.3

```
10 PRINT CHR$ (4) "BLOAD
   PRODOS.INIT"
20 POKE 32771, 6:
   REM Slot 1-7
30 POKE 32772, 1:
   REM Drive 1-2
40 POKE 32773, 1:
   REM Volume 1-254
50 POKE 32774, 80:
   REM Spuren 35-80
60 CALL 32768:
   REM ProDOS.Init
```

PRODOS.INIT, A$8000, L$1000
Hex-Dump

```
$8000: 4C 07 80 06 01 01 50 20
$8008: 58 FC A9 C9 20 ED FD A9
$8010: CE 20 ED FD A9 C9 20 ED
$8018: FD A9 D4 20 ED FD A9 A0
$8020: 20 ED FD A9 D3 20 ED FD
$8028: AD 03 80 20 E3 FD A9 AC
$8030: 20 ED FD A9 C4 20 ED FD
$8038: AD 04 80 20 E3 FD A2 00
$8040: BD E3 80 F0 06 20 ED FD
$8048: E8 D0 F5 A9 A0 20 6C FD
$8050: AD 00 02 C9 8D F0 B0 C9
$8058: FF D0 03 4C D0 03 20 B5
$8060: 80 A9 30 8D D7 88 A9 01
$8068: 8D D8 88 AD 03 80 0A 0A
$8070: 0A 0A 8D C9 88 AD 04 80
$8078: 8D CA 88 AD 05 80 8D CB
$8080: 88 8D D6 88 AD 06 80 8D
$8088: FE 87 A9 00 8D CC 88 8D
$8090: CD 88 A9 00 8D D0 88 A9
$8098: 90 8D D1 88 A9 04 8D D4
$80A0: 88 20 81 89 B0 03 4C AC
$80A8: 89 20 BC 80 AD D5 88 20
$80B0: DA FD 4C D0 03 20 C3 80
$80B8: 20 13 89 60 20 D3 80 20
$80C0: DD 88 60 A2 26 A0 00 B5
$80C8: 00 99 69 83 C8 E8 E0 50
$80D0: D0 F5 60 A2 26 A0 00 B9
$80D8: 69 83 95 00 C8 E8 E0 50
$80E0: D0 F5 60 8D 8D 8D 8D CE
$80E8: C1 CD C5 A0 A8 C4 C5 CC
$80F0: A0 BD A0 C5 D8 C9 D4 A9
$80F8: BA 00 D3 D4 C9 C5 C8 CC
$8100: A2 00 A0 02 88 B1 3E 4A
$8108: 3E 00 85 4A 3E 00 85 99
$8110: 00 84 E8 E0 56 90 ED A2
$8118: 00 98 D0 E8 A2 55 BD 00
$8120: 85 29 3F 9D 00 85 CA 10
$8128: F5 60 38 86 27 8E 78 06
$8130: BD 8D C0 BD 8E C0 30 7C
$8138: AD 00 85 85 26 A9 FF 9D
$8140: 8F C0 1D 8C C0 48 68 EA
$8148: A0 04 48 68 20 B9 81 88
$8150: D0 F8 A9 D5 20 B8 81 A9
$8158: AA 20 B8 81 A9 AD 20 B8
$8160: 81 98 A0 56 D0 03 B9 00
$8168: 85 59 FF 84 AA BD 29 83
$8170: A6 27 9D 8D C0 BD 8C C0
$8178: 88 D0 EB A5 26 EA 59 00
$8180: 84 AA BD 29 83 AE 78 06
$8188: 9D 8D C0 BD 8C C0 B9 00
$8190: 84 C8 D0 EA AA BD 29 83
$8198: A6 27 20 BB 81 A9 DE 20
$81A0: B8 81 A9 AA 20 B8 81 A9
$81A8: EB 20 B8 81 A9 FF 20 B8
$81B0: 81 BD 8E C0 BD 8C C0 60
$81B8: 18 48 68 9D 8D C0 1D 8C
$81C0: C0 60 A0 00 A2 56 CA 30
$81C8: FB B9 00 84 5E 00 85 2A
$81D0: 5E 00 85 2A 91 3E C8 C4
$81D8: 26 D0 EB 60 A0 20 88 F0
$81E0: 61 BD 8C C0 10 FB 49 D5
$81E8: D0 F4 EA BD 8C C0 10 FB
$81F0: C9 AA D0 F2 A0 56 BD 8C
$81F8: C0 10 FB C9 AD D0 E7 A9
$8200: 00 88 84 26 BC 8C C0 10
$8208: FB 59 00 83 A4 26 99 00
$8210: 85 D0 EE 84 26 BC 8C C0
$8218: 10 FB 59 00 83 A4 26 99
$8220: 00 84 C8 D0 EE BC 8C C0
$8228: 10 FB D9 00 83 D0 13 BD
$8230: 8C C0 10 FB C9 DE D0 0A
$8238: EA BD 8C C0 10 FB C9 AA
$8240: F0 5C 38 60 A0 FC 84 26
$8248: C8 D0 04 E6 26 F0 F3 BD
$8250: 8C C0 10 FB C9 D5 D0 F0
$8258: EA BD 8C C0 10 FB C9 AA
$8260: D0 F2 A0 03 BD 8C C0 10
$8268: FB C9 96 D0 E7 A9 00 85
$8270: 27 BD 8C C0 10 FB 2A 85
$8278: 26 BD 8C C0 10 FB 25 26
$8280: 99 2C 00 45 27 88 10 E7
$8288: A8 D0 B7 BD 8C C0 10 FB
$8290: C9 DE D0 AE EA BD 8C C0
$8298: 10 FB C9 AA D0 A4 18 60
$82A0: 86 2B 85 2A CD 78 04 F0
$82A8: 53 A9 00 85 26 AD 78 04
$82B0: 85 27 38 E5 2A F0 33 B0
$82B8: 07 49 FF EE 78 04 90 05
$82C0: 69 FE CE 78 04 C5 26 90
$82C8: 02 A5 26 C9 0C B0 01 A8
$82D0: 38 20 EE 82 B9 11 83 20
$82D8: 00 83 A5 27 18 20 F1 82
$82E0: B9 1D 83 20 00 83 E6 26
$82E8: D0 C3 20 00 83 18 AD 78
$82F0: 04 29 03 2A 05 2B AA BD
```

```
$82F8: 80 C0 A6 2B 60 AA A0 A0
$8300: A2 11 CA D0 FD E6 46 D0
$8308: 02 E6 47 38 E9 01 D0 F0
$8310: 60 01 30 28 24 20 1E 1D
$8318: 1C 1C 1C 1C 1C 70 2C 26
$8320: 22 1F 1E 1D 1C 1C 1C 1C
$8328: 1C 96 97 9A 9B 9D 9E 9F
$8330: A6 A7 AB AC AD AE AF B2
$8338: B3 B4 B5 B6 B7 B9 BA BB
$8340: BC BD BE BF CB CD CE CF
$8348: D3 D6 D7 D9 DA DB DC DD
$8350: DE DF E5 E6 E7 E9 EA EB
$8358: EC ED EE EF F2 F3 F4 F5
$8360: F6 F7 F9 FA FB FC FD FE
$8368: FF 00 00 00 00 00 00 00
$8370: 00 00 00 00 00 00 00 00
$8378: 00 00 00 00 00 00 00 00
$8380: 00 00 00 00 00 00 00 00
$8388: 00 00 00 00 00 00 00 00
$8390: 00 00 00 00 00 00 00 01
$8398: 98 99 02 03 9C 04 05 06
$83A0: A0 A1 A2 A3 A4 A5 07 08
$83A8: A8 A9 AA 09 0A 0B 0C 0D
$83B0: B0 B1 0E 0F 10 11 12 13
$83B8: B8 14 15 16 17 18 19 1A
$83C0: C0 C1 C2 C3 C4 C5 C6 C7
$83C8: C8 C9 CA 1B CC 1C 1D 1E
$83D0: D0 D1 D2 1F D4 D5 20 21
$83D8: D8 22 23 24 25 26 27 28
$83E0: E0 E1 E2 E3 E4 29 2A 2B
$83E8: E8 2C 2D 2E 2F 30 31 32
$83F0: F0 F1 33 34 35 36 37 38
$83F8: F8 39 3A 3B 3C 3D 3E 3F
($8400-$8555 Nibble-Puffer)
$8550: 00 00 00 00 00 00 38 BD
$8558: 8D C0 BD 8E C0 30 5E A9
$8560: FF 9D 8F C0 DD 8C C0 48
$8568: 68 20 C3 85 20 C3 85 9D
$8570: 8D C0 DD 8C C0 EA 88 D0
$8578: F0 A9 D5 20 D5 85 A9 AA
$8580: 20 D5 85 A9 96 20 D5 85
$8588: A5 41 20 C4 85 A5 44 20
$8590: C4 85 A5 3F 20 C4 85 A5
$8598: 41 45 44 45 3F 48 4A 05
$85A0: 3E 9D 8D C0 BD 8C C0 68
$85A8: 09 AA 20 D4 85 A9 DE 20
$85B0: D5 85 A9 AA 20 D5 85 A9
$85B8: EB 20 D5 85 18 BD 8E C0
$85C0: BD 8C C0 60 48 4A 05 3E
$85C8: 9D 8D C0 DD 8C C0 68 EA
$85D0: EA EA 09 AA EA EA 48 68
$85D8: 9D 8D C0 DD 8C C0 60 00
$85E0: 00 00 00 00 00 00 00 00
$85E8: 00 00 00 00 00 00 00 00
$85F0: 00 00 00 00 00 00 00 00
$85F8: 00 00 00 00 00 00 00 00
$8600: 84 48 85 49 A0 02 8C F8
$8608: 06 A0 04 8C F8 04 A0 01
$8610: B1 48 AA A0 0F D1 48 F0
$8618: 1B 8A 48 B1 48 AA 68 48
$8620: 91 48 BD 8E C0 A0 08 BD
$8628: 8C C0 DD 8C C0 D0 F6 88
$8630: D0 F8 68 AA BD 8E C0 BD
$8638: 8C C0 A0 08 BD 8C C0 48
$8640: 68 48 68 8E F8 05 DD 8C
$8648: C0 D0 03 88 D0 EE 08 BD
$8650: 89 C0 A0 06 B1 48 99 36
$8658: 00 C8 C0 0A D0 F6 A0 03
$8660: B1 3C 85 47 A0 02 B1 48
$8668: A0 10 D1 48 F0 06 91 48
$8670: 28 A0 00 08 6A 90 05 BD
$8678: 8A C0 B0 03 BD 8B C0 66
$8680: 35 28 08 D0 0B A0 07 20
$8688: 00 83 88 D0 FA AE F8 05
$8690: A0 04 B1 48 20 5A 87 28
$8698: D0 11 A4 47 10 0D A0 12
$86A0: 88 D0 FD E6 46 D0 F7 E6
$86A8: 47 D0 F3 A0 0C B1 48 F0
$86B0: 5A C9 04 F0 58 6A 08 B0
$86B8: 03 20 00 81 A0 30 8C 78
$86C0: 05 AE F8 05 20 44 82 90
$86C8: 24 CE 78 05 10 F3 AD 78
$86D0: 04 48 A9 60 20 95 87 CE
$86D8: F8 06 F0 28 A9 04 8D F8
$86E0: 04 A9 00 20 5A 87 68 20
$86E8: 5A 87 4C BC 86 A4 2E CC
$86F0: 78 04 F0 1C AD 78 04 48
$86F8: 98 20 95 87 68 CE F8 04
$8700: D0 E5 F0 CA 68 A9 40 28
$8708: 4C 48 87 F0 39 4C AF 87
$8710: A0 03 B1 48 48 A5 2F A0
$8718: 0E 91 48 68 F0 08 C5 2F
$8720: F0 04 A9 20 D0 E1 A0 05
$8728: B1 48 A8 B9 B8 88 C5 2D
$8730: D0 97 28 90 1C 20 DC 81
$8738: 08 B0 8E 28 A2 00 86 26
$8740: 20 C2 81 AE F8 05 18 24
$8748: 38 A0 0D 91 48 BD 88 C0
$8750: 60 20 2A 81 90 F0 A9 10
$8758: B0 EE 48 A0 01 B1 3C 6A
$8760: 68 90 08 0A 20 6B 87 4E
$8768: 78 04 60 85 2A 20 8E 87
$8770: B9 78 04 24 35 30 03 B9
$8778: F8 04 8D 78 04 A5 2A 24
$8780: 35 30 05 99 F8 04 10 03
$8788: 99 78 04 4C A0 82 8A 4A
$8790: 4A 4A 4A A8 60 48 A0 02
$8798: B1 48 6A 66 35 20 8E 87
$87A0: 68 0A 24 35 30 05 99 F8
$87A8: 04 10 03 99 78 04 60 A0
$87B0: 03 B1 48 85 41 A9 AA 85
$87B8: 3E A0 56 A9 00 85 44 99
$87C0: FF 84 88 D0 FA 99 00 84
$87C8: 88 D0 FA A9 22 20 95 87
```

```
$87D0: A9 28 85 45 A5 44 20 5A
$87D8: 87 20 0D 88 A9 08 B0 24
$87E0: A9 30 8D 78 05 38 CE 78
$87E8: 05 F0 19 20 44 82 B0 F5
$87F0: A5 2D D0 F1 20 DC 81 B0
$87F8: EC E6 44 A5 44 C9 23 90
$8800: D3 18 90 05 A0 0D 91 48
$8808: 38 BD 88 C0 60 A9 00 85
$8810: 3F A0 80 D0 02 A4 45 20
$8818: 56 85 B0 6B 20 2A 81 B0
$8820: 66 E6 3F A5 3F C9 10 90
$8828: EC A0 0F 84 3F A9 30 8D
$8830: 78 05 99 A8 88 88 10 FA
$8838: A4 45 20 87 88 20 87 88
$8840: 20 87 88 48 68 EA 88 D0
$8848: F1 20 44 82 B0 23 A5 2D
$8850: F0 15 A9 10 C5 45 A5 45
$8858: E9 01 85 45 C9 05 B0 11
$8860: 38 60 20 44 82 B0 05 20
$8868: DC 81 90 1C CE 78 05 D0
$8870: F1 20 44 82 B0 0B A5 2D
$8878: C9 0F D0 05 20 DC 81 90
$8880: 8C CE 78 05 D0 EB 38 60
$8888: A4 2D B9 A8 88 30 DD A9
$8890: FF 99 A8 88 C6 3F 10 CA
$8898: A5 44 D0 0A A5 45 C9 10
$88A0: 90 E5 C6 45 C6 45 18 60
$88A8: FF FF FF FF FF FF FF FF
$88B0: FF FF FF FF FF FF FF FF
$88B8: 00 02 04 06 08 0A 0C 0E
$88C0: 01 03 05 07 09 0B 0D 0F
$88C8: 01 60 01 01 00 00 D9 88
$88D0: 00 90 00 00 01 00 00 60
$88D8: 01 00 01 EF D8 A2 3F A9
$88E0: 07 8D 02 89 20 F1 88 CE
$88E8: 02 89 AD 02 89 C9 04 D0
$88F0: F3 A9 F8 8D 01 89 20 FE
$88F8: 88 A9 78 8D 01 89 A0 07
$8900: B9 78 04 9D 41 89 CA 88
$8908: 10 F6 60 47 4F 57 5F 67
$8910: 6F 77 7F A2 3F A9 07 8D
$8918: 3B 89 20 27 89 CE 3B 89
$8920: AD 3B 89 C9 04 D0 F3 A9
$8928: F8 8D 3A 89 20 34 89 A9
$8930: 78 8D 3A 89 A0 07 BD 41
$8938: 89 99 78 04 CA 88 10 F6
$8940: 60 FF FF FF FF FF FF FF
$8948: FF FF FF FF FF FF FF FF
$8950: FF FF FF FF FF FF FF FF
$8958: FF FF FF FF FF FF FF FF
$8960: FF FF FF FF FF FF FF FF
$8968: FF FF FF FF FF FF FF FF
$8970: FF FF FF FF FF FF FF FF
$8978: FF FF FF FF FF FF FF FF
$8980: FF A9 88 A0 C8 20 00 86
$8988: 60 A2 00 8A 9D 00 90 E8
$8990: D0 FA 60 A9 00 8D AB 89
$8998: AD 06 80 8D AA 89 A2 03
$89A0: 0E AA 89 2E AB 89 CA D0
$89A8: F7 60 00 00 AD CA 88 8D
$89B0: D8 88 AD C9 88 8D D7 88
$89B8: A9 02 8D D4 88 A9 00 8D
$89C0: D0 88 A9 8C 8D D1 88 A9
$89C8: 00 8D CC 88 8D CD 88 20
$89D0: 81 89 B0 21 EE D1 88 EE
$89D8: CD 88 20 81 89 B0 16 EE
$89E0: D1 88 EE CD 88 20 81 89
$89E8: B0 0B EE D1 88 EE CD 88
$89F0: 20 81 89 90 03 4C A9 80
$89F8: A9 00 8D D0 88 A9 90 8D
$8A00: D1 88 20 89 89 A9 03 8D
$8A08: 02 90 A2 00 A0 05 BD 00
$8A10: 02 29 7F C9 0D F0 0C 99
$8A18: 00 90 E0 0E F0 04 E8 C8
$8A20: D0 EC E8 8A 09 F0 8D 04
$8A28: 90 A9 C3 8D 22 90 A9 27
$8A30: 8D 23 90 A9 0D 8D 24 90
$8A38: A9 06 8D 27 90 20 93 89
$8A40: AD AA 89 8D 29 90 AD AB
$8A48: 89 8D 2A 90 A9 00 8D CC
$8A50: 88 A9 04 8D CD 88 20 81
$8A58: 89 B0 40 20 89 89 A9 02
$8A60: 8D 00 90 A9 04 8D 02 90
$8A68: A9 06 8D CD 88 20 81 89
$8A70: B0 29 20 89 89 A9 03 8D
$8A78: 00 90 A9 05 8D 02 90 A9
$8A80: 08 8D CD 88 20 81 89 B0
$8A88: 12 20 89 89 A9 04 8D 00
$8A90: 90 A9 0A 8D CD 88 20 81
$8A98: 89 90 03 4C A9 80 20 89
$8AA0: 89 AE 06 80 CA A9 FF 9D
$8AA8: 00 90 CA D0 FA A9 01 8D
$8AB0: 00 90 A9 0C 8D CD 88 20
$8AB8: 81 89 B0 DF 20 BC 80 4C
$8AC0: 07 80 00 00 00 00 00 00
($8AC8-$8BFF frei)
$8C00: 01 38 B0 03 4C 32 A1 86
$8C08: 43 C9 03 08 8A 29 70 4A
$8C10: 4A 4A 4A 09 C0 85 49 A0
$8C18: FF 84 48 28 C8 B1 48 D0
$8C20: 3A B0 0E A9 03 8D 00 08
$8C28: E6 3D A5 49 48 A9 5B 48
$8C30: 60 85 40 85 48 A0 63 B1
$8C38: 48 99 94 09 C8 C0 EB D0
$8C40: F6 A2 06 BC 1D 09 BD 24
$8C48: 09 99 F2 09 BD 2B 09 9D
$8C50: 7F 0A CA 10 EE A9 09 85
$8C58: 49 A9 86 A0 00 C9 F9 B0
$8C60: 2F 85 48 84 60 84 4A 84
$8C68: 4C 84 4E 84 47 C8 84 42
$8C70: C8 84 46 A9 0C 85 61 85
$8C78: 4B 20 12 09 B0 68 E6 61
$8C80: E6 61 E6 46 A5 46 C9 06
$8C88: 90 EF AD 00 0C 0D 01 0C
```

```
$8C90: D0 6D A9 04 D0 02 A5 4A
$8C98: 18 6D 23 0C A8 90 0D E6
$8CA0: 4B A5 4B 4A B0 06 C9 0A
$8CA8: F0 55 A0 04 84 4A AD 02
$8CB0: 09 29 0F A8 B1 4A D9 02
$8CB8: 09 D0 DB 88 10 F6 29 F0
$8CC0: C9 20 D0 3B A0 10 B1 4A
$8CC8: C9 FF D0 33 C8 B1 4A 85
$8CD0: 46 C8 B1 4A 85 47 A9 00
$8CD8: 85 4A A0 1E 84 4B 84 61
$8CE0: C8 84 4D 20 12 09 B0 17
$8CE8: E6 61 E6 61 A4 4E E6 4E
$8CF0: B1 4A 85 46 B1 4C 85 47
$8CF8: 11 4A D0 E7 4C 00 20 4C
$8D00: 3F 09 26 50 52 4F 44 4F
$8D08: 53 20 20 20 20 20 20 20
$8D10: 20 20 A5 60 85 44 A5 61
$8D18: 85 45 6C 48 00 08 1E 24
$8D20: 3F 45 47 76 F4 D7 D1 B6
$8D28: 4B B4 AC A6 2B 18 60 4C
$8D30: BC 09 A9 9F 48 A9 FF 48
$8D38: A9 01 A2 00 4C 79 F4 20
$8D40: 58 FC A0 1C B9 50 09 99
$8D48: AE 05 88 10 F7 4C 4D 09
$8D50: AA AA AA A0 D5 CE C1 C2
$8D58: CC C5 A0 D4 CF A0 CC CF
$8D60: C1 C4 A0 D0 D2 CF C4 CF
$8D68: D3 A0 AA AA AA A5 53 29
$8D70: 03 2A 05 2B AA BD 80 C0
$8D78: A9 2C A2 11 CA D0 FD E9
$8D80: 01 D0 F7 A6 2B 60 A5 46
$8D88: 29 07 C9 04 29 03 08 0A
$8D90: 28 2A 85 3D A5 47 4A A5
$8D98: 46 6A 4A 4A 85 41 0A 85
$8DA0: 51 A5 45 85 27 A6 2B BD
$8DA8: 89 C0 20 BC 09 E6 27 E6
$8DB0: 3D E6 3D B0 03 20 BC 09
$8DB8: BC 88 C0 60 A5 40 0A 85
$8DC0: 53 A9 00 85 54 A5 53 85
$8DC8: 50 38 E5 51 F0 14 B0 04
$8DD0: E6 53 90 02 C6 53 38 20
$8DD8: 6D 09 A5 50 18 20 6F 09
$8DE0: D0 E3 A0 7F 84 52 08 28
$8DE8: 38 C6 52 F0 CE 18 08 88
$8DF0: F0 F5 BD 8C C0 10 FB 00
$8DF8: 00 00 00 00 00 00 00 00
$8E00: 4C 6E A0 53 4F 53 20 42
$8E08: 4F 4F 54 20 20 31 2E 31
$8E10: 20 0A 53 4F 53 2E 4B 45
$8E18: 52 4E 45 4C 20 20 20 20
$8E20: 20 53 4F 53 20 4B 52 4E
$8E28: 4C 49 2F 4F 20 45 52 52
$8E30: 4F 52 08 00 46 49 4C 45
$8E38: 20 27 53 4F 53 2E 4B 45
$8E40: 52 4E 45 4C 27 20 4E 4F
$8E48: 54 20 46 4F 55 4E 44 25
$8E50: 00 49 4E 56 41 4C 49 44
$8E58: 20 4B 45 52 4E 45 4C 20
$8E60: 46 49 4C 45 3A 00 00 0C
$8E68: 00 1E 0E 1E 04 A4 78 D8
$8E70: A9 77 8D DF FF A2 FB 9A
$8E78: 2C 10 C0 A9 40 8D CA FF
$8E80: A9 07 8D EF FF A2 00 CE
$8E88: EF FF 8E 00 20 AD 00 20
$8E90: D0 F5 A9 01 85 E0 A9 00
$8E98: 85 E1 A9 00 85 85 A9 A2
$8EA0: 85 86 20 BE A1 E6 E0 A9
$8EA8: 00 85 E6 E6 86 E6 86 E6
$8EB0: E6 20 BE A1 A0 02 B1 85
$8EB8: 85 E0 C8 B1 85 85 E1 D0
$8EC0: EA A5 E0 D0 E6 AD 6C A0
$8EC8: 85 E2 AD 6D A0 85 E3 18
$8ED0: A5 E3 69 02 85 E5 38 A5
$8ED8: E2 ED 23 A4 85 E4 A5 E5
$8EE0: E9 00 85 E5 A0 00 B1 E2
$8EE8: 29 0F CD 11 A0 D0 21 A8
$8EF0: B1 E2 D9 11 A0 D0 19 88
$8EF8: D0 F6 A0 00 B1 E2 29 F0
$8F00: C9 20 F0 3E C9 F0 F0 08
$8F08: AE 64 A0 A0 13 4C D4 A1
$8F10: 18 A5 E2 6D 23 A4 85 E2
$8F18: A5 E3 69 00 85 E3 A5 E4
$8F20: C5 E2 A5 E5 E5 E3 B0 BC
$8F28: 18 A5 E4 6D 23 A4 85 E2
$8F30: A5 E5 69 00 85 E3 C6 E6
$8F38: D0 95 AE 4F A0 A0 1B 4C
$8F40: D4 A1 A0 11 B1 E2 85 E0
$8F48: C8 B1 E2 85 E1 AD 66 A0
$8F50: 85 85 AD 67 A0 85 86 20
$8F58: BE A1 AD 68 A0 85 85 AD
$8F60: 69 A0 85 86 AD 00 0C 85
$8F68: E0 AD 00 0D 85 E1 20 BE
$8F70: A1 A2 07 BD 00 1E DD 21
$8F78: A0 F0 08 AE 64 A0 A0 13
$8F80: 4C D4 A1 CA 10 ED A9 00
$8F88: 85 E7 E6 E7 E6 86 E6 86
$8F90: A6 E7 BD 00 0C 85 E0 BD
$8F98: 00 0D 85 E1 A5 E0 D0 04
$8FA0: A5 E1 F0 06 20 BE A1 4C
$8FA8: 8A A1 18 AD 6A A0 6D 08
$8FB0: 1E 85 E8 AD 6B A0 6D 09
$8FB8: 1E 85 E9 6C E8 00 A9 01
$8FC0: 85 87 A5 E0 A6 E1 20 79
$8FC8: F4 B0 01 60 AE 32 A0 A0
$8FD0: 09 4C D4 A1 84 E7 38 A9
$8FD8: 28 E5 E7 4A 18 65 E7 A8
$8FE0: BD 29 A0 99 A7 05 CA 88
$8FE8: C6 E7 D0 F4 AD 40 C0 4C
$8FF0: EF A1 00 00 00 00 00 00
$8FF8: 00 00 00 00 00 00 00 00
(9000-90FF: Puffer)
```

```
                      1               ORG  $6000
                      2    *
                      3    * 64K RAM-Disk abstellen
                      4    * ======================
                      5    *
                      6    * U.Stiehl/11.4.84
                      7    *
                      8    *
6000: AD 83 C0        9               LDA  $C083
6003: AD 83 C0        10              LDA  $C083
                      11   *
                      12   * Alt $FF18: F0 2A BEQ $FF44
                      13   *
6006: AD 18 FF        14              LDA  $FF18
6009: C9 F0           15              CMP  #$F0
600B: D0 1D           16              BNE  NO
600D: AD 19 FF        17              LDA  $FF18+1
6010: C9 2A           18              CMP  #$2A
6012: D0 16           19              BNE  NO
                      20   *
                      21   * Neu $FF18: 4C 3B FF JMP $FF3B
                      22   *                     I/O-Error
                      23   *
6014: A9 4C           24   YES        LDA  #$4C
6016: 8D 18 FF        25              STA  $FF18
6019: A9 3B           26              LDA  #$3B
601B: 8D 19 FF        27              STA  $FF18+1
601E: A9 FF           28              LDA  #$FF
6020: 8D 1A FF        29              STA  $FF18+2
6023: AD 81 C0        30              LDA  $C081
6026: AD 81 C0        31              LDA  $C081
6029: 60              32              RTS
602A: AD 81 C0        33   NO         LDA  $C081
602D: AD 81 C0        34              LDA  $C081
6030: 4C 3A FF        35              JMP  $FF3A      ;BELL

--End assembly--

51 bytes

Errors: 0
```

```
                        1               ORG  $6000
                        2   *
                        3   * Datum-Eingabe für Prodos
                        4   * ========================
                        5   *
                        6   * U.Stiehl/12.4.84
                        7   *
                        8   PROMPT   EQU  $33
                        9   PUFFER   EQU  $0200
                       10   DATE     EQU  $BF90
                       11   PRINT    EQU  $FDED
                       12   GETLN    EQU  $FD6A
                       13   *
6000: A5 33             14           LDA  PROMPT
6002: BD F0 60          15           STA  PROMPT1
6005: A9 A0             16           LDA  #$A0
6007: 85 33             17           STA  PROMPT
6009: A2 00             18   GETLN0  LDX  #0
600B: BD F1 60          19   GETLN1  LDA  PROMPT2,X
600E: F0 06             20           BEQ  GETLN2
6010: 20 ED FD          21           JSR  PRINT
6013: E8                22           INX
6014: D0 F5             23           BNE  GETLN1
6016: 20 6A FD          24   GETLN2  JSR  GETLN
                       25   *
                       26   *                 01234567
                       27   * Größer als 7: TT.MM.JJ ?
                       28   *
6019: CA                29           DEX
601A: E0 07             30           CPX  #7
601C: D0 EB             31           BNE  GETLN0     ;Error!
                       32   *
                       33   * Nur Ziffern und "." ?
                       34   *
601E: BD 00 02          35   GETLN3  LDA  PUFFER,X
6021: 29 7F             36           AND  #$7F
6023: 9D 03 61          37           STA  DATUM,X
6026: C9 2E             38           CMP  #'.'
6028: F0 0D             39           BEQ  GETLN5
602A: C9 30             40           CMP  #'0'
602C: 90 DB             41           BCC  GETLN0     ;Error!
602E: C9 3A             42           CMP  #':'
6030: B0 D7             43           BCS  GETLN0     ;Error!
6032: CA                44   GETLN4  DEX
6033: 10 E9             45           BPL  GETLN3
6035: 30 0A             46           BMI  CHECK
6037: E0 02             47   GETLN5  CPX  #2
6039: F0 F7             48           BEQ  GETLN4
603B: E0 05             49           CPX  #5
603D: F0 F3             50           BEQ  GETLN4
603F: D0 C8             51           BNE  GETLN0     ;Error!
                       52   *
                       53   * Datum korrekt ?
```

```
                        55   *
6041: A2 FF             56   CHECK     LDX  #$FF          ;WRAP!
6043: 20 C0 60          57             JSR  PACKEN
6046: 8D EA 60          58             STA  TT            ;TAG
6049: E8                59             INX                ;1.Punkt
604A: 20 C0 60          60             JSR  PACKEN
604D: 8D EB 60          61             STA  MM            ;MONAT
6050: E8                62             INX                ;2.Punkt
6051: 20 C0 60          63             JSR  PACKEN
6054: 8D EC 60          64             STA  JJ            ;JAHR
                        65   *
                        66   * Jahr okay (0-99) ?
                        67   *
6057: C9 64             68             CMP  #100
6059: 90 02             69             BCC  JAHR
605B: B0 AC             70             BCS  GETLNO
605D: C9 00             71   JAHR      CMP  #0
605F: B0 02             72             BCS  MONAT1
6061: 90 A6             73             BCC  GETLNO
                        74   *
                        75   * Monat okay (1-12) ?
                        76   *
6063: AD EB 60          77   MONAT1    LDA  MM
6066: C9 01             78             CMP  #1
6068: B0 02             79             BCS  MONAT2
606A: 90 9D             80             BCC  GETLNO
606C: C9 0D             81   MONAT2    CMP  #13
606E: 90 02             82             BCC  TAG1
6070: B0 97             83             BCS  GETLNO
                        84   *
                        85   * Tag okay (nach Liste) ?
                        86   *
6072: AA                87   TAG1      TAX
6073: CA                88             DEX
6074: AD EA 60          89             LDA  TT
6077: C9 01             90             CMP  #1
6079: B0 02             91             BCS  TAG2
607B: 90 8C             92             BCC  GETLNO
607D: DD DE 60          93   TAG2      CMP  TAGMON,X
6080: 90 05             94             BCC  POKEN1
6082: F0 03             95             BEQ  POKEN1
6084: 4C 09 60          96             JMP  GETLNO
                        97   *
                        98   * In Prodos Global Page poken
                        99   * --------------------------
                       100   *
                       101   * $BF91     $BF90
                       102   * Poke1     Poke2
                       103   * FEDCBA98 76543210
                       104   * JJJJJJJM MMMTTTTT
                       105   *
6087: AD EC 60         106   POKEN1    LDA  JJ
608A: 0A               107             ASL
```

```
608B: 8D EE 60  109           STA  POKE1
608E: AD EB 60  110           LDA  MM
6091: 0A        111           ASL
6092: 0A        112           ASL
6093: 0A        113           ASL
6094: 0A        114           ASL
6095: 0A        115           ASL
6096: 8D ED 60  116           STA  TEMP
6099: 90 09     117           BCC  POKEN2
609B: 18        118           CLC
609C: AD EE 60  119           LDA  POKE1
609F: 69 01     120           ADC  #1
60A1: 8D EE 60  121           STA  POKE1
60A4: 18        122  POKEN2   CLC
60A5: AD ED 60  123           LDA  TEMP
60A8: 6D EA 60  124           ADC  TT
60AB: 8D EF 60  125           STA  POKE2
                126  *
                127  * "Low Byte first", d.h.
                128  * erst MM/TT, dann JJ/MM
                129  *
60AE: AD EF 60  130           LDA  POKE2
60B1: 8D 90 BF  131           STA  DATE
60B4: AD EE 60  132           LDA  POKE1
60B7: 8D 91 BF  133           STA  DATE+1
                134  *
60BA: AD F0 60  135           LDA  PROMPT1
60BD: 85 33     136           STA  PROMPT
60BF: 60        137           RTS
                138  *
                139  * Packen in 1 Byte
                140  *
60C0: E8        141  PACKEN   INX
60C1: BD 03 61  142           LDA  DATUM,X
60C4: 29 0F     143           AND  #$0F
60C6: 0A        144           ASL                ;MAL 2
60C7: 8D ED 60  145           STA  TEMP
60CA: 0A        146           ASL                ;MAL 2
60CB: 0A        147           ASL                ;MAL 2
60CC: 18        148           CLC
60CD: 6D ED 60  149           ADC  TEMP
60D0: 8D ED 60  150           STA  TEMP
60D3: E8        151           INX
60D4: BD 03 61  152           LDA  DATUM,X
60D7: 29 0F     153           AND  #$0F
60D9: 18        154           CLC
60DA: 6D ED 60  155           ADC  TEMP
60DD: 60        156           RTS
                157  *
                158  * Tage pro Monat maximal
                159  *
60DE: 1F        160  TAGMON   DFB  31           ;JAN
60DF: 1D        161           DFB  29           ;FEB
```

```
60E0: 1F          163             DFB  31          ;MÄRZ
60E1: 1E          164             DFB  30          ;APRIL
60E2: 1F          165             DFB  31          ;MAI
60E3: 1E          166             DFB  30          ;JUNI
60E4: 1F          167             DFB  31          ;JULI
60E5: 1F          168             DFB  31          ;AUGUST
60E6: 1E          169             DFB  30          ;SEPT
60E7: 1F          170             DFB  31          ;OKT
60E8: 1E          171             DFB  30          ;NOV
60E9: 1F          172             DFB  31          ;DEZ
                  173  *
60EA: 00          174  TT         HEX  00          ;TT
60EB: 00          175  MM         HEX  00          ;MM
60EC: 00          176  JJ         HEX  00          ;JJ
60ED: 00          177  TEMP       HEX  00
                  178  *
60EE: 00          179  POKE1      HEX  00          ;JJ+MM
60EF: 00          180  POKE2      HEX  00          ;MM+TT
                  181  *
60F0: 00          182  PROMPT1    HEX  00          ;saven!
60F1: C4 E1 F4    183  PROMPT2    ASC  "Datum (TT.MM.JJ):"
60F4: F5 ED A0 A8 D4 D4 AE CD
60FC: CD AE CA CA A9 BA
6102: 00          184             HEX  00
6103: 54 54 2E    185  DATUM      ASC  'TT.MM.JJ'
6106: 4D 4D 2E 4A 4A

--End assembly--

267 bytes

Errors: 0
```

1.4. Externe Speicherorganisation

Unter externer Speicherorganisation verstehen wir die blockmäßige Strukturierung von Disketten, Festplatten und RAM-Disks unter ProDOS. In diesem Kapitel beschränken wir uns auf die klassischen 35-Track-Disketten, die auch unter DOS 3.3 verwendet werden. Doch gelten unsere Analysen mutatis mutandis auch für Harddisks und RAM-Disks.

1.4.1. Blocks, Sektoren, Tracks

Die klassische Apple-Diskette hat unter ProDOS:

35 ($23) Spuren, numeriert von 0-34 ($00-$22)
16 ($10) Sektoren je Spur, numeriert von 0-15 ($00-$0F)
256 ($100) Bytes je Sektor, numeriert von 0-255 ($00-$FF)

280 ($0118) Blocks je Diskette, numeriert von 0-279 ($0000-$0117)
8 ($08) Blocks je Spur, relativ (!) numeriert von 0-7 ($00-$07)
2 ($02) Sektoren je Block = linke (erste) und rechte (zweite) Hälfte
512 ($200) Bytes je Block, numeriert von 0-511 ($0000-$01FF)

Die ersten 256 Bytes eines Blocks nennen wir linke und die zweiten 256 Bytes rechte Hälfte des Blocks.

Auf eine 35-Track-Diskette passen demnach 280 Blocks · 512 Bytes oder 560 Sektoren · 256 Bytes = 143.360 Bytes = 140K. Auf eine einzelne Spur passen 8 Blocks · 512 Bytes oder 16 Sektoren · 256 Bytes = 4.096 Bytes = 4K.

1.4.1.1. Block-Reader

Da zu ProDOS praktisch noch keine Disketten-Utilities vorliegen, wird nachstehend ein einfaches, aber effizientes „Block-Reader"-Programm vorgestellt, mit dessen Hilfe auch die späteren Block-Dumps erzeugt wurden. Dieses Programm, das mit BRUN BLOCKREADER gestartet wird, initialisiert den Ampersand-&-Vektor. Zuvor schalte man – falls vorhanden – die 80-Zeichen-Karte und/oder den Drucker ein. Mit & + dezimaler Block-Nummer, z.B.

&10

wird der gewünschte Block in den RAM-Speicher ab $1000 eingelesen und in ASCII sowie als Hex-Dump angezeigt. Nach der Anzeige der linken Hälfte des jeweiligen Blocks wartet das Programm auf einen beliebigen Tastendruck, bevor die rechte Hälfte angezeigt wird. Grund: Trotz 80-Zeichen-Darstellung paßt nicht der gesamte Block-Dump auf den Bildschirm.

Das Programm kann durch folgende ,,legale'' Pokes verändert werden:

0806: Slot (z.B. 3 für RAM-Disk, z. Zt. 6)
0807: Drive (z.B. 2 für RAM-Disk, z. Zt. 1)

Als ,,illegale'' Pokes sind möglich:

0808: LL HH Block-Obergrenze (z. Zt. $0117 = 279)
080A: Command (z.Zt. $80 = Read; möglich $81 = Write)

Wenn die Block-Obergrenze erhöht wird, können auch Blocks von Disketten mit mehr als 35 Spuren eingelesen werden, z.B. von Disketten, die mit dem ,,PRODOS.INIT''-Programm initialisiert wurden und bei denen PRODOS selbst entsprechend gepatcht wurde (siehe Kapitel 1.3.7.2). Besitzer von normalen 35-Spur-Laufwerken können testweise mit 36 Tracks operieren, da die 36. Spur bei vielen Laufwerken initialisierbar ist.

Der Write-Command bezieht sich auf den Inhalt von $1000-$11FF, d.h. es sollte zuvor der entsprechende Block eingelesen und dann im Monitor ggf. geändert worden sein. Für die ersten ProDOS-Übungen ist dieser ,,illegale'' Poke jedoch nicht zu empfehlen!

1.4.1.2. Skewing

Die Sektoren sind auf einer Spur physisch von $00-$0F angeordnet, doch verwendet der Disk-Driver von ProDOS das ,,Ascending Skew 2''-Verfahren (ascending = aufsteigend, descending = absteigend, skewing or interleaving = versetztes Anordnen), d.h. die logischen Sektoren $00-$0F haben auf der Diskette einen physischen Zweierabstand und die logischen Blocks $00-$07 einen physischen Einerabstand, wobei die linke Hälfte der rechten vorausgeht. Unter ProDOS sollten die logischen Sektoren einer Spur aufsteigend (0, 1, 2, 3 usw.) und unter DOS absteigend (15, 14, 13, 12 usw.) gelesen werden, damit optimale Zugriffsgeschwindigkeit erzielt wird.

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	physische Sektoren-Folge
00 02 04 06 08 0A 0C 0E 01 03 05 07 09 0B 0D 0F	logische ProDOS-Sektoren-Folge
L0 R0 L1 R1 L2 R2 L3 R3 L4 R4 L5 R5 L6 R6 L7 R7	logische ProDOS-Block-Folge
00 0D 0B 09 07 05 03 01 0E 0C 0A 08 06 04 02 0F	logische DOS-Sektoren-Folge

Aus der obigen Aufstellung ist übrigens zu ersehen, daß die Sektoren $00 und $0F die beiden einzigen sind, bei denen physische und logische Sektoren identisch sind, und zwar sowohl unter DOS wie ProDOS.

1.4.1.3. Skewing-Tabelle

Die nachstehende ,,Skewing-Tabelle", die mit dem Programm ,,ProDOS Skewing-Tabelle" erzeugt wurde, enthält zum bequemen Ablesen die Blocks von $0000-$0117 mit den entsprechenden Tracks und Sektoren:

,,Ts Ss Ts Ss" = zweimal Track-Sektor s = Skewing (logisch)
,,To So To So" = zweimal Track-Sektor o = Original (physisch)

Im PRODOS-Betriebssystem in der Language Card existiert keine Skewing-Tabelle, sondern die Umrechnung erfolgt nach einem Algorithmus, der in das Programm ,,ProDOS Skewing-Tabelle", Zeile 104 ff. eingeflossen ist. Der Algorithmus findet sich in dem LC-Disk-Driver ab $F810.

```
                        1               ORG   $803
                        2    *
                        3    * Block-Reader
                        4    * ============
                        5    *
                        6    * U.Stiehl/13.4.84
                        7    *
                        8    * 80-Zeichenkarte und/oder
                        9    * Drucker einschalten, dann
                        10   * nach BRUN BLOCKREADER jeweils
                        11   * &+Blocknummer eingeben, z.B.
                        12   *
                        13   * &10 = Block 10
                        14   *
                        15   * Blocknummern im Bereich 0-279
                        16   *
                        17   * Bei Bedarf Slot + Drive ändern
                        18   * POKE 2054,S und POKE 2055,D
                        19   *
                        20   *
                        21   IND       EQU   $CE
                        22   LINENUM   EQU   $50
                        23   DOSWARM   EQU   $3D0
                        24   AMPER     EQU   $3F5
                        25   PUFFANF   EQU   $1000
                        26   PUFFMID   EQU   $1100
                        27   PUFFEND   EQU   $1200
                        28   MLI       EQU   $BF00
                        29   FRMEVAL   EQU   $DD7B
                        30   GETADR    EQU   $E752
                        31   TEXT      EQU   $FB39
                        32   HOME      EQU   $FC58
                        33   HEXOUT    EQU   $FDDA
                        34   PRINT     EQU   $FDED
                        35   BELL      EQU   $FF3A
                        36   *
0803: 4C 0B 08          37               JMP   AMPER1
0806: 06                38   SLOT      HEX   06          ;6
0807: 01                39   DRIVE     HEX   01          ;1
                        40   *
                        41   * Block-Obergrenze: 279
                        42   *
0808: 17                43   BLOCKLG   HEX   17          ;279
0809: 01                44   BLOCKHG   HEX   01
                        45   *
                        46   * Command: Read
                        47   *
080A: 80                48   COMMAND   HEX   80          ;Read
                        49   *
                        50   * Ampersand-Vektor
                        51   *
080B: A9 4C             52   AMPER1    LDA   #$4C        ;JMP
080D: 8D F5 03          53               STA   AMPER
```

```
0810: A9 1B      55             LDA  #<START
0812: 8D F6 03   56             STA  AMPER+1
0815: A9 08      57             LDA  #>START
0817: 8D F7 03   58             STA  AMPER+2
081A: 60         59             RTS
                 60   *
                 61   * Verwandelt Zahl nach &
                 62   * in Fließkommazahl in FAC
                 63   *
081B: 20 7B DD   64   START     JSR  FRMEVAL
                 65   *
                 66   * Verwandelt Fließkommazahl
                 67   * in Hexzahl in LINENUM
                 68   *
081E: 20 52 E7   69             JSR  GETADR
0821: A5 50      70             LDA  LINENUM
0823: 8D 34 09   71             STA  BLOCKL
0826: A5 51      72             LDA  LINENUM+1
0828: 8D 35 09   73             STA  BLOCKH
082B: 4C 34 08   74             JMP  CHECK
                 75   *
082E: 20 3A FF   76   ERROR     JSR  BELL
0831: 4C D0 03   77             JMP  DOSWARM
                 78   *
                 79   * Parameter okay?
                 80   *
0834: CD 09 08   81   CHECK     CMP  BLOCKHG
0837: 90 08      82             BCC  CHECK1
0839: AD 08 08   83             LDA  BLOCKLG
083C: CD 34 09   84             CMP  BLOCKL
083F: 90 ED      85             BCC  ERROR
0841: AD 07 08   86   CHECK1    LDA  DRIVE
0844: F0 E8      87             BEQ  ERROR       ;=0?
0846: C9 03      88             CMP  #3          ;>2?
0848: B0 E4      89             BCS  ERROR
084A: AD 06 08   90             LDA  SLOT
084D: F0 DF      91             BEQ  ERROR       ;=0?
084F: C9 08      92             CMP  #8          ;>7?
0851: B0 DB      93             BCS  ERROR
                 94   *
0853: AD 06 08   95   SLOTDRIV LDA  SLOT
0856: 0A         96             ASL
0857: 0A         97             ASL
0858: 0A         98             ASL
0859: 0A         99             ASL
085A: 8D 31 09   100            STA  UNITNUM
085D: 38         101            SEC
085E: AD 07 08   102            LDA  DRIVE
0861: E9 01      103            SBC  #1
0863: F0 09      104            BEQ  MENU1
0865: 18         105            CLC
0866: AD 31 09   106            LDA  UNITNUM
0869: 69 80      107            ADC  #%10000000
```

```
086B: 8D 31 09  109             STA  UNITNUM
                110  *
086E: 20 39 FB  111  MENU1      JSR  TEXT
0871: 20 58 FC  112             JSR  HOME
0874: A0 00     113             LDY  #0
0876: B9 36 09  114  MENU2      LDA  STRING,Y
0879: F0 06     115             BEQ  MENU3
087B: 20 ED FD  116             JSR  PRINT
087E: C8        117             INY
087F: D0 F5     118             BNE  MENU2
0881: AD 35 09  119  MENU3      LDA  BLOCKH
0884: 20 DA FD  120             JSR  HEXOUT
0887: AD 34 09  121             LDA  BLOCKL
088A: 20 DA FD  122             JSR  HEXOUT
088D: A9 8D     123             LDA  #$8D
088F: 20 ED FD  124             JSR  PRINT
0892: A9 A0     125             LDA  #$A0
0894: 20 ED FD  126             JSR  PRINT
0897: A9 8D     127             LDA  #$8D
0899: 20 ED FD  128             JSR  PRINT
                129  *
089C: AD 0A 08  130             LDA  COMMAND
089F: 8D 25 09  131             STA  COMMAND1
                132  *
08A2: 20 22 09  133             JSR  RDBLOCK
                134  *
08A5: A9 00     135             LDA  #<PUFFANF
08A7: 85 CE     136             STA  IND
08A9: A9 10     137             LDA  #>PUFFANF
08AB: 85 CF     138             STA  IND+1
                139  *
08AD: A9 A4     140  ADDRESS    LDA  #"$"
08AF: 20 ED FD  141             JSR  PRINT
08B2: A5 CF     142             LDA  IND+1
08B4: 20 DA FD  143             JSR  HEXOUT
08B7: A5 CE     144             LDA  IND
08B9: 20 DA FD  145             JSR  HEXOUT
08BC: A9 A0     146             LDA  #$A0
08BE: 20 ED FD  147             JSR  PRINT
08C1: A9 A0     148             LDA  #$A0
08C3: 20 ED FD  149             JSR  PRINT
                150  *
08C6: A0 00     151             LDY  #0
08C8: B1 CE     152  ASCII1     LDA  (IND),Y
08CA: 09 80     153             ORA  #$80
08CC: C9 FF     154             CMP  #$FF        ;DEL
08CE: F0 04     155             BEQ  PUNKT
08D0: C9 A0     156             CMP  #$A0
08D2: B0 02     157             BCS  ASCII2
08D4: A9 AE     158  PUNKT      LDA  #"."
08D6: 20 ED FD  159  ASCII2     JSR  PRINT
08D9: C8        160             INY
08DA: C0 10     161             CPY  #16
```

```
08DC: D0 EA      163            BNE  ASCII1
08DE: A9 A0      164            LDA  #$A0
08E0: 20 ED FD   165            JSR  PRINT
                 166  *
08E3: A0 00      167            LDY  #0
08E5: A9 A0      168  HEX1      LDA  #$A0
08E7: 20 ED FD   169            JSR  PRINT
08EA: B1 CE      170            LDA  (IND),Y
08EC: 20 DA FD   171            JSR  HEXOUT
08EF: C8         172            INY
08F0: C0 10      173            CPY  #16
08F2: D0 F1      174            BNE  HEX1
08F4: A9 8D      175            LDA  #$8D
08F6: 20 ED FD   176            JSR  PRINT
                 177  *
                 178  * Um 16 erhöhen
                 179  *
08F9: 18         180  INCR1     CLC
08FA: A5 CE      181            LDA  IND
08FC: 69 10      182            ADC  #16
08FE: 85 CE      183            STA  IND
0900: A5 CF      184            LDA  IND+1
0902: 69 00      185            ADC  #0
0904: 85 CF      186            STA  IND+1
0906: C9 11      187            CMP  #>PUFFMID
0908: D0 0C      188            BNE  INCR3
090A: A5 CE      189            LDA  IND
090C: D0 08      190            BNE  INCR3
090E: AD 00 C0   191  INCR2     LDA  $C000
0911: 10 FB      192            BPL  INCR2
0913: 2C 10 C0   193            BIT  $C010
0916: A5 CF      194  INCR3     LDA  IND+1
0918: C9 12      195            CMP  #>PUFFEND
091A: 90 03      196            BCC  INCR4
091C: 4C D0 03   197            JMP  DOSWARM    ;Exit
091F: 4C AD 08   198  INCR4     JMP  ADDRESS
                 199  *
                 200  * Read-Block-Routine
                 201  *
0922: 20 00 BF   202  RDBLOCK   JSR  MLI
0925: 80         203  COMMAND1 HEX  80         ;Read
0926: 30 09      204            DA   COUNT
0928: D0 01      205            BNE  ERROR1
092A: 60         206            RTS
092B: 68         207  ERROR1    PLA
092C: 68         208            PLA
092D: 4C 2E 08   209            JMP  ERROR
0930: 03         210  COUNT     HEX  03
0931: 00         211  UNITNUM   HEX  00
0932: 00 10      212  PUFFER    DA   PUFFANF
0934: 00         213  BLOCKL    HEX  00
0935: 00         214  BLOCKH    HEX  00
                 215  *
```

```
0936: A0 8D 8D  217  STRING   HEX  A08D8D
0939: C2 CC CF  218           ASC  "BLOCK: $"
093C: C3 CB BA A0 A4
0941: 00        219           HEX  00

--End assembly--

319 bytes

Errors: 0
```

Hätten Sie's gewußt?

Wenn man von der 64K-Karte-RAM-Disk den Block 7 per MLI zu lesen versucht, dreht der RAM-Disk-Driver durch und ,,liest'' statt Block 7 die Zero-Page und den Stack der unteren 64K!

```
                        1     *
                        2     * ProDOS Skewing-Tabelle
                        3     * ======================
                        4     *
                        5     * Blocks, Tracks + Sektoren
                        6     *
                        7     * U.Stiehl/20.04.84
                        8     *
                        9     * s = mit Skewing
                        10    * o = original
                        11    *
                        12              ORG   $6000
                        13    PRINT     EQU   $FDED
                        14    HEXOUT    EQU   $FDDA
                        15    *
6000: A2 00             16              LDX   #0
6002: BD F5 60          17    KOPF      LDA   STRING,X
6005: F0 06             18              BEQ   INIT
6007: 20 ED FD          19              JSR   PRINT
600A: E8                20              INX
600B: D0 F5             21              BNE   KOPF
                        22    *
600D: 8D EF 60          23    INIT      STA   LOW
6010: 8D F0 60          24              STA   HIGH
6013: 8D F1 60          25              STA   LOW1
6016: 8D F2 60          26              STA   HIGH1
                        27    *
6019: 20 D0 60          28    LOOPER    JSR   RECHNEN1
601C: A9 A4             29              LDA   #"$"
601E: 20 ED FD          30              JSR   PRINT
6021: AD F0 60          31              LDA   HIGH
6024: 20 DA FD          32              JSR   HEXOUT
6027: AD EF 60          33              LDA   LOW
602A: 20 DA FD          34              JSR   HEXOUT
602D: A9 A0             35              LDA   #$A0
602F: 20 ED FD          36              JSR   PRINT
6032: 20 ED FD          37              JSR   PRINT
                        38    *
6035: AD F4 60          39              LDA   XSAVE
6038: 20 DA FD          40              JSR   HEXOUT
603B: A9 A0             41              LDA   #$A0
603D: 20 ED FD          42              JSR   PRINT
6040: AD F3 60          43              LDA   ASAVE
6043: 20 DA FD          44              JSR   HEXOUT
6046: A9 A0             45              LDA   #$A0
6048: 20 ED FD          46              JSR   PRINT
604B: 20 ED FD          47              JSR   PRINT
                        48    *
604E: AD F4 60          49              LDA   XSAVE
6051: 20 DA FD          50              JSR   HEXOUT
6054: A9 A0             51              LDA   #$A0
6056: 20 ED FD          52              JSR   PRINT
6059: 18                53              CLC
```

```
605A: AD F3 60   55              LDA  ASAVE
605D: 69 02      56              ADC  #2
605F: 20 DA FD   57              JSR  HEXOUT
6062: A9 A0      58              LDA  #$A0
6064: 20 ED FD   59              JSR  PRINT
6067: 20 ED FD   60              JSR  PRINT
                 61   *
606A: AD F2 60   62              LDA  HIGH1
606D: 20 DA FD   63              JSR  HEXOUT
6070: A9 A0      64              LDA  #$A0
6072: 20 ED FD   65              JSR  PRINT
6075: AD F1 60   66              LDA  LOW1
6078: 20 DA FD   67              JSR  HEXOUT
607B: A9 A0      68              LDA  #$A0
607D: 20 ED FD   69              JSR  PRINT
6080: 20 ED FD   70              JSR  PRINT
                 71   *
6083: EE F1 60   72              INC  LOW1
6086: AD F1 60   73              LDA  LOW1
6089: C9 10      74              CMP  #$10
608B: D0 08      75              BNE  INCREM
608D: A9 00      76              LDA  #$00
608F: 8D F1 60   77              STA  LOW1
6092: EE F2 60   78              INC  HIGH1
6095: AD F2 60   79   INCREM     LDA  HIGH1
6098: 20 DA FD   80              JSR  HEXOUT
609B: A9 A0      81              LDA  #$A0
609D: 20 ED FD   82              JSR  PRINT
60A0: AD F1 60   83              LDA  LOW1
60A3: 20 DA FD   84              JSR  HEXOUT
60A6: EE F1 60   85              INC  LOW1
60A9: AD F1 60   86              LDA  LOW1
60AC: C9 10      87              CMP  #$10
60AE: D0 08      88              BNE  RETURN
60B0: A9 00      89              LDA  #$00
60B2: 8D F1 60   90              STA  LOW1
60B5: EE F2 60   91              INC  HIGH1
                 92   *
60B8: A9 8D      93   RETURN     LDA  #$8D
60BA: 20 ED FD   94              JSR  PRINT
60BD: EE EF 60   95              INC  LOW
60C0: D0 0B      96              BNE  ZURUECK
60C2: EE F0 60   97              INC  HIGH
60C5: AD F0 60   98              LDA  HIGH
60C8: C9 02      99              CMP  #2
60CA: D0 01      100             BNE  ZURUECK
60CC: 60         101             RTS
60CD: 4C 19 60   102  ZURUECK    JMP  LOOPER
                 103  *
60D0: AD EF 60   104  RECHNEN1   LDA  LOW
60D3: AE F0 60   105             LDX  HIGH
60D6: 8E F4 60   106             STX  XSAVE
60D9: A0 05      107             LDY  #5
```

```
60DB: 0A          109  RECHNEN2 ASL
60DC: 2E F4 60    110           ROL  XSAVE
60DF: 88          111           DEY
60E0: D0 F9       112           BNE  RECHNEN2
60E2: 0A          113           ASL
60E3: 90 02       114           BCC  RECHNEN3
60E5: 09 10       115           ORA  #$10
60E7: 4A          116  RECHNEN3 LSR
60E8: 4A          117           LSR
60E9: 4A          118           LSR
60EA: 4A          119           LSR
60EB: 8D F3 60    120           STA  ASAVE
60EE: 60          121           RTS
                  122  *
60EF: 00          123  LOW       HEX  00
60F0: 00          124  HIGH      HEX  00
60F1: 00          125  LOW1      HEX  00
60F2: 00          126  HIGH1     HEX  00
60F3: 00          127  ASAVE     HEX  00
60F4: 00          128  XSAVE     HEX  00
                  129  *
                  130  *
60F5: 8D 8D       131  STRING    HEX  8D8D
60F7: C2 EC EF    132           ASC  "Block   "
60FA: E3 EB A0 A0
60FE: D4 F3 A0    133           ASC  "Ts Ss  Ts Ss  "
6101: D3 F3 A0 A0 D4 F3 A0 D3
6109: F3 A0 A0
610C: D4 EF A0    134           ASC  "To So  To So"
610F: D3 EF A0 A0 D4 EF A0 D3
6117: EF
6118: 8D 8D 00    135           HEX  8D8D00


--End assembly--

283 bytes

Errors: 0
```

Block	Ts	Ss	Ts	Ss	To	So	To	So	Block	Ts	Ss	Ts	Ss	To	So	To	So
$0000	00	00	00	02	00	00	00	01	$0030	06	00	06	02	06	00	06	01
$0001	00	04	00	06	00	02	00	03	$0031	06	04	06	06	06	02	06	03
$0002	00	08	00	0A	00	04	00	05	$0032	06	08	06	0A	06	04	06	05
$0003	00	0C	00	0E	00	06	00	07	$0033	06	0C	06	0E	06	06	06	07
$0004	00	01	00	03	00	08	00	09	$0034	06	01	06	03	06	08	06	09
$0005	00	05	00	07	00	0A	00	0B	$0035	06	05	06	07	06	0A	06	0B
$0006	00	09	00	0B	00	0C	00	0D	$0036	06	09	06	0B	06	0C	06	0D
$0007	00	0D	00	0F	00	0E	00	0F	$0037	06	0D	06	0F	06	0E	06	0F
$0008	01	00	01	02	01	00	01	01	$0038	07	00	07	02	07	00	07	01
$0009	01	04	01	06	01	02	01	03	$0039	07	04	07	06	07	02	07	03
$000A	01	08	01	0A	01	04	01	05	$003A	07	08	07	0A	07	04	07	05
$000B	01	0C	01	0E	01	06	01	07	$003B	07	0C	07	0E	07	06	07	07
$000C	01	01	01	03	01	08	01	09	$003C	07	01	07	03	07	08	07	09
$000D	01	05	01	07	01	0A	01	0B	$003D	07	05	07	07	07	0A	07	0B
$000E	01	09	01	0B	01	0C	01	0D	$003E	07	09	07	0B	07	0C	07	0D
$000F	01	0D	01	0F	01	0E	01	0F	$003F	07	0D	07	0F	07	0E	07	0F
$0010	02	00	02	02	02	00	02	01	$0040	08	00	08	02	08	00	08	01
$0011	02	04	02	06	02	02	02	03	$0041	08	04	08	06	08	02	08	03
$0012	02	08	02	0A	02	04	02	05	$0042	08	08	08	0A	08	04	08	05
$0013	02	0C	02	0E	02	06	02	07	$0043	08	0C	08	0E	08	06	08	07
$0014	02	01	02	03	02	08	02	09	$0044	08	01	08	03	08	08	08	09
$0015	02	05	02	07	02	0A	02	0B	$0045	08	05	08	07	08	0A	08	0B
$0016	02	09	02	0B	02	0C	02	0D	$0046	08	09	08	0B	08	0C	08	0D
$0017	02	0D	02	0F	02	0E	02	0F	$0047	08	0D	08	0F	08	0E	08	0F
$0018	03	00	03	02	03	00	03	01	$0048	09	00	09	02	09	00	09	01
$0019	03	04	03	06	03	02	03	03	$0049	09	04	09	06	09	02	09	03
$001A	03	08	03	0A	03	04	03	05	$004A	09	08	09	0A	09	04	09	05
$001B	03	0C	03	0E	03	06	03	07	$004B	09	0C	09	0E	09	06	09	07
$001C	03	01	03	03	03	08	03	09	$004C	09	01	09	03	09	08	09	09
$001D	03	05	03	07	03	0A	03	0B	$004D	09	05	09	07	09	0A	09	0B
$001E	03	09	03	0B	03	0C	03	0D	$004E	09	09	09	0B	09	0C	09	0D
$001F	03	0D	03	0F	03	0E	03	0F	$004F	09	0D	09	0F	09	0E	09	0F
$0020	04	00	04	02	04	00	04	01	$0050	0A	00	0A	02	0A	00	0A	01
$0021	04	04	04	06	04	02	04	03	$0051	0A	04	0A	06	0A	02	0A	03
$0022	04	08	04	0A	04	04	04	05	$0052	0A	08	0A	0A	0A	04	0A	05
$0023	04	0C	04	0E	04	06	04	07	$0053	0A	0C	0A	0E	0A	06	0A	07
$0024	04	01	04	03	04	08	04	09	$0054	0A	01	0A	03	0A	08	0A	09
$0025	04	05	04	07	04	0A	04	0B	$0055	0A	05	0A	07	0A	0A	0A	0B
$0026	04	09	04	0B	04	0C	04	0D	$0056	0A	09	0A	0B	0A	0C	0A	0D
$0027	04	0D	04	0F	04	0E	04	0F	$0057	0A	0D	0A	0F	0A	0E	0A	0F
$0028	05	00	05	02	05	00	05	01	$0058	0B	00	0B	02	0B	00	0B	01
$0029	05	04	05	06	05	02	05	03	$0059	0B	04	0B	06	0B	02	0B	03
$002A	05	08	05	0A	05	04	05	05	$005A	0B	08	0B	0A	0B	04	0B	05
$002B	05	0C	05	0E	05	06	05	07	$005B	0B	0C	0B	0E	0B	06	0B	07
$002C	05	01	05	03	05	08	05	09	$005C	0B	01	0B	03	0B	08	0B	09
$002D	05	05	05	07	05	0A	05	0B	$005D	0B	05	0B	07	0B	0A	0B	0B
$002E	05	09	05	0B	05	0C	05	0D	$005E	0B	09	0B	0B	0B	0C	0B	0D
$002F	05	0D	05	0F	05	0E	05	0F	$005F	0B	0D	0B	0F	0B	0E	0B	0F

Block	Ts	Ss	Ts	Ss	To	So	To	So	Block	Ts	Ss	Ts	Ss	To	So	To	So
$0060	0C	00	0C	02	0C	00	0C	01	$0090	12	00	12	02	12	00	12	01
$0061	0C	04	0C	06	0C	02	0C	03	$0091	12	04	12	06	12	02	12	03
$0062	0C	08	0C	0A	0C	04	0C	05	$0092	12	08	12	0A	12	04	12	05
$0063	0C	0C	0C	0E	0C	06	0C	07	$0093	12	0C	12	0E	12	06	12	07
$0064	0C	01	0C	03	0C	08	0C	09	$0094	12	01	12	03	12	08	12	09
$0065	0C	05	0C	07	0C	0A	0C	0B	$0095	12	05	12	07	12	0A	12	0B
$0066	0C	09	0C	0B	0C	0C	0C	0D	$0096	12	09	12	0B	12	0C	12	0D
$0067	0C	0D	0C	0F	0C	0E	0C	0F	$0097	12	0D	12	0F	12	0E	12	0F
$0068	0D	00	0D	02	0D	00	0D	01	$0098	13	00	13	02	13	00	13	01
$0069	0D	04	0D	06	0D	02	0D	03	$0099	13	04	13	06	13	02	13	03
$006A	0D	08	0D	0A	0D	04	0D	05	$009A	13	08	13	0A	13	04	13	05
$006B	0D	0C	0D	0E	0D	06	0D	07	$009B	13	0C	13	0E	13	06	13	07
$006C	0D	01	0D	03	0D	08	0D	09	$009C	13	01	13	03	13	08	13	09
$006D	0D	05	0D	07	0D	0A	0D	0B	$009D	13	05	13	07	13	0A	13	0B
$006E	0D	09	0D	0B	0D	0C	0D	0D	$009E	13	09	13	0B	13	0C	13	0D
$006F	0D	0D	0D	0F	0D	0E	0D	0F	$009F	13	0D	13	0F	13	0E	13	0F
$0070	0E	00	0E	02	0E	00	0E	01	$00A0	14	00	14	02	14	00	14	01
$0071	0E	04	0E	06	0E	02	0E	03	$00A1	14	04	14	06	14	02	14	03
$0072	0E	08	0E	0A	0E	04	0E	05	$00A2	14	08	14	0A	14	04	14	05
$0073	0E	0C	0E	0E	0E	06	0E	07	$00A3	14	0C	14	0E	14	06	14	07
$0074	0E	01	0E	03	0E	08	0E	09	$00A4	14	01	14	03	14	08	14	09
$0075	0E	05	0E	07	0E	0A	0E	0B	$00A5	14	05	14	07	14	0A	14	0B
$0076	0E	09	0E	0B	0E	0C	0E	0D	$00A6	14	09	14	0B	14	0C	14	0D
$0077	0E	0D	0E	0F	0E	0E	0E	0F	$00A7	14	0D	14	0F	14	0E	14	0F
$0078	0F	00	0F	02	0F	00	0F	01	$00A8	15	00	15	02	15	00	15	01
$0079	0F	04	0F	06	0F	02	0F	03	$00A9	15	04	15	06	15	02	15	03
$007A	0F	08	0F	0A	0F	04	0F	05	$00AA	15	08	15	0A	15	04	15	05
$007B	0F	0C	0F	0E	0F	06	0F	07	$00AB	15	0C	15	0E	15	06	15	07
$007C	0F	01	0F	03	0F	08	0F	09	$00AC	15	01	15	03	15	08	15	09
$007D	0F	05	0F	07	0F	0A	0F	0B	$00AD	15	05	15	07	15	0A	15	0B
$007E	0F	09	0F	0B	0F	0C	0F	0D	$00AE	15	09	15	0B	15	0C	15	0D
$007F	0F	0D	0F	0F	0F	0E	0F	0F	$00AF	15	0D	15	0F	15	0E	15	0F
$0080	10	00	10	02	10	00	10	01	$00B0	16	00	16	02	16	00	16	01
$0081	10	04	10	06	10	02	10	03	$00B1	16	04	16	06	16	02	16	03
$0082	10	08	10	0A	10	04	10	05	$00B2	16	08	16	0A	16	04	16	05
$0083	10	0C	10	0E	10	06	10	07	$00B3	16	0C	16	0E	16	06	16	07
$0084	10	01	10	03	10	08	10	09	$00B4	16	01	16	03	16	08	16	09
$0085	10	05	10	07	10	0A	10	0B	$00B5	16	05	16	07	16	0A	16	0B
$0086	10	09	10	0B	10	0C	10	0D	$00B6	16	09	16	0B	16	0C	16	0D
$0087	10	0D	10	0F	10	0E	10	0F	$00B7	16	0D	16	0F	16	0E	16	0F
$0088	11	00	11	02	11	00	11	01	$00B8	17	00	17	02	17	00	17	01
$0089	11	04	11	06	11	02	11	03	$00B9	17	04	17	06	17	02	17	03
$008A	11	08	11	0A	11	04	11	05	$00BA	17	08	17	0A	17	04	17	05
$008B	11	0C	11	0E	11	06	11	07	$00BB	17	0C	17	0E	17	06	17	07
$008C	11	01	11	03	11	08	11	09	$00BC	17	01	17	03	17	08	17	09
$008D	11	05	11	07	11	0A	11	0B	$00BD	17	05	17	07	17	0A	17	0B
$008E	11	09	11	0B	11	0C	11	0D	$00BE	17	09	17	0B	17	0C	17	0D
$008F	11	0D	11	0F	11	0E	11	0F	$00BF	17	0D	17	0F	17	0E	17	0F

Block	Ts	Ss	Ts	Ss	To	So	To	So
$00C0	18	00	18	02	18	00	18	01
$00C1	18	04	18	06	18	02	18	03
$00C2	18	08	18	0A	18	04	18	05
$00C3	18	0C	18	0E	18	06	18	07
$00C4	18	01	18	03	18	08	18	09
$00C5	18	05	18	07	18	0A	18	0B
$00C6	18	09	18	0B	18	0C	18	0D
$00C7	18	0D	18	0F	18	0E	18	0F
$00C8	19	00	19	02	19	00	19	01
$00C9	19	04	19	06	19	02	19	03
$00CA	19	08	19	0A	19	04	19	05
$00CB	19	0C	19	0E	19	06	19	07
$00CC	19	01	19	03	19	08	19	09
$00CD	19	05	19	07	19	0A	19	0B
$00CE	19	09	19	0B	19	0C	19	0D
$00CF	19	0D	19	0F	19	0E	19	0F
$00D0	1A	00	1A	02	1A	00	1A	01
$00D1	1A	04	1A	06	1A	02	1A	03
$00D2	1A	08	1A	0A	1A	04	1A	05
$00D3	1A	0C	1A	0E	1A	06	1A	07
$00D4	1A	01	1A	03	1A	08	1A	09
$00D5	1A	05	1A	07	1A	0A	1A	0B
$00D6	1A	09	1A	0B	1A	0C	1A	0D
$00D7	1A	0D	1A	0F	1A	0E	1A	0F
$00D8	1B	00	1B	02	1B	00	1B	01
$00D9	1B	04	1B	06	1B	02	1B	03
$00DA	1B	08	1B	0A	1B	04	1B	05
$00DB	1B	0C	1B	0E	1B	06	1B	07
$00DC	1B	01	1B	03	1B	08	1B	09
$00DD	1B	05	1B	07	1B	0A	1B	0B
$00DE	1B	09	1B	0B	1B	0C	1B	0D
$00DF	1B	0D	1B	0F	1B	0E	1B	0F
$00E0	1C	00	1C	02	1C	00	1C	01
$00E1	1C	04	1C	06	1C	02	1C	03
$00E2	1C	08	1C	0A	1C	04	1C	05
$00E3	1C	0C	1C	0E	1C	06	1C	07
$00E4	1C	01	1C	03	1C	08	1C	09
$00E5	1C	05	1C	07	1C	0A	1C	0B
$00E6	1C	09	1C	0B	1C	0C	1C	0D
$00E7	1C	0D	1C	0F	1C	0E	1C	0F
$00E8	1D	00	1D	02	1D	00	1D	01
$00E9	1D	04	1D	06	1D	02	1D	03
$00EA	1D	08	1D	0A	1D	04	1D	05
$00EB	1D	0C	1D	0E	1D	06	1D	07
$00EC	1D	01	1D	03	1D	08	1D	09
$00ED	1D	05	1D	07	1D	0A	1D	0B
$00EE	1D	09	1D	0B	1D	0C	1D	0D
$00EF	1D	0D	1D	0F	1D	0E	1D	0F

Block	Ts	Ss	Ts	Ss	To	So	To	So
$00F0	1E	00	1E	02	1E	00	1E	01
$00F1	1E	04	1E	06	1E	02	1E	03
$00F2	1E	08	1E	0A	1E	04	1E	05
$00F3	1E	0C	1E	0E	1E	06	1E	07
$00F4	1E	01	1E	03	1E	08	1E	09
$00F5	1E	05	1E	07	1E	0A	1E	0B
$00F6	1E	09	1E	0B	1E	0C	1E	0D
$00F7	1E	0D	1E	0F	1E	0E	1E	0F
$00F8	1F	00	1F	02	1F	00	1F	01
$00F9	1F	04	1F	06	1F	02	1F	03
$00FA	1F	08	1F	0A	1F	04	1F	05
$00FB	1F	0C	1F	0E	1F	06	1F	07
$00FC	1F	01	1F	03	1F	08	1F	09
$00FD	1F	05	1F	07	1F	0A	1F	0B
$00FE	1F	09	1F	0B	1F	0C	1F	0D
$00FF	1F	0D	1F	0F	1F	0E	1F	0F
$0100	20	00	20	02	20	00	20	01
$0101	20	04	20	06	20	02	20	03
$0102	20	08	20	0A	20	04	20	05
$0103	20	0C	20	0E	20	06	20	07
$0104	20	01	20	03	20	08	20	09
$0105	20	05	20	07	20	0A	20	0B
$0106	20	09	20	0B	20	0C	20	0D
$0107	20	0D	20	0F	20	0E	20	0F
$0108	21	00	21	02	21	00	21	01
$0109	21	04	21	06	21	02	21	03
$010A	21	08	21	0A	21	04	21	05
$010B	21	0C	21	0E	21	06	21	07
$010C	21	01	21	03	21	08	21	09
$010D	21	05	21	07	21	0A	21	0B
$010E	21	09	21	0B	21	0C	21	0D
$010F	21	0D	21	0F	21	0E	21	0F
$0110	22	00	22	02	22	00	22	01
$0111	22	04	22	06	22	02	22	03
$0112	22	08	22	0A	22	04	22	05
$0113	22	0C	22	0E	22	06	22	07
$0114	22	01	22	03	22	08	22	09
$0115	22	05	22	07	22	0A	22	0B
$0116	22	09	22	0B	22	0C	22	0D
$0117	22	0D	22	0F	22	0E	22	0F

1.4.1.4. Read Address Field

Um sich davon zu überzeugen, daß auf mit dem ,,FILER''-Programm initialisierten ProDOS-Disketten die Sektoren nicht physisch (!) versetzt angeordnet sind, wurde das kleine Programm ,,Read Address Field'' entwickelt, das nur unter DOS 3.3 läuft, da bei ProDOS keine Seek-Routine (als MLI-Befehl) implementiert ist. Dieses Programm sucht zunächst Track 0, Sektor 0 und liest dann alle Sektorvorspanns ab Sektor $00, die am Lesekopf vorbeikommen, insgesamt viermal ein (= 64 Vorspanns) und legt den jeweiligen Sektorvorspann (Address Field, Address Header), der sich aus Prüf-Nummer, Sektor-Nummer, Spur-Nummer und Volume-Nummer zusammensetzt, ab $1100 im Speicher als je 4 Bytes ab. Im Monitor kann man dann sehen, daß die Sektorvorspanns tatsächlich von $00-$0F ohne Lücken, d.h. ohne physisches Interleaving, eingelesen wurden. Dies gilt sowohl für ProDOS- wie auch für DOS-formatierte Disketten. Der Vollständigkeit halber sei erwähnt, daß die 256 Bytes eines Daten-Sektors auf der Diskette ,,nibble-mäßig'' verschlüsselt sind und einen Vorspann und Nachspann haben, damit der Disk-Driver Anfang und Ende eines Sektors erkennen kann:

0. Synchronisierungs-Nibbles
1. Adreßfeld
1.1. Adreßfeld-Vorspann $D5 $AA $96
1.2. Volume-Nummer als 2 Nibbles
1.3. Track-Nummer als 2 Nibbles
1.4. Sektor-Nummer als 2 Nibbles
1.5. Prüfsumme als 2 Nibbles
1.6. Adreßfeld-Nachspann $DE $AA $EB
2. Datenfeld
2.1. Datenfeld-Vorspann $D5 $AA $AD
2.2. 256 Daten-Bytes als 342 Nibbles
2.3. Prüfsumme als 1 Nibble
2.4. Datenfeld-Nachspann $DE $AA $EB

1.4.1.5. PRODOS-READER unter DOS 3.3

Will man die Spuren einer ProDOS-Diskette unter dem DOS 3.3 Betriebssystem lesen, dann ergeben sich zwei Möglichkeiten:

a) Man ersetzt vorübergehend die 16 Bytes umfassende DOS-Skewing-Tabelle ab $BFB8 durch die logische ProDOS-Sektoren-Folge. Dieses Verfahren ist weniger empfehlenswert, da die Originalwerte nach Beendigung der Lese-Routine wieder zurückgepatcht werden müssen (andernfalls würde später DOS bei CATALOG usw. ,,abstürzen").

b) Man implementiert in dem Spur-Lese-Programm eine Skewing-Konvertiertabelle, die die originalen DOS-Skewing-Werte entsprechend umrechnet. Dieses letztere Verfahren ist bei dem Programm ,,PRODOS-READER unter DOS 3.3" angewandt worden. Die Konvertiertabelle steht in den Zeilen 142-143 des Assemblerlistings.

Der PRODOS-READER ist menü-gesteuert. Mit der ,,R"-Taste kann man lesen und mit der ,,E"-Taste das Programm verlassen. Mit dem Minus- bzw. Kleiner-als-Zeichen kann man den Spur-Zähler vermindern und sinngemäß mit dem Plus- bzw. Größer-als-Zeichen erhöhen. Der PRODOS-READER ist nur für die 80-Zeichen-Karte des Apple IIe gedacht und enthält übrigens eine sehr schnelle, neuentwickelte 80-Zeichen-Routine, die bis zu 43% schneller als die Apple-Firmware-Routine ist und die man in eigene 80-Zeichen-Programme einbauen kann. Nach Änderung der Konvertiertabelle lassen sich auch andere 16-Sektoren-Disketten, z.B. CP/M-Disketten, einlesen. Für Pascal-Disketten ist keine Änderung erforderlich.

```
                        1               ORG   $1000
                        2    *
                        3    * Read Address Field
                        4    * ==================
                        5    *
                        6    * DOS 3.3 Programm
                        7    * U.Stiehl/30.4.84
                        8    *
1000: 4C 05 10          9               JMP   START
                        10   *
                        11   * Hier ggf. Track
                        12   * + Sektor poken
                        13   *
1003: 00                14   SOLLTRK   HEX   00
1004: 00                15   SOLLSEC   HEX   00
                        16   *
                        17   CHECK     EQU   $2C          ;Prüfs.
                        18   SECTOR    EQU   $2D
                        19   TRACK     EQU   $2E
                        20   VOLUME    EQU   $2F
                        21   IND1      EQU   $CE
                        22   SLOT      EQU   $FE
                        23   YSAVE     EQU   $FF
                        24   IOBWO?    EQU   $3E3
                        25   RWTS      EQU   $3D9
                        26   PUFFER    EQU   $1100
                        27   READADR   EQU   $B944
                        28   BELL      EQU   $FF3A
                        29   *
1005: 20 E3 03          30   START     JSR   IOBWO?
1008: 84 CE             31             STY   IND1
100A: 85 CF             32             STA   IND1+1
                        33   *
100C: A0 01             34             LDY   #$01
100E: B1 CE             35             LDA   (IND1),Y
1010: 85 FE             36             STA   SLOT
                        37   *
1012: A0 04             38             LDY   #$04
1014: AD 03 10          39             LDA   SOLLTRK
1017: 91 CE             40             STA   (IND1),Y
                        41   *
1019: C8                42             INY
101A: AD 04 10          43             LDA   SOLLSEC
101D: 91 CE             44             STA   (IND1),Y
                        45   *
101F: A0 0C             46             LDY   #$0C
1021: A9 00             47             LDA   #$00         ;Seek
1023: 91 CE             48             STA   (IND1),Y
                        49   *
1025: A0 00             50             LDY   #0
1027: 84 FF             51             STY   YSAVE
                        52   *
1029: 20 E3 03          53             JSR   IOBWO?
```

```
102C: 20 D9 03   55              JSR  RWTS
                 56   *
102F: A6 FE      57              LDX  SLOT
1031: BD 89 C0   58              LDA  $C089,X     ;Motoron
1034: BD 8E C0   59              LDA  $C08E,X     ;Latch
1037: BD 8C C0   60   READ1      LDA  $C08C,X     ;Strobe
103A: 10 FB      61              BPL  READ1
                 62   *
103C: 20 44 B9   63   READ2      JSR  READADR
103F: B0 35      64              BCS  ERROR1
1041: A5 2D      65   READ3      LDA  SECTOR
1043: CD 04 10   66              CMP  SOLLSEC
1046: D0 F4      67              BNE  READ2
                 68   *
                 69   * Hier werden jeweils
                 70   * Prüfsumme, Track, Sektor
                 71   * und Volume-Nummer vom
                 72   * Address Header ab $1100
                 73   * abgespeichert, und zwar
                 74   * 64mal $1000-$1003,
                 75   * $1004-$1007 usw.
                 76   *
1048: A4 FF      77   READ4      LDY  YSAVE
104A: A5 2C      78              LDA  CHECK
104C: 99 00 11   79              STA  PUFFER,Y
104F: C8         80              INY
1050: A5 2D      81              LDA  SECTOR
1052: 99 00 11   82              STA  PUFFER,Y
1055: C8         83              INY
1056: A5 2E      84              LDA  TRACK
1058: 99 00 11   85              STA  PUFFER,Y
105B: C8         86              INY
105C: A5 2F      87              LDA  VOLUME
105E: 99 00 11   88              STA  PUFFER,Y
1061: C8         89              INY
1062: F0 15      90              BEQ  EXIT1
1064: 84 FF      91              STY  YSAVE
1066: BD 89 C0   92              LDA  $C089,X     ;Motoron
1069: BD 8E C0   93              LDA  $C08E,X     ;Latch
106C: BD 8C C0   94   READ5      LDA  $C08C,X     ;Strobe
106F: 10 FB      95              BPL  READ5
1071: 20 44 B9   96              JSR  READADR
1074: 90 D2      97              BCC  READ4
1076: 20 3A FF   98   ERROR1     JSR  BELL
1079: BD 88 C0   99   EXIT1      LDA  $C088,X     ;Mot.off
107C: 60         100             RTS

--End assembly--

125 bytes

Errors: 0
```

```
                        1               ORG   $3000       ;12288
                        2    *
                        3    *
                        4    * PRODOS-READER unter DOS 3.3
                        5    * ============================
                        6    *
                        7    * 15.03.84/U.Stiehl
                        8    * mit normalem Skewing und
                        9    * 80-Zeichen-Darstellung PR#3!
                       10    *
                       11    PUFFER    EQU   $2000
                       12    IND       EQU   $CE
                       13    GETIOB    EQU   $3E3
                       14    RWTS      EQU   $3D9
                       15    DOSWARM   EQU   $3D0
                       16    HEXOUT    EQU   $FDDA
                       17    HOME      EQU   $FC58
                       18    *
3000: 20 00 C3         19              JSR   $C300       ;PR#3
3003: 20 58 FC         20    MENU1     JSR   HOME
3006: A2 00            21              LDX   #0
3008: BD 3A 31         22    MENU2     LDA   STRING,X
300B: F0 06            23              BEQ   MENU3
300D: 20 88 31         24              JSR   PRINT80
3010: E8               25              INX
3011: D0 F5            26              BNE   MENU2
3013: AD 6F 30         27    MENU3     LDA   TRACK
3016: 20 DA FD         28              JSR   HEXOUT
3019: AD 00 C0         29    MENU4     LDA   $C000
301C: 10 FB            30              BPL   MENU4
301E: 2C 10 C0         31              BIT   $C010
3021: C9 C5            32              CMP   #"E"
3023: F0 25            33              BEQ   EXIT
3025: C9 E5            34              CMP   #"e"
3027: F0 21            35              BEQ   EXIT
3029: C9 D2            36              CMP   #"R"
302B: F0 17            37              BEQ   READ
302D: C9 F2            38              CMP   #"r"
302F: F0 13            39              BEQ   READ
3031: C9 BC            40              CMP   #"<"
3033: F0 28            41              BEQ   DECR1
3035: C9 AD            42              CMP   #"-"
3037: F0 24            43              BEQ   DECR1
3039: C9 BE            44              CMP   #">"
303B: F0 10            45              BEQ   INCR1
303D: C9 AB            46              CMP   #"+"
303F: F0 0C            47              BEQ   INCR1
3041: 4C 19 30         48              JMP   MENU4
                       49    *
3044: 20 80 30         50    READ      JSR   INITIAL
3047: 4C 03 30         51              JMP   MENU1
                       52    *
304A: 4C D0 03         53    EXIT      JMP   DOSWARM
```

```
                        55   *
304D: EE 6F 30          56   INCR1     INC  TRACK
3050: AD 6F 30          57             LDA  TRACK
3053: C9 23             58             CMP  #$23
3055: 90 AC             59             BCC  MENU1
3057: CE 6F 30          60             DEC  TRACK
305A: 4C 03 30          61             JMP  MENU1
                        62   *
305D: CE 6F 30          63   DECR1     DEC  TRACK
3060: AD 6F 30          64             LDA  TRACK
3063: 10 9E             65             BPL  MENU1
3065: EE 6F 30          66             INC  TRACK
3068: 4C 03 30          67             JMP  MENU1
                        68   *
306B: 01                69   IOB       HEX  01
306C: 60                70   SLOT      HEX  60          ;SLOT6
306D: 01                71   DRIVE     HEX  01          ;DRIVE1
306E: 00                72   VOLUME    HEX  00
306F: 00                73   TRACK     HEX  00
3070: 00                74   SECTOR    HEX  00          ;SECT
3071: 7C                75   DCTLOW    DFB  #<DCT
3072: 30                76   DCTHIGH   DFB  #>DCT
3073: 00                77   PUFFLOW   HEX  00          ;$2000
3074: 20                78   PUFFHIGH  HEX  20
3075: 00                79             HEX  00          ;UNUSED
3076: 00                80             HEX  00          ;UNUSED
3077: 01                81   COMMAND   HEX  01          ;READ
3078: 00                82   DOSERROR  HEX  00
3079: 00                83   VORVOL    HEX  00
307A: 60                84   VORSLOT   HEX  60          ;SLOT6
307B: 01                85   VORDRIVE  HEX  01          ;DRIVE1
                        86   *
307C: 00 01 EF          87   DCT       HEX  0001EFD8
307F: D8
                        88   *
                        89   * 1 Track in Puffer $2000-2FFF
                        90   *
3080: 20 E3 03          91   INITIAL   JSR  GETIOB
3083: 84 CE             92             STY  IND         ;LOWIOB
3085: 85 CF             93             STA  IND+1       ;HIGHIOB
3087: A0 01             94             LDY  #1
3089: B1 CE             95             LDA  (IND),Y
308B: 8D 6C 30          96             STA  SLOT
308E: 8D 7A 30          97             STA  VORSLOT
3091: C8                98             INY
3092: B1 CE             99             LDA  (IND),Y
3094: 8D 6D 30          100            STA  DRIVE
3097: 8D 7B 30          101            STA  VORDRIVE
                        102  *
309A: A2 10             103  READO     LDX  #16
309C: 8E DE 30          104            STX  XSAVE
309F: A9 00             105            LDA  #$00
30A1: 8D 73 30          106            STA  PUFFLOW
                        107  *
```

```
                        109  * Sektor 0F-00 absteigend
                        110  *
30A4: CE DE 30          111  READ1    DEC  XSAVE
30A7: AE DE 30          112          LDX  XSAVE
30AA: 30 43             113          BMI  DISPLAY
30AC: 8E 70 30          114          STX  SECTOR
                        115  *
30AF: BD DF 30          116          LDA  SKEW,X
30B2: 18                117          CLC
30B3: 69 20             118          ADC  #>PUFFER
30B5: 8D 74 30          119          STA  PUFFHIGH
                        120  *
30B8: A9 30             121          LDA  #>IOB
30BA: A0 6B             122          LDY  #<IOB
30BC: 20 D9 03          123          JSR  RWTS
30BF: A9 00             124          LDA  #0
30C1: 85 48             125          STA  $48        ;P-REG
30C3: B0 02             126          BCS  ERROR
30C5: 90 DD             127          BCC  READ1
                        128  *
30C7: A9 87             129  ERROR    LDA  #$87       ;CTRL-G
30C9: 20 88 31          130          JSR  PRINT80
30CC: 20 58 FC          131          JSR  HOME
30CF: AD 78 30          132          LDA  DOSERROR
30D2: 20 DA FD          133          JSR  HEXOUT
30D5: AD 00 C0          134  ENDE     LDA  $C000
30D8: 10 FB             135          BPL  ENDE
30DA: 2C 10 C0          136          BIT  $C010
30DD: 60                137          RTS
                        138  *
30DE: 00                139  XSAVE    HEX  00         ;SECTOR
                        140  *
                        141  *
                        142  SKEW     HEX  000E0D0C0B0A0908
                        143          HEX  0706050403020100F
                        144  *
                        145  * ASCII-Speicher lesen
                        146  *
30EF: 20 58 FC          147  DISPLAY  JSR  HOME
30F2: A9 00             148          LDA  #0
30F4: 85 CE             149          STA  IND
30F6: A2 FF             150          LDX  #$FF       ;FF+1
30F8: 8E DE 30          151          STX  XSAVE
                        152  *
30FB: EE DE 30          153  LOOP0    INC  XSAVE
30FE: AE DE 30          154          LDX  XSAVE
3101: E0 10             155          CPX  #16
3103: F0 D0             156          BEQ  ENDE
3105: 8A                157          TXA
3106: 18                158          CLC
3107: 69 20             159          ADC  #>PUFFER
3109: 85 CF             160          STA  IND+1
                        161  *
```

```
310B: AD 00 C0  163  KEY1     LDA  $C000
310E: 10 0F     164           BPL  SEKNUM
3110: 2C 10 C0  165           BIT  $C010
3113: C9 9B     166           CMP  #$9B        ;ESC
3115: F0 BE     167           BEQ  ENDE
3117: AD 00 C0  168  KEY2     LDA  $C000       ;WAIT
311A: 10 FB     169           BPL  KEY2
311C: 2C 10 C0  170           BIT  $C010
                171  *
311F: A9 8D     172  SEKNUM   LDA  #$8D
3121: 20 88 31  173           JSR  PRINT80
3124: A0 00     174  LOOP1    LDY  #0
3126: B1 CE     175  LOOP2    LDA  (IND),Y
3128: 09 80     176           ORA  #%10000000
312A: C9 A0     177           CMP  #$A0
312C: B0 03     178           BCS  DRUCK
312E: 38        179           SEC
312F: E9 40     180           SBC  #$40
3131: 20 88 31  181  DRUCK    JSR  PRINT80
3134: C8        182           INY
3135: D0 EF     183           BNE  LOOP2
3137: 4C FB 30  184           JMP  LOOP0
                185  *
313A: D0 D2 CF  186  STRING   ASC  "PRODOS-READER"
3147: 8D        187           HEX  8D
3148: D5 AE D3  188           ASC  "U.STIEHL 1984"
3155: 8D        189           HEX  8D
3156: D0 D5 C6  190           ASC  "PUF:2000-2FFF"
3163: 8D        191           HEX  8D
3164: BC AD A0  192           ASC  "<-  TRACK  +>"
3171: 8D        193           HEX  8D
3172: C5 BD C5  194           ASC  "E=ENDE R=READ"
317F: 8D 8D     195           HEX  8D8D
3181: D4 D2 C1  196           ASC  "TRACK:"
3187: 00        197           HEX  00
                198  *
                199  * Schnelle 80-Column-Routine
                200  * -------------------------
                201  * 27-43%, im Mittel 40% schneller
                202  * als die Apple-Firmware-Routine
                203  *
                204  WLFT     EQU  $20
                205  WWDTH    EQU  $21
                206  WTOP     EQU  $22
                207  WBTM     EQU  $23
                208  CH       EQU  $24
                209  CV       EQU  $25
                210  BASL     EQU  $28
                211  BASH     EQU  $29
                212  BAS2L    EQU  $2A
                213  BAS2H    EQU  $2B
                214  PAGE1    EQU  $C054       ;MAIN
                215  PAGE2    EQU  $C055       ;AUX
```

```
                        217  *--------PRINT--------------------
3188: 8D 4B 32          218  PRINT80  STA  AREG
318B: 8C 4C 32          219           STY  YREG
318E: 8E 4D 32          220           STX  XREG
3191: C9 8D             221           CMP  #$8D
3193: F0 23             222           BEQ  RETURN
                        223  *--------STORE A IN PAGE 1/2-----
3195: A5 24             224  STA80    LDA  CH
3197: 4A                225           LSR
3198: 90 05             226           BCC  STAPG2
319A: 8C 54 C0          227  STAPG1   STY  PAGE1
319D: B0 03             228           BCS  STA80A
319F: 8C 55 C0          229  STAPG2   STY  PAGE2
31A2: A8                230  STA80A   TAY
31A3: AD 4B 32          231           LDA  AREG
31A6: 91 28             232           STA  (BASL),Y
                        233  *--------ADVANCE CURSOR----------
31A8: 20 C3 31          234           JSR  NEXTCOL
                        235  *--------RESTORE REGISTERS-------
31AB: 8C 54 C0          236  PRINT80A STY  PAGE1
31AE: AD 4B 32          237           LDA  AREG
31B1: AC 4C 32          238           LDY  YREG
31B4: AE 4D 32          239           LDX  XREG
31B7: 60                240           RTS
                        241  *--------CARRIAGE RETURN---------
31B8: A4 24             242  RETURN   LDY  CH
31BA: 20 1D 32          243           JSR  CLEOL1
31BD: 20 CC 31          244           JSR  NEXTLINE
31C0: 4C AB 31          245           JMP  PRINT80A
                        246  *--------ADVANCE 1 COLUMN--------
31C3: E6 24             247  NEXTCOL  INC  CH
31C5: A5 24             248           LDA  CH
31C7: C5 21             249           CMP  WWDTH
31C9: B0 01             250           BCS  NEXTLINE
31CB: 60                251           RTS
                        252  *--------ADVANCE 1 LINE----------
31CC: A9 00             253  NEXTLINE LDA  #$00
31CE: 85 24             254           STA  CH
31D0: E6 25             255           INC  CV
31D2: A5 25             256           LDA  CV
31D4: C5 23             257           CMP  WBTM
31D6: 90 64             258           BCC  BASCALC
31D8: C6 25             259           DEC  CV
                        260  *--------SCROLL UP---------------
31DA: A5 22             261           LDA  WTOP
31DC: 48                262           PHA
31DD: 20 3C 32          263           JSR  BASCALC
31E0: A5 28             264  SCRL1    LDA  BASL
31E2: 85 2A             265           STA  BAS2L
31E4: A5 29             266           LDA  BASH
31E6: 85 2B             267           STA  BAS2H
31E8: 68                268           PLA
31E9: 69 01             269           ADC  #$01
```

```
31EB: C5 23      271             CMP  WBTM
31ED: B0 26      272             BCS  SCRL3
31EF: 48         273             PHA
31F0: 20 3C 32   274             JSR  BASCALC
                 275  *--------COPY LINE+1 TO LINE-----
31F3: A5 21      276  SCRL2      LDA  WWDTH
31F5: 4A         277             LSR
31F6: A8         278             TAY
31F7: 88         279             DEY
31F8: 8C 4E 32   280             STY  YSAVE
31FB: 8E 54 C0   281  SCRL2A     STX  PAGE1
31FE: B1 28      282             LDA  (BASL),Y    ;LINE+1
3200: 91 2A      283             STA  (BAS2L),Y   ;LINE
3202: 88         284             DEY
3203: 10 F6      285             BPL  SCRL2A
3205: AC 4E 32   286             LDY  YSAVE
3208: 8E 55 C0   287  SCRL2B     STX  PAGE2
320B: B1 28      288             LDA  (BASL),Y    ;LINE+1
320D: 91 2A      289             STA  (BAS2L),Y   ;LINE
320F: 88         290             DEY
3210: 10 F6      291             BPL  SCRL2B
3212: 4C E0 31   292             JMP  SCRL1
3215: A0 00      293  SCRL3      LDY  #$00
3217: 20 1D 32   294             JSR  CLEOL1
321A: 4C 3A 32   295             JMP  VTAB1
                 296  *--------CLEAR END OF LINE-------
321D: 8C 4E 32   297  CLEOL1     STY  YSAVE
3220: 98         298             TYA
3221: 4A         299             LSR
3222: 90 05      300             BCC  CLEOLPG2
3224: 8C 54 C0   301  CLEOLPG1   STY  PAGE1
3227: B0 03      302             BCS  CLEOL2
3229: 8C 55 C0   303  CLEOLPG2   STY  PAGE2
322C: A8         304  CLEOL2     TAY
322D: A9 A0      305             LDA  #$A0        ;SPACE
322F: 91 28      306             STA  (BASL),Y
3231: AC 4E 32   307             LDY  YSAVE
3234: C8         308             INY
3235: C4 21      309             CPY  WWDTH
3237: 90 E4      310             BCC  CLEOL1
3239: 60         311             RTS
                 312  *--------VTAB--------------------
323A: A5 25      313  VTAB1      LDA  CV
                 314  *--------BASCALC----------------
323C: AA         315  BASCALC    TAX
323D: BD 67 32   316             LDA  BASHADR,X
3240: 85 29      317             STA  BASH
3242: 18         318             CLC
3243: BD 4F 32   319             LDA  BASLADR,X
3246: 65 20      320             ADC  WLFT
3248: 85 28      321             STA  BASL
324A: 60         322             RTS
                 323  *--------SAVE REGISTERS----------
```

```
324B: 00          325  AREG      HEX  00
324C: 00          326  YREG      HEX  00
324D: 00          327  XREG      HEX  00
                  328  *--------------------------------
324E: 00          329  YSAVE     HEX  00
                  330  *---------SCREEN BASE LINES-------
324F: 00 80 00    331  BASLADR   HEX  0080008000800080
3252: 80 00 80 00 80
3257: 28 A8 28    332            HEX  28A828A828A828A8
325A: A8 28 A8 28 A8
325F: 50 D0 50    333            HEX  50D050D050D050D0
3262: D0 50 D0 50 D0
3267: 04 04 05    334  BASHADR   HEX  0404050506060707
326A: 05 06 06 07 07
326F: 04 04 05    335            HEX  0404050506060707
3272: 05 06 06 07 07
3277: 04 04 05    336            HEX  0404050506060707
327A: 05 06 06 07 07


--End assembly--

639 bytes

Errors: 0
```

Hätten Sie's gewußt?

Wenn man mit PR#6 kaltstartet, wird der RAM-Disk-Inhalt der 64K-Karte stets zerstört.
Fazit: Keine wichtigen Daten auf der RAM-Disk zwischenspeichern!

1.4.2. ProDOS-Diskette nach der Formatierung

Hinweis: Zu dem Kapitel 1.4.2 gehören die ,,Dumps zu Kap. 1.4.2'': Blocks $0000-$0006, die auf den Seiten 114-117 abgebildet sind.

Mit Hilfe des ,,FILER''-Programms von der ,,User's Disk'' formatieren wir eine Testdiskette, der wir den Namen ,,VOLUME'' geben. Wenn wir danach mit dem Block-Reader einzelne Blocks ab Block $00 einlesen, stellen wir fest, daß insgesamt nur die Blocks $00-$06 belegt sind. Dies sind die einzigen Blocks einer Diskette, die stets derselben Funktion dienen, d.h. weder der Urlader noch das Volume-Directory noch die Volume Bit Map können sich in anderen als den nachstehend genannten Blocks befinden.

Block $0000: Urlader, erster Block
Block $0001: Urlader, zweiter Block
Block $0002: Volume-Directory, erster Block (Key Block)
Block $0003: Volume-Directory, zweiter Block
Block $0004: Volume-Directory, dritter Block } (Non-Key-Blocks)
Block $0005: Volume-Directory, vierter Block
Block $0006: Volume Bit Map (erster und zugleich letzter Block)

Der Urlader wurde bereits kurz erwähnt, da er z.B. für das Programm ,,PRODOS.INIT'' benötigt wird.

Bei den nachfolgenden Teilkapiteln lese man zum besseren Verständnis den Text stets in Verbindung mit dem entsprechenden Block-Dump.

1.4.2.1. Struktur des Volume-Directory-Kopfes

Das maximal 4 Blocks umfassende Volume-Directory (= Hauptinhaltsverzeichnis) kann bis zu 51 File-Namen aufnehmen. Da auf der frisch initialisierten Diskette noch keine Files gespeichert wurden, interessiert uns zunächst nur der Volume-Directory-Header oder Kopf des Hauptinhaltsverzeichnisses, der sich stets im ersten Volume-Directory-Block (Block $0002) befindet, der deshalb auch als Key Block bezeichnet wird. Zunächst werden die Inhalte der relativen Byte-Stellen kurz zusammengefaßt und dann im einzelnen beschrieben. Die in Klammern angegebenen Beispiel-Werte beziehen sich auf Block $0002.

Zusammenfassung Volume-Directory-Kopf

$00-$01:	Rückwärts-Pointer (00 00 = $0000)
$02-$03:	Vorwärts-Pointer (03 00 = $0003)
$04:	Speichertyp und Namenslänge ($F6; $F = Volume-Directory-Kopf)
$05-$13:	File-Name („VOLUME“)
$14-$1B:	unbenutzt (00 00 00 00 00 00 00 00)
$1C-$1F:	Datum und Uhrzeit der Formatierung (00 00 00 00)
$20-$21:	ProDOS-Versionen (00 00)
$22:	Zugriffsbefugnis ($C3)
$23:	Länge jedes Eintrages ($27 = dezimal 39)
$24:	Anzahl der Einträge pro Block ($0D = dezimal 13)
$25-$26:	Gesamtzahl aller aktiven Einträge (00 00)
$27-$28:	Volume Bit Map Zeiger (06 00 = $0006)
$29-$2A:	Gesamtzahl der Blocks der Diskette (18 01 = $0118 = dezimal 280)

Byte $00-$01: Rückwärts-Pointer (LL HH)

Pointer zum vorangehenden Volume-Directory-Block. Da dies der erste Directory-Block ist, ist der Pointer 00 00 und damit leer.

Byte $02-$03: Vorwärts-Pointer (LL HH)

Pointer zum nächsten Volume-Directory-Block, also zum Block 03 00 = $0003.

Byte $04: Speichertyp und Namenslänge (Storage Type und Name Length)

Das Byte dieser Speicherstelle teilt sich in linkes und rechtes Nibble (Halbbyte):

a) Speichertyp (linkes Nibble)

Es gibt insgesamt folgende Speichertypen:

$0: nicht-aktiver, d.h. entweder gelöschter oder noch nicht angelegter Catalog-Eintrag.

$1: „Seedling“ – Sämling-Dateieintrag: Eine Sämling-Datei ist $0000 bis $0200 (dezimal 512) Bytes lang, nimmt also maximal 1 Datenblock ein. Eine Sämling-Datei hat keinen Index-Block, sondern nur einen Zeiger zum Datenblock in Verbindung mit dem Storage Type $1. (Im „Technical Manual“, S. 155, steht irrtümlicherweise,

daß eine Sämling-Datei kleiner/gleich 256 Bytes sei.)

$2: ,,Sapling" – Schößling-Dateieintrag: Eine Schößling-Datei ist $0201 bis $20000 (dezimal 131.072) Bytes lang, nimmt also maximal 256 Datenblocks ein. 256 Blocks · 512 Bytes = 131.072 Bytes = 128K. Eine Schößling-Datei hat 1 Index-Block, der der TSL unter DOS 3.3 entspricht.

$3: ,,Tree" – Baum-Dateieintrag: Eine Baum-Datei ist $020001 bis $1000000 (dezimal 16.777.216) Bytes = 16 Megabytes lang und nimmt (praktisch) maximal 32768 Datenblocks ein. Eine Baum-Datei hat 1 Master-Index-Block sowie 128 Index-Blocks, wobei jeder Index-Block 256 Datenblocks verwaltet. 128 Index-Blocks · 256 Datenblocks · 512 Bytes = 16.777.216 Bytes = 16M = 1024K = ,,K · K" Bytes. (1K = 1024 Bytes).
Theoretisch wären 256 Index-Blocks in einem Master-Index-Block unterzubringen, aber die maximale Dateilänge ist auf 16M und nicht 32M begrenzt. Ferner liegt die maximale Dateigröße bei exakt 16.777.216 minus 1 = 16777.215 Bytes. Aber wer wollte schon bei dieser Größenordnung um ein einziges Byte feilschen!

$D: Subdirectory-Dateieintrag im Volume-Directory

$E: Subdirectory-Kopf

$F: Volume-Directory-Kopf

b) Namenslänge (rechtes Nibble)

Da nur vier Bits zur Verfügung stehen, kann ein Name – Volume-Name, Subdirectory-Name, File-Name – maximal 15 Zeichen lang sein.

In unserem Block-Dump steht als viertes Byte $F6 und besagt mithin, daß es sich um den Volume-Directory-Kopf handelt (= $F) und daß der Volume-Name 6 Bytes lang ist ($6).

Byte $05-$13: Dateiname (File-Name)

Hier steht der maximal 15 Zeichen lange Name, und zwar im Volume-Directory-Kopf der Volume-Name ,,VOLUME", der nur 6 Bytes einnimmt und dessen restliche Bytes Nullen enthalten. DOS-Programmierer werden bemerken, daß der Name mit Bit 7 off abgespeichert ist, während beim DOS-Catalog die Namen mit Bit 7 on eingetragen und im übrigen durch $A0's (= Leertasten) statt Nullen aufgefüllt sind.

Ein Name darf nur Versalbuchstaben (Großbuchstaben), Ziffern und Punkt enthalten und muß mit einem Versal beginnen. Das BASIC.SYSTEM läßt allerdings die Eingabe von kleingeschrieben Befehlen und Dateinamen zu, da diese intern vor dem Diskettenzugriff in Großbuchstaben umgewandelt werden.

Richtige Beispiele:

VOLUME
DATEI.XYZ
PROGRAMM1

Falsche Beispiele:

PRODOS SYSTEM (enthält Leertaste)
Programm1 (enthält Gemeine)
1.PROGRAMM (fängt mit Ziffer an)
APPLEWRITERDATEI (zu lang)

Der Schrägstrich (z.B. CATALOG/USERS.DISK) gehört nicht zum im Volume-Directory eingetragenen Namen.

Nach der Art des Namens lassen sich folgende Typen unterscheiden:

1. File-Name (Dateiname im engeren Sinne)

Dies ist der eigentliche Name einer Programm- oder sonstigen Daten-Datei (z.B. SPIELPROGRAMM1, KALULATIONEN1) mit Storage Type $1, $2 oder $3.

2. Volume-Name (Name des Hauptinhaltsverzeichnisses)

Dies ist der Name des Hauptinhaltsverzeichnisses und steht im Volume-Directory-Kopf mit Storage Type $F.

3. Subdirectory-Name (Name des Unterinhaltsverzeichnisses)

Dies ist der Name eines Unterinhaltsverzeichnisses und steht im Falle eines Unterinhaltsverzeichnisses des 1. Grades erstens im Volume-Directory als File-Name, aber mit Storage Type $D, und zweitens im Key Block des Subdirectory selbst mit Storage Type $E.

Nach der Vollständigkeit des Namens lassen sich folgende Arten unterscheiden:

1. Pathname („Pfadname" = vollständiger Name)

Dies ist der vollständige Name und setzt sich aus dem Volume-Directory-Namen, dem ggf. vorhandenen Subdirectory-Namen sowie dem eigentlichen File-Namen zusammen, wobei jeder Teilname mit einem Schrägstrich beginnt, z.B. entsprechend dem später in Kapitel 1.4.3.1 gelisteten Block-Dump:

/VOLUME/SUBDIRECTORY/SUB.BASIC

Ein (vollständiger) Pathname darf bis zu 64 Zeichen lang sein, Schrägstriche eingeschlossen.

2. Präfix (PREFIX, Teil-Directory-Name, File-Name-Vorspann)

Ein Präfix besteht aus dem Volume-Directory-Namen und ggf. einem Subdirectory-Namen (und ggf. weiteren Subsubdirectory-Namen), abgegrenzt durch Schrägstriche (auch am Ende!), z.B.:

/VOLUME/SUBDIRECTORY/

Ein Präfix kann für sich bis zu 64 Zeichen umfassen.

3. Unvollständiger File-Name (Partial Pathname)

Der unvollständige File-Name ist der eigentliche Name der Datei und sozusagen der zweite Teil des vollständigen Namens, z.B.

SUB.BASIC

Ein Partial Pathname ist nur zulässig, falls zuvor das Präfix definiert wurde, beispielsweise durch das BASIC.SYSTEM:

PREFIX/VOLUME/SUBDIRECTORY
CATALOG/SUB.BASIC

Der eigentliche File-Name kann für sich maximal 64 Zeichen lang sein. Damit können *gesondert* (!) definiertes Präfix *plus* Partial Pathname insgesamt 128 Zeichen ein-

schließlich der Schrägstriche umfassen. Definiert man hingegen direkt den vollständigen Namen, dann ist man auf 64 Bytes beschränkt.

Byte $14-$1B: enthält acht z. Zt. unbenutzte Bytes

Byte $1C-$1F: Formatierungs-Datum und -Uhrzeit

Diese vier Bytes enthalten das Datum und die Uhrzeit zum Zeitpunkt der Formatierung von den Speicherstellen $BF90-$BF93 der PRODOS Global Page (siehe dort). Da unsere formatierte Testdiskette ohne Hardware-Uhr angelegt wurde, sind diese Bytes leer.

Byte $20-$21: Jetzige und frühere ProDOS-Versionen

Diese Stellen enthalten bei ProDOS Version 1.0 zwei Nullen.

Byte $22: Zugriffsbefugnis (Access)

In diesem Byte wird bitmäßig der zulässige Zugriff (Access) definiert, und zwar wie folgt:

7	6	5	4	3	2	1	0	
D	N	B	0	0	0	W	R	
1	1	0	0	0	0	1	1	= $C3 (Beispiel)

„1" bedeutet ja (= zulässig), „0" bedeutet nein (= unzulässig).

Bit 7: Destroy-Bit (Delete-Bit)

Bit 7 besagt, ob die Datei mit dem DELETE-Befehl durch das BASIC.SYSTEM bzw. mit dem DESTROY-Befehl durch das MLI gelöscht werden darf. Bei $C3 ist dies der Fall, doch beachte man, daß nur Files und Subdirectories, und letztere auch nur dann, wenn sie leer sind, gelöscht werden können. Das Volume-Directory kann nur durch Reinitialisierung zerstört werden.

Bit 6: Name kann geändert werden

Bei $C3 ist das Bit gesetzt, folglich ist der RENAME-Befehle zulässig. Volume-Namen ändert man übrigens unter dem BASIC.SYSTEM folgendermaßen:

RENAME/VOLUME.ALT,/VOLUME.NEU

Man beachte das Komma vor dem Schrägstrich.

Bit 5: Backup-Bit

Siehe hierzu das PRODOS Global Page Assemblerlisting.

Bits 4-2: z. Zt. nicht benutzt (000)

Bit 1: Write (Beschreiben) zulässig

Bit 0: Read (Lesen) zulässig

Beides – Read und Write – trifft bei $C3 zu. $C3 ist also das ,,normale'' Access-Byte.

Falls folgende Bits

7	6	5	4	3	2	1	0
1	1	–	–	–	–	1	–

gesetzt sind, gilt der entsprechende File als UNLOCKED. Und falls diese Bits nicht gesetzt sind, gilt der File als LOCKED (im Catalog wie bei DOS 3.3 durch Sternchen ersichtlich). Andere Kombinationen werden als beschränker Zugriff (Restricted Access) bezeichnet. Beispielsweise haben SYS-Files (PRODOS.SYSTEM, BASIC.SYSTEM, FILER, EXERCISER usw.) meist folgendes Bit-Muster beim Access-Byte:

7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	1	= $21

Wenn man dieses Access-Byte in $C3 ändert, lassen sich diese SYS-Files mit dem BLOAD-Befehl ohne TSYS-Parameter (aber mit A-Parameter!) einlesen.

Byte $23: Dateieintrag-Länge (Entry Length)

Hier wird die Länge des Catalog-Eintrages – z. Zt. $27 = dezimal 39 – definiert, d.h. jeder Catalog-Eintrag umfaßt exakt 39 Bytes.

Byte $24: Anzahl der Einträge pro Block (Entries per Block)

Anzahl der Catalog-Einträge pro Block: $0D = dezimal 13. Da das Volume-Directory vier Blocks umfaßt, können maximal 4 · 13 = 52 Einträge gemacht werden. Da jedoch der Volume-Directory-Kopf den ersten Eintrag belegt, verbleiben nur noch 51 mögliche Einträge von Files und Subdirectories.

Byte $25-$26: Anzahl der aktiven Einträge (LL HH; File Count)

Gesamtzahl aller aktiven File-Namen (mit Storage Type ungleich $0) im gesamten Volume-Directory Blocks $0002-$0005, hier bei unserer neuformatierten Diskette 00 00, d.h. $0000 Einträge (da der Volume-Directory-Kopf nicht zählt). Sind alle 51 Einträge belegt, dann steht hier 33 00 = $0033 = 51 File-Namen.

Byte $27-$28: Volume Bit Map Zeiger (LL HH)

Hier wird die Block-Nummer des ersten Blocks der Volume Bit Map abgelegt. Dies ist der Block 06 00 = $0006. Weitere Volume-Bit-Map-Blocks müßten sich direkt an Block 6 anschließen, doch haben 35-Track-Disketten nur einen einzigen Volume-Bit-Map-Block. Näheres siehe weiter unten.

Byte $29-$2A: Gesamtzahl der Blocks der Diskette (LL HH; Total Blocks)

Enthält die Blockgesamtanzahl. Bei den normalerweise 280 Blocks einer Apple-Diskette steht hier 18 01 = $0118 = 280. Theoretisch sind $FFFF = 65535 Blocks möglich.

Die restlichen Bytes $02B-$1FF von Block $0002 sind auf der frisch formatierten Testdiskette zunächst leer.

1.4.2.2. Volume-Directory Non-Key-Blocks

Block $0003: Zweiter Volume-Directory-Block

Bytes $00-$01 enthalten den Zeiger zum vorangehenden Volume-Directory-Block, also 02 00 = $0002 = Block 2.
Bytes $02-$03 enthalten den Zeiger zum nachfolgenden Volume-Directory-Block, also 04 00 = $0004 = Block 4.

Block $0004: Dritter Volume-Directory-Block

Bytes $00-$01 enthalten den Zeiger zum vorangehenden Volume-Directory-Block, also 03 00 = $0003 = Block 3.
Bytes $02-$03 enthalten den Zeiger zum nachfolgenden Volume-Directory-Block, also 05 00 = $0005 = Block 5.

Block $0005: Vierter und zugleich letzter Volume-Directory-Block

Bytes $00-$01 enthalten den Zeiger zum vorangehenden Volume-Directory-Block, also 04 00 = $0004 = Block 4.
Bytes $02-$03 enthalten Nullen, da dies der letzte Volume-Directory-Block ist.

Die übrigen Bytes der Blocks $0003-$0005 unserer Testdiskette sind noch leer.

Ein Volume-Directory umfaßt also unter ProDOS Version 1.0 vier Blocks. Durch Änderung der Zeiger im Block $0005 (04 00 07 00) und in Block $0007 (05 00 00 00) könnte man beispielsweise das Volume-Directory um 1 Block erweitern, doch funktioniert dann der CATALOG-Befehl des BASIC.SYSTEM – nach dem 51. Dateinamen kommt die Meldung „RANGE ERROR" – nicht mehr, obgleich z.B. SAVE und LOAD auch in bezug auf die Dateien ab dem 52. Eintrag möglich ist. Mithin wären weitergehende Patches erforderlich.

1.4.2.3. Volume Bit Map (und System Bit Map)

Unter ProDOS gibt es zwei Arten von Bit Maps, die System Bit Map und die Volume Bit Map. Beide haben sozusagen nur den Namen gemein.

a) System Bit Map

Die System Bit Map, deren Beschreibung dem PRODOS Global Page Assemblerlisting zu entnehmen ist, zeigt an, welche Pages des 48K-RAM-Speichers für PRODOS tabu sind. In derart markierte Tabu-Pages läßt sich beispielsweise mit dem BLOAD-Befehl kein Binärfile laden. Umgekehrt lassen sich aber Tabu-Pages mit dem BSAVE-Befehl auf der Diskette speichern. Allerdings kann man diese Files dann nicht mehr einlesen.

Wenn man unter DOS 3.3 Assemblerprogramme gegen Überschreiben (durch Applesoft-Variablen usw.) schützten will, braucht man nur HIMEM (Highest Memory Location) entsprechend vermindern und das Assemblerprogramm oberhalb von HIMEM einladen. Dieses Verfahren funktioniert unter dem BASIC.SYSTEM nicht mehr. Hinzu kommt, daß HIMEM nunmehr erstens stets auf einer Seitengrenze liegen muß (z.B. HIMEM: 30000 wäre verboten) und daß zweitens das BASIC.SYSTEM entsprechend der benötigten File-Puffer HIMEM bei Bedarf herab- und später wieder heraufsetzt.

Unter ProDOS ist deshalb folgendes Verfahren anzuwenden:

1) Das Assemblerprogramm wird zunächst in einen unteren RAM-Bereich eingeladen, der aufgrund der System Bit Map frei ist.

2) Dem Assemblerprogramm wird ein kurzes Move-Programm vorangestellt, das erstens die Volume Bit Map sowie HIMEM ändert und zweitens das eigentliche Assemblerprogramm in den vorgesehenen Bereich verschiebt.

Das Move-Programm für den „PRODOS.EDITOR" – ein neuer Applesoft-Editor speziell für das ProDOS-Betriebssystem, der über den Hüthig Software Service erhältlich ist – zeigt beispielmäßig, wie ein solches Programm aussehen muß.

b) Volume Bit Map

Die Volume Bit Map befindet sich stets im Block 6 (und ggf. nachfolgenden Blocks, doch gibt es keine Zeiger zum vorangehenden respektive nachfolgenden Volume-Bit-Map-Block). Bei einer 35-Track-Diskette werden lediglich die ersten 35 Bytes des Volume-Bit-Map-Blocks belegt, wobei jedes der 8 Bits eines Bytes für einen Block steht. Ist ein Bit gesetzt (= 1), gilt der Block als belegt. Ist umgekehrt ein Bit zurückgesetzt (= 0), ist der Block wieder frei.

Byte $00 = Spur $00 = Blocks $0000-$0007
Byte $01 = Spur $01 = Blocks $0008-$000F
usw. bis
Byte $22 = Spur $22 = Blocks $0110-$0117

Betrachten wir die ersten 2 Bytes der Volume Bit Map unserer Testdiskette als Bit-Muster (High Bits und Low Bits sind hier vertauscht):

Byte $00								Byte $01								
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	$01 $FF

Da die Blocks $00-$06 (= 7 Blocks) durch den Urlader, das Volume-Directory sowie die Volume Bit Map selbst belegt sind, sind die ersten 7 Bits auf 0 zurückgesetzt. Ab Block $0008-$0117 ist die frisch formatierte Diskette noch frei und sinngemäß sind alle restlichen Bits der Volume Bit Map auf 1 gesetzt.

Das bereits im vorangehenden Kapitel gelistete Initialisierungsprogramm „PRO-DOS.INIT" formatiert nicht nur die gewünschte Anzahl an Spuren, sondern legt auch in der modfizierten Catalog-Spur eine entsprechend erweiterte Volume Bit Map an.

Der Volume-Bit-Map-Block umfaßt 512 Bytes, wobei je Byte 8 Blocks verschlüsselt werden können. Mithin kann ein einziger Volume-Bit-Map-Block 8 · 512 = 4096 Blocks bzw. 8 · 512 · 512 = 2.097.152 Bytes = 2 Megabytes verwalten. Eine 5M-Profile benötigt 3 Volume-Bit-Map-Blocks.

```
BLOCK: $0000

$1000  .8O.L2!.CI...)pJ  01 38 B0 03 4C 32 A1 86 43 C9 03 08 8A 29 70 4A
$1010  JJJ.$.I ..H(HIHP  4A 4A 4A 09 C0 85 49 A0 FF 84 48 28 C8 B1 4B D0
$1020  :O.)....f=%IH)ÄH  3A B0 0E A9 03 8D 00 08 E6 3D A5 49 48 A9 5B 48
$1030  '.$.H c1H...H$kP  60 85 40 85 48 A0 63 B1 48 99 94 09 C8 C0 EB D0
$1040  v".<..=$..r.=+..  F6 A2 06 BC 1D 09 BD 24 09 99 F2 09 BD 2B 09 9D
$1050  ..J.n)..I). .IyO  7F 0A CA 10 EE A9 09 85 49 A9 86 A0 00 C9 F9 B0
$1060  /.H.'.J.L.N.GH.B  2F 85 48 84 60 84 4A 84 4C 84 4E 84 47 C8 84 42
$1070  H.F)..a.K ..Ohfa  C8 84 46 A9 0C 85 61 85 4B 20 12 09 B0 68 E6 61
$1080  fafF%FI..o-.....  E6 61 E6 46 A5 46 C9 06 90 EF AD 00 0C 0D 01 0C
$1090  Fm).P.%J.m#.(..f  D0 6D A9 04 D0 02 A5 4A 18 6D 23 0C A8 90 0D E6
$10A0  K%KJO.I.pU ..J-.  4B A5 4B 4A B0 06 C9 0A F0 55 A0 04 84 4A AD 02
$10B0  .).(1JY..PÄ..v)p  09 29 0F A8 B1 4A D9 02 09 D0 DB 88 10 F6 29 F0
$10C0  I P; .1JI.P3H1J.  C9 20 D0 3B A0 10 B1 4A C9 FF D0 33 C8 B1 4A 85
$10D0  FH1J.G)..J ..K.a  46 C8 B1 4A 85 47 A9 00 85 4A A0 1E 84 4B 84 61
$10E0  H.M ..O.fafa$N+N  C8 84 4D 20 12 09 B0 17 E6 61 E6 61 A4 4E E6 4E
$10F0  1J.F1L.G.JPgL. L  B1 4A 85 46 B1 4C 85 47 11 4A D0 E7 4C 00 20 4C
$1100  ?.&PRODOS         3F 09 26 50 52 4F 44 4F 53 20 20 20 20 20 20 20
$1110    %'.D%a.E1H...$  20 20 A5 60 85 44 A5 61 85 45 6C 48 00 08 1E 24
$1120  ?EGvtWQ6K4,&+.'L  3F 45 47 76 F4 D7 D1 B6 4B B4 AC A6 2B 18 60 4C
$1130  <.).H).H).".Lyt   BC 09 A9 9F 48 A9 FF 48 A9 01 A2 00 4C 79 F4 20
$1140  Xß .9P.....wLM.  58 FC A0 1C B9 50 09 99 AE 05 88 10 F7 4C 4D 09
$1150  *** UNABLE TO LO  AA AA AA A0 D5 CE C1 C2 CC C5 A0 D4 CF A0 CC CF
$1160  AD PRODOS ***%S)  C1 C4 A0 D0 D2 CF C4 CF D3 A0 AA AA AA A5 53 29
$1170  .*.+*=.§).".JPüi  03 2A 05 2B AA BD 80 C0 A9 2C A2 11 CA D0 FD E9
$1180  .Pw&+'%F).I.)...  01 D0 F7 A6 2B 60 A5 46 29 07 C9 04 29 03 08 0A
$1190  (*.=%GJ%FjJJ.A..  28 2A 85 3D A5 47 4A A5 46 6A 4A 4A 85 41 0A 85
$11A0  @%E.'&+=.§ <.f'f  51 A5 45 85 27 A6 2B BD 89 C0 20 BC 09 E6 27 E6
$11B0  =f=O. <.<.§'%§..  3D E6 3D B0 03 20 BC 09 BC 88 C0 60 A5 40 0A 85
$11C0  S)..T%S.PB@@p.O.  53 A9 00 85 54 A5 53 85 50 38 E5 51 F0 14 B0 04
$11D0  fS..FS8 m.%P. o.  E6 53 90 02 C6 53 38 20 6D 09 A5 50 18 20 6F 09
$11E0  Pc ..R.(8FRpN...  D0 E3 A0 7F 84 52 08 28 38 C6 52 F0 CE 18 08 88
$11F0  pu=.§.ä.........  F0 F5 BD BC C0 10 FB 00 00 00 00 00 00 00 00 00

BLOCK: $0001

$1000  Ln SOS BOOT  1.1  4C 6E A0 53 4F 53 20 42 4F 4F 54 20 20 31 2E 31
$1010    .SOS.KERNEL     20 0A 53 4F 53 2E 4B 45 52 4E 45 4C 20 20 20 20
$1020    SOS KRNLI/O ERR 20 53 4F 53 20 4B 52 4E 4C 49 2F 4F 20 45 52 52
$1030  OR..FILE 'SOS.KE  4F 52 08 00 46 49 4C 45 20 27 53 4F 53 2E 4B 45
$1040  RNEL' NOT FOUND%  52 4E 45 4C 27 20 4E 4F 54 20 46 4F 55 4E 44 25
$1050  .INVALID KERNEL   00 49 4E 56 41 4C 49 44 20 4B 45 52 4E 45 4C 20
$1060  FILE:.........$xX 46 49 4C 45 3A 00 00 0C 00 1E 0E 1E 04 A4 78 D8
$1070  *)w...*ä...§)§.J. A9 77 8D DF FF A2 FB 9A 2C 10 C0 A9 40 8D CA FF
$1080  )..o.".No... -.   A9 07 8D EF FF A2 00 CE EF FF 8E 00 20 AD 00 20
$1090  Pu)..')..a)...)"  D0 F5 A9 01 85 E0 A9 00 85 E1 A9 00 85 85 A9 A2
$10A0  .. >!f')..ff.f.f  85 86 20 BE A1 E6 E0 A9 00 85 E6 E6 86 E6 86 E6
$10B0  f >! .1..'H1..aP  E6 20 BE A1 A0 02 B1 85 85 E0 C8 B1 85 85 E1 D0
$10C0  j%'Pf-1 .b-m .c.  EA A5 E0 D0 E6 AD 6C A0 85 E2 AD 6D A0 85 E3 18
$10D0  %ci..e8%bm#$.d%e  A5 E3 69 02 85 E5 38 A5 E2 ED 23 A4 85 E4 A5 E5
$10E0  i..e .1b).M. P!(  E9 00 85 E5 A0 00 B1 E2 29 0F CD 11 A0 D0 21 A8
$10F0  1bY. P..Pv .1b)p  B1 E2 D9 11 A0 D0 19 88 D0 F6 A0 00 B1 E2 29 F0
$1100  I p>Ipp..d  .LT!  C9 20 F0 3E C9 F0 F0 08 AE 64 A0 A0 13 4C D4 A1
$1110  .%bm#$.b%ci..c%d  18 A5 E2 6D 23 A4 85 E2 A5 E3 69 00 85 E3 A5 E4
$1120  Eb%ec0<.%dm#$.b  C5 E2 A5 E5 E5 E3 B0 BC 18 A5 E4 6D 23 A4 85 E2
$1130  %ei..cFfP..O  .L  A5 E5 69 00 85 E3 C6 E6 D0 95 AE 4F A0 A0 1B 4C
$1140  T! .1b.'H1b.a-f  D4 A1 A0 11 B1 E2 85 E0 C8 B1 E2 85 E1 AD 66 A0
$1150  ..-g .. >!-h ..-  85 85 AD 67 A0 85 86 20 BE A1 AD 68 A0 85 85 AD
$1160  i ..-...'-...a >  69 A0 85 86 AD 00 0C 85 E0 AD 00 0D 85 E1 20 BE
$1170  !".=..Ü! p..d  .  A1 A2 07 BD 00 1E DD 21 A0 F0 08 AE 64 A0 A0 13
$1180  LT!J.m)..gfgf.f.  4C D4 A1 CA 10 ED A9 00 85 E7 E6 E7 E6 86 E6 86
$1190  &g=...'=...a%'P.  A6 E7 BD 00 0C 85 E0 BD 00 0D 85 E1 A5 E0 D0 04
$11A0  %ap. >!L.!.-j m.  A5 E1 F0 06 20 BE A1 4C 8A A1 18 AD 6A A0 6D 08
$11B0  ..h-k m...ilh.).  1E 85 E8 AD 6B A0 6D 09 1E 85 E9 6C E8 00 A9 01
$11C0  ..%'&a ytO.'.2    85 87 A5 E0 A6 E1 20 79 F4 B0 01 60 AE 32 A0 A0
$11D0  .LT!.g8)(egJ.eg(  09 4C D4 A1 84 E7 38 A9 28 E5 E7 4A 18 65 E7 A8
$11E0  =) .'.J.FgPt-§§L  BD 29 A0 99 A7 05 CA 88 C6 E7 D0 F4 AD 40 C0 4C
$11F0  o!.............   EF A1 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

BLOCK: $0002

$1000	....vVOLUME.....	00	00	03	00	F6	56	4F	4C	55	4D	45	00	00	00	00	00
$1010	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1020	..C'.....	00	00	C3	27	0D	00	00	06	00	18	01	00	00	00	00	00
$1030	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1040	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1050	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1060	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1070	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1080	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1090	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10A0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10B0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10C0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10D0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10E0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10F0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1100	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1110	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1120	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1130	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1140	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1150	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1160	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1170	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1180	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1190	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11A0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11B0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11C0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11D0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11E0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11F0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

BLOCK: $0003

$1000	.....	02	00	04	00	00	00	00	00	00	00	00	00	00	00	00	00
$1010	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1020	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1030	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1040	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1050	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1060	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1070	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1080	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1090	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10A0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10B0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10C0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10D0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10E0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$10F0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1100	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1110	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1120	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1130	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1140	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1150	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1160	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1170	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1180	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$1190	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11A0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11B0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11C0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11D0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11E0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
$11F0	.....	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

```
BLOCK: $0004

$1000  ................  03 00 05 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


BLOCK: $0005

$1000  ................  04 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BLOCK: $0006

$1000  ................  01 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1010  ................  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1020  ................  FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Hätten Sie's gewußt?

Das Booten dauert bei ProDOS 10 Sekunden und bei DOS 3.3 5 Sekunden. Der Programmwechsel ist also bei DOS 3.3 doppelt so schnell.

```
                        1               ORG   $3FA5
                        2    *
                        3    * PRODOS-EDITOR
                        4    * =============
                        5    *
                        6    * Applesoft-Editor unter ProDOS
                        7    * (c) 1984 U.Stiehl, Heidelberg
                        8    *
                        9    * Editor von $4000 nach $8400
                       10    * verschieben und Bit-Map ändern
                       11    *
                       12    * HIMEM:   $8000
                       13    * PUFFER:  $8000-$83FF
                       14    * EDITOR:  $8400-$99FF
                       15    * PRODOS:  $9A00-$BFFF
                       16    *
3FA5: A9 0F            17              LDA   #%00001111 ;$8000-
3FA7: 8D 68 BF         18              STA   $BF68      ;$87FF
                       19    *
3FAA: A9 FF            20              LDA   #%11111111 ;$8800-
3FAC: 8D 69 BF         21              STA   $BF69      ;$8FFF
                       22    *
                       23    *                        ;$9000-
3FAF: 8D 6A BF         24              STA   $BF6A      ;$97FF
                       25    *
                       26    *                        ;$9800-
3FB2: 8D 6B BF         27              STA   $BF6B      ;$9FFF
                       28    *
3FB5: A9 00            29              LDA   #$00       ;HIMEM:
3FB7: 85 73            30              STA   $73
3FB9: A9 80            31              LDA   #$80
3FBB: 85 74            32              STA   $74        ;$8000
3FBD: A0 00            33              LDY   #$00
3FBF: 84 CE            34              STY   $CE
3FC1: 84 FE            35              STY   $FE
3FC3: A9 40            36              LDA   #$40       ;$4000
3FC5: 85 CF            37              STA   $CF
3FC7: A9 84            38              LDA   #$84       ;$8400
3FC9: 85 FF            39              STA   $FF
3FCB: A2 9A            40              LDX   #$9A       ;<$9A00
                       41    *
                       42    * $CE-CF: $4000
                       43    * $FE-FF: $8400
                       44    *
3FCD: B1 CE            45    MOVER     LDA   ($CE),Y
3FCF: 91 FE            46              STA   ($FE),Y
3FD1: C8               47              INY
3FD2: D0 F9            48              BNE   MOVER
3FD4: E6 CF            49              INC   $CF
3FD6: E6 FF            50              INC   $FF
3FD8: E4 FF            51              CPX   $FF
3FDA: D0 F1            52              BNE   MOVER
3FDC: 4C 00 84         53              JMP   $8400      ;Editor
```

1.4.3. Dateieinträge im Volume-Directory

Hinweis: Zu dem Kapitel 1.4.3 gehören die ,,Dumps zu Kap. 1.4.3'': Blocks $0002-$0005, die auf den Seiten 128-129 abgebildet sind.

Wir haben bislang nur den Kopf des Volume-Directory besprochen. Als nächstes beschreiben wir unsere Testdiskette mit folgendem Applesoft-Programm:

```
10 FOR X = 1 TO 51: PRINT CHR$ (4) ,,SAVE APPLESOFT''; X: NEXT
```

Damit speichert sich das Programm selbst 51 mal unter den Namen APPLESOFT1, APPLESOFT2 usw. bis APPLESOFT51 ab. Die vier Block-Dumps der Blocks $0002-$0004 enthalten das dadurch modifizierte Volume-Directory.

Wir haben oben bereits festgestellt, daß 1 Volume-Directory-Block 13 Einträge umfassen kann, daß jeder Eintrag 39 Bytes lang ist und daß jeder Block mit 4 zusätzlichen Bytes (Rückwärts- und Vorwärts-Pointer) beginnt. Damit findet man jeden beliebigen X-ten Dateieintrag mit folgender Formel:

X-ter Dateieintrag = 4 + X · 39

Für den fünften Eintrag gilt z.B. 4 + 5 · 39 = 199, d.h. der fünfte Eintrag beginnt ab der relativen Blockposition 199 = $0C7 in Block $0002, wo wir das Byte $1A vorfinden. ,,1'' steht für Sämling-Datei und ,,A'' steht für 10 Zeichen Namenslänge (,,APPLESOFT5''). Für den zwölften Eintrag gilt z.B. 4 + 12 · 39 = 472 = $1D8, wo wir das Byte $1B vorfinden. ,,1'' ist bereits bekannt. ,,B'' steht jetzt für 11 Zeichen Namenslänge (,,APPLESOFT12''). Betrachten wir abschließend noch den Eintrag für ,,APPLESOFT13''. Da im ersten Block nur 12 echte File-Einträge möglich waren – denn der erste Eintrag ist der Volume-Directory-Kopf –, ist ,,APPLESOFT13'' der nullte Eintrag in Block $0003, d.h. die obige Formel bezieht sich auf die Numerierung 0-12 und nicht 1-13. Dann gilt 4 + 0 · 39 = relative Position 4 = $004, wo wieder $1B steht.

Wenn man 4 + 13 · 39 ausrechnet, kommt man zum letzten relativen Byte $1FF, das bei jedem Volume-Directory-Block unbenutzt bleibt.

1.4.3.1. Struktur des Dateieintrages

Zunächst erfolgt eine Kurzzusammenfassung der Inhalte der relativen Bytes und dann werden die Speicherstellen im einzelnen beschrieben, soweit sie uns nicht schon

vom Volume-Directory-Kopf her bekannt sind. Die bei der Zusammenfassung in Klammern hinzugefügten Beispiel-Werte beziehen sich auf den Dump von Block $0002 = erster Volume-Directory-Block ab absolutem Byte $002B, wo sich der Dateieintrag von ,,APPLESOFT1'' befindet. Die absolute Byte-Position $002B ist zugleich die relative Byte-Position $00, da der Dateieintrag ab dieser Stelle beginnt.

Zusammenfassung Dateieintrag

$00:	Speichertyp und Namenslänge ($1A = Seedling $1, Länge $A)
$01-$0F:	Dateiname (,,APPLESOFT1.....'')
$10:	Dateityp ($FC = BAS)
$11-$12:	Hauptzeiger zum Dateianfang (07 00 = Block $0007)
$13-$14:	Anzahl der belegten Blocks (00 01 = $0001 Block)
$15-$17:	Dateilänge (EOF-Pointer) (29 00 00 = $000029 Bytes)
$18-$1B:	Datum und Uhrzeit der Dateianlage (00 00 00 00)
$1C-$1D:	ProDOS-Versionen (00 00)
$1E:	Zugriffsbefugnis ($E3 = 11100011, Backup-Bit gesetzt!)
$1F-$20:	Zusatzinfo zum Dateityp (01 08 = Startadresse $0801)
$21-$24:	Datum und Uhrzeit der letzten Änderung (00 00 00 00)
$25-$26:	Pointer zum Volume- oder Subdirectory-Kopf (02 00 = Block $0002)

$00: Speichertyp und Namenslänge: bereits bekannt

Wenn eine Datei mit DELETE gelöscht wird, dann wird dieses Byte auf Null gesetzt, jedoch der übrige Dateieintrag unverändert gelassen. Zusätzlich wird jedoch der Inhalt des entsprechenden Index-Blocks gelöscht, so daß ein UNDELETE später kaum noch möglich ist! (Anders bei DOS 3.3, wo die TSL-Sektoren durch DELETE nicht berührt werden.)

$01-$0F: Dateiname: bereits bekannt

$10: Dateityp (File Type)

Hier wird der Dateityp als Hex-Zahl verschlüsselt. Die reinen SOS-Dateitypen – $01-$03, $05, $07-$0E, $10-$BF – interessieren uns nicht und seien nur der Vollständigkeit halber erwähnt. Der File Type ist nicht mit dem Storage Type zu verwechseln. ProDOS-Dateitypen:

$00:		Typenloser File
$04:	TXT	ASCII-Textfile
$06:	BIN	Binärfile
$0F:	DIR	Subdirectory File
$C0-$EF:		reserviert für ProDOS
$F0:	CMD	Befehls-Datei
$F1-$F8:		benutzerdefinierte Filetypen
$F9:		reserviert für ProDOS
$FA:	INT	Integer-Basic-File
$FB:	IVR	Integer-Basic-Variablendatei
$FC:	BAS	Applesoft-File
$FD:	VAR	Applesoft-Variablendatei
$FE:	REL	Relokativer Objektcode (EDASM-Assembler)
$FF:	SYS	System-File

Anmerkungen zu den Dateitypen:

1. Ein Teil der Dateitypen gibt es nur dem Namen nach. Beispielsweise ist Integer-Basic weder unter ProDOS lauffähig noch existieren die INT- und IVR-Files. Integer-Basic dürfte unter ProDOS mit großer Wahrscheinlichkeit nicht mehr von der Firma Apple implementiert werden, so daß diese Dateitypen mehr ,,kosmetischer'' Natur sind.

2. Dateitypen, die dem BASIC.SYSTEM bekannt sind, werden beim CATALOG-Befehl aufgrund der im BASIC.SYSTEM enthalten Kurzworttabelle als 3stelliges Kürzel angezeigt: TXT, BAS, SYS, BIN usw. Bei unbekannten Dateitypen wird lediglich die entsprechende Hex-Zahl ausgewiesen.

3. Der Dateityp TXT entspricht dem DOS 3.3 Textfile ,,T'' mit einem entscheidenden Unterschied: Das BASIC.SYSTEM speichert Textfiles stets mit Bit 7 off, während unter DOS 3.3 Textfiles mit Bit 7 on gespeichert werden. Beispielsweise findet man nach Ablauf des folgenden Demos

```
10 PRINT CHR$ (4) ,,OPEN TEXT'' : PRINT CHR$ (4) ,,WRITE TEXT''
20 PRINT ,,ABC'' : PRINT CHR$ (4) ,,CLOSE''
```

den String ,,ABC + Return''

auf der ProDOS-Diskette als 41 42 43 0D
und auf der DOS-Diskette als C1 C2 C3 8D.

Das CONVERT-Programm von der ,,User's Disk'' nimmt automatisch die entsprechenden Umwandlungen vor. Die im ,,DOS-Buch'' geschilderten Textfile-Zugriffsverfahren mit COUT und RDKEY sind unter dem BASIC.SYSTEM nicht mehr möglich! Erstens verwandelt das BASIC.SYSTEM bei COUT jedes Zeichen automatisch in den entsprechenden ASCII-Code mit Bit 7 off, gleichviel ob es z.B. LDA #$A0 JSR COUT oder LDA #$20 JSR COUT heißt. Zweitens bricht das BASIC.SYSTEM zusammen, wenn man versucht, einen Textfile mit RDKEY einzulesen.

Bei Random-Access-Textfiles findet sich die Record-Länge im Zusatzinfo-Feld.

4. Der Dateityp BIN entspricht dem DOS 3.3 Binärfile ,,B''. Beim DOS-Binärfile wird die Startadresse und die Länge als je 2 Bytes im ersten Datensektor des Files mit abgespeichert. Bei ProDOS-Binärfiles findet sich die Startadresse in dem Zusatzinfo-Feld und die Länge in dem Dateilänge-Feld, die beide Bestandteil des Dateieintrages sind und deshalb nicht etwa wie bei DOS im ersten Datenblock resp. Datensektor enthalten sind.

5. Der Dateityp BAS entspricht dem DOS 3.3 Applesoft-File ,,A''. Die Länge des ProDOS-Applesoft-Files steht auch hier nicht im ersten Datenblock, sondern in dem Dateilänge-Feld. Bei DOS-Applesoft-Files findet sich im ersten Datensektor nur die Länge, nicht aber die Startadresse. Bei ProDOS-Applesoft-Files wird demgegenüber auch die Startadresse gespeichert, und zwar wieder in dem Zusatzinfo-Feld.

6. Der Dateityp VAR ist neu. Gemeint ist eine Applesoft-Variablendatei, die mit dem STORE-Befehl auf der Diskette – nicht auf der Kassette! – als Speicherauszug aller Variablen erzeugt wird. Mit dem RESTORE-Befehl können diese Variablen dann wieder eingelesen werden. Demo:

```
10 V = 1.23456789
20 V$ = ,,ABCDEFGHIJ''
30 PRINT V, V$
40 PRINT CHR$ (4) ,,STORE VARIABLEN''
50 CLEAR
60 PRINT V, V$
70 PRINT CHR$ (4) ,,RESTORE VARIABLEN''
80 PRINT V, V$
```

In Zeile 40 werden mit STORE VARIABLEN die Inhalte oder Wertzuweisungen von V und V$ sowie die Variablennamen selbst einschließlich entsprechender

Pointer unter dem Namen „VARIABLEN" auf der Diskette als VAR-File gespeichert und in Zeile 70 wieder eingelesen. Die Befehle STORE und RESTORE sind nützlich, wenn *alle* Variablen gesichert werden sollen.

Das Zusatzinfo-Feld enthält übrigens die Startadresse des Speicherraums im RAM, in dem die Variablen vor der Diskettenspeicherung zunächst zwischengespeichert werden; dieser liegt unmittelbar unterhalb des BASIC.SYSTEM-Puffers. Bei dem obigen Demo würde unter der Voraussetzung von HIMEM: $9600 das Zusatzinfo-Feld mit der Adresse $95E8 belegt.

Der Vorteil des VAR-Files besteht in der relativ schnellen Sicherung aller Variablen. Nachteilig ist, daß auch Hilfsvariablen gespeichert werden, die man normalerweise nicht speichern würde. Ein VAR-File ist teils größer und teils kleiner als der entsprechende TXT-File. Strings nehmen als VAR-File ca. 25% mehr Raum auf der Diskette ein. Zahlen lassen sich als VAR-File dann erheblich kompakter speichern, wenn es sich um echte Fließkommazahlen aus mathematischen Berechnungen handelt, also Zahlen mit einer neunstelligen Mantisse, z.B. Sinus-Werte. Kaufmännische Zahlen nehmen hingegen als VAR-File meist mehr Raum ein. Ein VAR-File enthält die Fließkommazahlen in der gepackten Applesoft-internen Speicherform und nicht wie beim üblichen TXT-File als Strings.

7. Der Dateityp SYS ist ebenfalls neu. Ein System-File ist im Prinzip ein Binärfile mit beschränkter Zugriffsbefugnis und und gleichzeitiger Bootfähigkeit, d.h. er kann nicht mit BLOAD ohne TSYS-Parameter eingeladen, sondern muß gebootet bzw. mit dem Strichbefehl („-PRODOS") gestartet werden. Wird er hingegen durch Änderung des Access-Bytes in einen Binärfile umgewandelt, dann ist er nicht mehr bootfähig.

8. Der ebenfalls neue Dateityp DIR – gemeint ist stets ein Subdirectory oder Subsubdirectory, da das Volume-Directory kein Dateityp ist – wird im Kapitel 1.4.4 erläutert.

Byte $11-$12: Hauptzeiger (LL HH; Key Pointer)

Der Key Pointer ist der erste oder Ausgangszeiger zur Datei. Bei einer Sämling-Datei zeigt er direkt auf den ersten und einzigen Datenblock. Bei einer Schößling-Datei zeigt er auf den ersten und einzigen Index-Block, der seinerseits Zeiger zu den einzelnen Datenblocks enthält. Bei einer Baum-Datei zeigt er auf den ersten und einzigen Master-Index-Block, der seinerseits Zeiger zu den einzelnen Index-Blocks enthält, die ihrerseits Zeiger zu den eigentlichen Datenblocks enthalten. Näheres zu

diesem komplizierten Thema im Kapitel 1.4.5.
Im Falle eines Subdirectory-Eintrages zeigt der Key Pointer auf den Key Block = ersten Block des Subdirectory.

Byte $13-$14: Anzahl der belegten Blocks (LL HH)

Hier wird die Summe aus Datenblocks und Index-Blocks und ggf. vorhandenem Master-Index-Block gespeichert. Eine Sämling-Datei belegt 1 Block, eine Schößling-Datei mindestens 3 Blocks (1 Index-Block plus 2 Datenblocks) und höchstens 257 Blocks (1 Index-Block plus 256 Datenblocks). Eine Baum-Datei belegt mindestens 260 Blocks (1 Master-Index-Block, wenigstens 2 Index-Blocks und wenigstens 257 Datenblocks).
Die Anzahl der belegten Blocks liegt im Bereich $0001-$FFFF (dezimal 1-65535).

Byte $15-$17: Dateilänge, EOF = End of File = Endfile, LL MM HH

Die Dateilänge wird als 3 Bytes umfassende hexadezimale Zahl verschlüsselt: Low Byte – Middle Byte – High Byte. Damit kann die Dateilänge im Bereich $000001-$FFFFFF liegen. Nehmen wir an, ein Textfile habe die Länge

A0 10 01 = $0110A0
LL MM HH

Dann rechnen wir wie folgt um:

(160 · 1) + (16 · 256) + (1 · 65536) = 69.792 Bytes.

Allgemein gilt: (LL · 1) + (MM · 256) + (HH · 65536)

Unter EOF versteht man erstens den Zeiger zu demjenigen gedachten Byte, das dem Ende der Datei folgen würde (Endmarker), sowie die Gesamtlänge der Datei selbst. Beispiel: Der Textfile ,,ABCDEF" plus Return stellt sich auf der Diskette so dar:

41 42 43 44 45 46 0D 00 00 00
00 01 02 03 04 05 06 07 08 09 Zeiger auf nulltes, erstes, zweites ... Byte
01 02 03 04 05 06 07 08 09 0A Länge eins, zwei, drei ... Bytes

In der ersten Reihe steht der Inhalt des Datenblocks. In der zweiten Reihe stehen Ordinalzahlen, beginnend mit ,,Nulltes": 0., 1., 2., 3... Und in der dritten Reihe

finden sich Kardinalzahlen, beginnend mit ,,Eins'': 1, 2, 3, 4... .Das EOT wäre der Zeiger zu dem Byte, das auf $0D folgt oder folgen würde. Geht der Block ,,glatt'' auf, dann folgt physisch kein Byte mehr. Geht er nicht glatt auf, dann ist der Block-Rest meist mit Nullen (hexadezimal $00) aufgefüllt. Die erste Null nach $0D ist dann der Endmarker. ProDOS benötigt keinen physischen, sondern nur einen logischen, d.h. gedachten, imaginären Endmarker.
Bei dem Beispiel ist der EOF $07: Die Datei ist 7 Bytes lang und das 7. Byte ist die – vom Beginn der Datei aus gerechnet – absolute Position des Endmarkers.

Byte $18-$1B: Datum und Uhrzeit, als die Datei erstmals angelegt wurde

Byte $1C-$1D: Gegenwärtige und frühere ProDOS-Versionen (z. Zt. 00 00)

Byte $1E: Zugriffsbefugnis (Access, bereits bekannt)

Das übliche Access-Byte für UNLOCKED-Dateien ist entweder $C3 = 11000011 (Backup-Bit nicht gesetzt) oder $E3 = 11100011 (Backup-Bit gesetzt). Bei den neuangelegten ,,APPLESOFT1'' usw. Dateien in den Block-Dumps ist das Backup-Bit gesetzt.

Byte $1F-$20: Zusatzinfo (Hilfstyp, Aux Type, Auxiliary Type, Subtype)

Diese 2 Bytes – im ,,Technical Manual'' als Aux Type, im Catalog als Subtype und von uns als Zusatzinfo bezeichnet – sind bei einigen Dateitypen unbenutzt und bei anderen enthalten sie Zusatzinformationen unterschiedlicher Art:

Beim Dateityp TXT wird hier in der üblichen Form LL HH (low byte first) die Recordlänge von Random-Files abgelegt, z.B. 00 A0 = $00A0 = dezimal 160 nach der Applesoft-Programmzeile:

10 PRINT CHR$ (4) ,,OPEN RANDOM, L 160''

Bei Nicht-Random-Textfiles, d.h. bei sequentiellen Textfiles ist das Zusatzinfo-Feld leer.

Bei den Dateitypen BIN und BAS wird hier die Startadresse LL HH gespeichert.

Byte $21-$24: Datum und Uhrzeit der letzten Dateiänderung

Byte $25-$26: Kopfzeiger (LL HH; Header Pointer)

Hier wird der Zeiger zum Key Block (Kopf-Block) des Volume-Directory oder – falls die Datei in einem Subdirectory aufgeführt wird – zum Key Block des Subdirectory aufgeführt.

1.4.4. Subdirectory

Hinweis: Zu dem Kapitel 1.4.4 gehören die ,,Dumps zu Kap. 1.4.4'': Blocks $0002, $0007-$0009, die auf den Seiten 130-131 abgebildet sind.

Wir formatieren zunächst unsere Testdiskette mit dem ,,FILER'' neu unter dem Volume-Namen ,,VOLUME'' und verfahren dann mit unserer frischen Testdiskette wie folgt:

```
PREFIX/VOLUME

10 HOME
20 PRINT ,,MAIN.BASIC''

SAVE MAIN.BASIC

CREATE/VOLUME/SUBDIRECTORY
PREFIX/VOLUME/SUBDIRECTORY

10 HOME
20 PRINT ,,SUB.BASIC''

SAVE SUB.BASIC
```

Damit erzeugen wir einen Dateieintrag für das Applesoft-Programm ,,MAIN.BASIC'' sowie einen Dateieintrag für das Subdirectory ,,SUB-DIRECTORY'' im Key Block des Volume-Directory. Ferner wird der Key Block des Subdirectory ,,SUBDIRECTORY'' angelegt, der als Dateieintrag ,,SUB.BASIC'' für das entsprechende Applesoft-Programm enthält.

Das ,,Spielchen'' kann fortgesetzt werden, indem man ein Subsubdirectory (= Subdirectory zweiten Grades) etwa mit dem Namen ,,SUBSUBDIRECTORY'' erzeugt, dessen Dateieintrag sich im ,,SUBDIRECTORY'' (= Subdirectory ersten Grades) befindet, usw.

Je höher der Subdirectory-Grad, desto länger wird der ,,Pfad'' (Pathname) und desto länger dauert der Diskettenzugriff. Deshalb sollte man Subdirectories nur bei Festplattenlaufwerken, wo sie sinnvoll und notwendig sind, aber möglichst nicht bei Diskettenlaufwerken anlegen, da sonst die Zugriffsgeschwindigkeit dramatisch abnimmt (siehe die Tests im Kapitel 1.2.3.6).

Im Block $0002 hat der Eintrag für ,,SUBDIRECTORY'' den Speichertyp $D für Subdirectory Entry und den Dateityp $0F für Subdirectory File (= DIR). Der Hauptzeiger (Key Pointer) zeigt auf Block $0008, d.h. auf den Key Block des Subdirectory. Der EOF beträgt $000200, da das Subdirectory bei der Testdiskette 1 Block einnimmt.

```
BLOCK: $0002

$1000  ....vVOLUME.....  00 00 03 00 F6 56 4F 4C 55 4D 45 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ..C'.3......APPL  00 00 C3 27 0D 33 00 06 00 18 01 1A 41 50 50 4C
$1030  ESOFT1.....ö....  45 53 4F 46 54 31 00 00 00 00 00 1C 07 00 01 00
$1040  )........c......  29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050  ...APPLESOFT2...  02 00 1A 41 50 50 4C 45 53 4F 46 54 32 00 00 00
$1060  ..ö....).........  00 00 FC 08 00 01 00 29 00 00 00 00 00 00 00 00
$1070  c.........APPLES  E3 01 08 00 00 00 00 02 00 1A 41 50 50 4C 45 53
$1080  OFT3.....ö....).  4F 46 54 33 00 00 00 00 00 FC 09 00 01 00 29 00
$1090  .......c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00
$10A0  .APPLESOFT4.....  1A 41 50 50 4C 45 53 4F 46 54 34 00 00 00 00 00
$10B0  ö....).........c.  FC 0A 00 01 00 29 00 00 00 00 00 00 00 E3 01
$10C0  ........APPLESOF  08 00 00 00 00 02 00 1A 41 50 50 4C 45 53 4F 46
$10D0  T5.....ö....)....  54 35 00 00 00 00 00 FC 0B 00 01 00 29 00 00 00
$10E0  .....c.........A  00 00 00 00 00 E3 01 08 00 00 00 00 02 00 1A 41
$10F0  PPLESOFT6.....ö.  50 50 4C 45 53 4F 46 54 36 00 00 00 00 00 FC 0C
$1100  ...).........c...  00 01 00 29 00 00 00 00 00 00 00 E3 01 08 00
$1110  ......APPLESOFT7  00 00 00 02 00 1A 41 50 50 4C 45 53 4F 46 54 37
$1120  .....ö....).....  00 00 00 00 00 FC 0D 00 01 00 29 00 00 00 00 00
$1130  ...c.........APP  00 00 00 E3 01 08 00 00 00 00 02 00 1A 41 50 50
$1140  LESOFT8.....ö...  4C 45 53 4F 46 54 38 00 00 00 00 00 FC 0E 00 01
$1150  .)........c.....  00 29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00
$1160  ....APPLESOFT9..  00 02 00 1A 41 50 50 4C 45 53 4F 46 54 39 00 00
$1170  ....ö....).......  00 00 00 FC 0F 00 01 00 29 00 00 00 00 00 00 00
$1180  .c.........APPLE  00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45
$1190  SOFT10....ö....)  53 4F 46 54 31 30 00 00 00 00 FC 10 00 01 00 29
$11A0  ........c.......  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 00 02
$11B0  ..APPLESOFT11...  00 1B 41 50 50 4C 45 53 4F 46 54 31 31 00 00 00
$11C0  .ö....).........c  00 FC 11 00 01 00 29 00 00 00 00 00 00 00 00 E3
$11D0  .........APPLESO  01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F
$11E0  FT12....ö....)...  46 54 31 32 00 00 00 00 FC 12 00 01 00 29 00 00
$11F0  ......c.........  00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00 00

BLOCK: $0003

$1000  .....APPLESOFT13  02 00 04 00 1B 41 50 50 4C 45 53 4F 46 54 31 33
$1010  ....ö....).......  00 00 00 00 FC 13 00 01 00 29 00 00 00 00 00 00
$1020  ..c.........APPL  00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C
$1030  ESOFT14....ö....  45 53 4F 46 54 31 34 00 00 00 00 FC 14 00 01 00
$1040  )........c.......  29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050  ...APPLESOFT15..  02 00 1B 41 50 50 4C 45 53 4F 46 54 31 35 00 00
$1060  ..ö....).........  00 00 FC 15 00 01 00 29 00 00 00 00 00 00 00 00
$1070  c.........APPLES  E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53
$1080  OFT16....ö....).  4F 46 54 31 36 00 00 00 00 FC 16 00 01 00 29 00
$1090  .......c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00
$10A0  .APPLESOFT17....  1B 41 50 50 4C 45 53 4F 46 54 31 37 00 00 00 00
$10B0  ö....).........c.  FC 17 00 01 00 29 00 00 00 00 00 00 00 E3 01
$10C0  ........APPLESOF  08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46
$10D0  T18....ö....)....  54 31 38 00 00 00 00 FC 18 00 01 00 29 00 00 00
$10E0  .....c.........A  00 00 00 00 00 E3 01 08 00 00 00 00 02 00 1B 41
$10F0  PPLESOFT19....ö.  50 50 4C 45 53 4F 46 54 31 39 00 00 00 00 FC 19
$1100  ...).........c...  00 01 00 29 00 00 00 00 00 00 00 E3 01 08 00
$1110  ......APPLESOFT2  00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46 54 32
$1120  0....ö....).....  30 00 00 00 00 FC 1A 00 01 00 29 00 00 00 00 00
$1130  ...c.........APP  00 00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50
$1140  LESOFT21....ö...  4C 45 53 4F 46 54 32 31 00 00 00 00 FC 1B 00 01
$1150  .)........c.....  00 29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00
$1160  ....APPLESOFT22.  00 02 00 1B 41 50 50 4C 45 53 4F 46 54 32 32 00
$1170  ...ö....).......  00 00 00 FC 1C 00 01 00 29 00 00 00 00 00 00 00
$1180  .c.........APPLE  00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45
$1190  SOFT23....ö....)  53 4F 46 54 32 33 00 00 00 00 FC 1D 00 01 00 29
$11A0  ........c.......  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 00 02
$11B0  ..APPLESOFT24...  00 1B 41 50 50 4C 45 53 4F 46 54 32 34 00 00 00
$11C0  .ö....).........c  00 FC 1E 00 01 00 29 00 00 00 00 00 00 00 00 E3
$11D0  .........APPLESO  01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F
$11E0  FT25....ö....)...  46 54 32 35 00 00 00 00 FC 1F 00 01 00 29 00 00
$11F0  ......c.........  00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00 00
```

BLOCK: $0004

```
$1000  .....APPLESOFT26  03 00 05 00 1B 41 50 50 4C 45 53 4F 46 54 32 36
$1010  ....ö ...).......  00 00 00 00 FC 20 00 01 00 29 00 00 00 00 00 00
$1020  ..c.........APPL  00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C
$1030  ESOFT27....ö!...  45 53 4F 46 54 32 37 00 00 00 00 FC 21 00 01 00
$1040  ).........c.......  29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050  ...APPLESOFT28..  02 00 1B 41 50 50 4C 45 53 4F 46 54 32 38 00 00
$1060  ..ö"...).........  00 00 FC 22 00 01 00 29 00 00 00 00 00 00 00 00
$1070  c.........APPLES  E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53
$1080  OFT29....ö#...).  4F 46 54 32 39 00 00 00 00 FC 23 00 01 00 29 00
$1090  .......c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00
$10A0  .APPLESOFT30....  1B 41 50 50 4C 45 53 4F 46 54 33 30 00 00 00 00
$10B0  ö$...).......c.  FC 24 00 01 00 29 00 00 00 00 00 00 00 00 E3 01
$10C0  .........APPLESOF  08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46
$10D0  T31....ö%...)...  54 33 31 00 00 00 00 FC 25 00 01 00 29 00 00 00
$10E0  .....c.........A  00 00 00 00 00 E3 01 08 00 00 00 00 02 00 1B 41
$10F0  PPLESOFT32....ö&  50 50 4C 45 53 4F 46 54 33 32 00 00 00 00 FC 26
$1100  ...).........c...  00 01 00 29 00 00 00 00 00 00 00 00 E3 01 08 00
$1110  ......APPLESOFT3  00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46 54 33
$1120  3....ö'...).....  33 00 00 00 00 FC 27 00 01 00 29 00 00 00 00 00
$1130  ...c.........APP  00 00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50
$1140  LESOFT34....ö(..  4C 45 53 4F 46 54 33 34 00 00 00 00 FC 28 00 01
$1150  .).........c.....  00 29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00
$1160  ....APPLESOFT35.  00 02 00 1B 41 50 50 4C 45 53 4F 46 54 33 35 00
$1170  ...ö)...).........  00 00 00 FC 29 00 01 00 29 00 00 00 00 00 00 00
$1180  .c.........APPLE  00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45
$1190  SOFT36....ö*...)  53 4F 46 54 33 36 00 00 00 00 FC 2A 00 01 00 29
$11A0  ........c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 00 02
$11B0  ..APPLESOFT37...  00 1B 41 50 50 4C 45 53 4F 46 54 33 37 00 00 00
$11C0  .ö+...).........c  00 FC 2B 00 01 00 29 00 00 00 00 00 00 00 00 E3
$11D0  .........APPLESO  01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F
$11E0  FT38....ö,...)..  46 54 33 38 00 00 00 00 FC 2C 00 01 00 29 00 00
$11F0  .......c.........  00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00 00
```

BLOCK: $0005

```
$1000  .....APPLESOFT39  04 00 00 00 1B 41 50 50 4C 45 53 4F 46 54 33 39
$1010  ....ö-...).......  00 00 00 00 FC 2D 00 01 00 29 00 00 00 00 00 00
$1020  ..c.........APPL  00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C
$1030  ESOFT40....ö....  45 53 4F 46 54 34 30 00 00 00 00 FC 2E 00 01 00
$1040  ).........c.......  29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050  ...APPLESOFT41..  02 00 1B 41 50 50 4C 45 53 4F 46 54 34 31 00 00
$1060  ..ö/...).........  00 00 FC 2F 00 01 00 29 00 00 00 00 00 00 00 00
$1070  c.........APPLES  E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53
$1080  OFT42....ö0...).  4F 46 54 34 32 00 00 00 00 FC 30 00 01 00 29 00
$1090  .......c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00
$10A0  .APPLESOFT43....  1B 41 50 50 4C 45 53 4F 46 54 34 33 00 00 00 00
$10B0  ö1...).........c.  FC 31 00 01 00 29 00 00 00 00 00 00 00 00 E3 01
$10C0  .........APPLESOF  08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46
$10D0  T44....ö2...)....  54 34 34 00 00 00 00 FC 32 00 01 00 29 00 00 00
$10E0  .....c.........A  00 00 00 00 00 E3 01 08 00 00 00 00 02 00 1B 41
$10F0  PPLESOFT45....ö3  50 50 4C 45 53 4F 46 54 34 35 00 00 00 00 FC 33
$1100  ...).........c...  00 01 00 29 00 00 00 00 00 00 00 00 E3 01 08 00
$1110  ......APPLESOFT4  00 00 00 02 00 1B 41 50 50 4C 45 53 4F 46 54 34
$1120  6....ö4...).....  36 00 00 00 00 FC 34 00 01 00 29 00 00 00 00 00
$1130  ...c.........APP  00 00 00 E3 01 08 00 00 00 00 02 00 1B 41 50 50
$1140  LESOFT47....ö5..  4C 45 53 4F 46 54 34 37 00 00 00 00 FC 35 00 01
$1150  .).........c.....  00 29 00 00 00 00 00 00 00 00 E3 01 08 00 00 00
$1160  ....APPLESOFT48.  00 02 00 1B 41 50 50 4C 45 53 4F 46 54 34 38 00
$1170  ...ö6...).........  00 00 00 FC 36 00 01 00 29 00 00 00 00 00 00 00
$1180  .c.........APPLE  00 E3 01 08 00 00 00 00 02 00 1B 41 50 50 4C 45
$1190  SOFT49....ö7...)  53 4F 46 54 34 39 00 00 00 00 FC 37 00 01 00 29
$11A0  ........c.........  00 00 00 00 00 00 00 E3 01 08 00 00 00 00 00 02
$11B0  ..APPLESOFT50...  00 1B 41 50 50 4C 45 53 4F 46 54 35 30 00 00 00
$11C0  .ö8...).........c  00 FC 38 00 01 00 29 00 00 00 00 00 00 00 00 E3
$11D0  .........APPLESO  01 08 00 00 00 00 02 00 1B 41 50 50 4C 45 53 4F
$11E0  FT51....ö9...)...  46 54 35 31 00 00 00 00 FC 39 00 01 00 29 00 00
$11F0  .......c.........  00 00 00 00 00 00 E3 01 08 00 00 00 00 02 00 00
```

```
BLOCK: $0002

$1000  ....vVOLUME.....  00 00 03 00 F6 56 4F 4C 55 4D 45 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ..C'........MAIN  00 00 C3 27 0D 02 00 06 00 18 01 1A 4D 41 49 4E
$1030  .BASIC.....ö..:.  2E 42 41 53 49 43 00 00 00 00 00 FC 07 00 01 00
$1040  .........c......  1B 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050  ..öSUBDIRECTORY.  02 00 DC 53 55 42 44 49 52 45 43 54 4F 52 59 00
$1060  ................  00 00 0F 08 00 01 00 00 02 00 00 00 00 00 00 00
$1070  c...............  E3 00 00 00 00 00 00 02 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


BLOCK: $0007

$1000  ..........:"MAIN  07 08 0A 00 97 00 19 08 14 00 BA 22 4D 41 49 4E
$1010  .BASIC".........  2E 42 41 53 49 43 22 00 00 00 90 00 00 00 00 00
$1020  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$107Q  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

BLOCK: $0008

```
$1000 ....!SUBDIRECTOR 00 00 00 00 EC 53 55 42 44 49 52 45 43 54 4F 52
$1010 Y...uHUSTON!.... 59 00 00 00 75 48 55 53 54 4F 4E 21 00 00 00 00
$1020 ..C'......'.SUB. 00 00 C3 27 0D 01 00 02 00 03 27 19 53 55 42 2E
$1030 BASIC......ë.... 42 41 53 49 43 00 00 00 00 00 00 FC 09 00 01 00
$1040 .........c...... 1A 00 00 00 00 00 00 00 00 E3 01 08 00 00 00 00
$1050 ................ 08 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

BLOCK: $0009

```
$1000 ..........:"SUB. 07 08 0A 00 97 00 18 08 14 00 BA 22 53 55 42 2E
$1010 BASIC"........... 42 41 53 49 43 22 00 00 00 90 00 00 00 00 00 00
$1020 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0 ................ 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

1.4.4.1. Struktur des Subdirectory-Kopfes

Der Subdirectory-Kopf befindet sich bei unserer Testdiskette im Block $0008. Zunächst erfolgt eine Kurzzusammenfassung der relativen Bytes des Kopfes und dann werden die Speicherstellen im einzelnen beschrieben, soweit sie uns nicht schon aufgrund der vorangehenden Abschnitte bekannt sind. Die Struktur ist fast dieselbe wie bei dem Volume-Directory-Kopf. Die in Klammern hinzugefügten Beispiel-Werte beziehen sich auf Block $0008.

Zusammenfassung Subdirectory-Kopf

$00-$01:	Rückwärts-Pointer (00 00)
$02-$03:	Vorwärts-Pointer (00 00)
$04:	Speichertyp und Namenslänge ($EC; $E = Subdirectory-Kopf)
$05-$13:	Dateiname (Subdirectory-Name ,,SUBDIRECTORY'')
$14-$1B:	unbenutzt (,,HUSTON'')
$1C-$1F:	Datum und Uhrzeit (00 00 00 00)
$20-$21:	ProDOS-Versionen (00 00)
$22:	Zugriffsbefugnis ($C3)
$23:	Länge jedes Dateieintrages ($27 = dezimal 39)
$24:	Anzahl der Einträge pro Block ($0D = dezimal 13)
$25-$26:	Anzahl aller aktiven Einträge (01 00 = $0001)
$27-$28:	Vorgänger-Zeiger (02 00 = Block $0002)
$29:	Vorgänger-Dateieintrag-Nummer ($03, d.h. an 3. Stelle)
$2A:	Vorgänger-Dateieintrag-Länge ($27 = dezimal 39)

Anmerkungen zu den neuen Bytes:

Byte $04: Speichertyp und Namenslänge

Der Speichertyp des Subdirectory-Kopfes ist $E (gegenüber $F beim Volume-Directory-Kopf).

Byte $14-$1B: unbenutzt (,,HUSTON'') (J.R. Huston?)

Diese 8 Bytes sind beim Volume-Directory-Kopf leer. Bei Subdirectory-Kopf hat sich wahrscheinlich der Name des Programmierers von ProDOS HUSTON ,,eingeschlichen''. Er steht in der Language Card ab $EFA7, wo sich ein provisorischer Subdirectory-Kopf befindet.

Byte $27-$28: Vorgänger-Zeiger

Der Vorgänger-Zeiger – im ,,Technical Manual'' als Parent Pointer bezeichnet – beinhaltet die Nummer desjenigen Diskettenblocks, der den Dateieintrag für den gegenwärtigen Subdirectory-Kopf enthält. Dies ist in diesem Fall der Block $0002, also der Key Block des Volume-Directory.

Byte $29: Vorgänger-Dateieintrag-Nummer

Die Vorgänger-Dateieintrag-Nummer (Parent Entry Number) ist der Subdirectory-Dateieintrag im Volume-Directory als X-ter Eintrag. Da ,,SUBDIRECTORY'' nach ,,VOLUME'' und ,,MAIN.BASIC'' der dritte Eintrag im Volume-Directory ist, enthält Byte $29 den Wert 3. Normalerweise würde man hier den Wert 2 erwarten, da ja der Volume-Directory-Kopf kein eigentlicher Dateieintrag ist. Dies würde jedoch zu einer unnötigen Verkomplizierung führen, weil sich bei einer gefüllten Catalog-Spur der Eintrag ,,SUBDIRECTORY'' natürlich auch in einem Non-Key-Block befinden könnte.

Byte $2A: Vorgänger-Dateieintrag-Länge

Diese beträgt wie beim Volume-Directory $27 oder dezimal 39, doch könnte es theoretisch sein, daß bei zukünftigen ProDOS-Versionen verschiedene Längen für Volume-Directory- und Subdirectory-Einträge verwendet werden.

Wenn ein Subsubdirectory angelegt wird, dann hat es denselben Kopf-Aufbau wie das Vorgänger-Subdirectory, auf das durch den Vorgänger-Zeiger im Subsubdirectory verwiesen wird.

1.4.5. Index-Block

Der Index-Block (= Blockliste, Blockverzeichnis) unter ProDOS entspricht der Track-Sector-Liste unter DOS 3.3 mit dem Unterschied, daß der Index-Block Datenblocks und nicht Datensektoren verzeichnet. Eine ,,normale'' Datei, z.B. eine Schößling-Datei wird auf dem externen Datenspeicher in 3 verschiedenen Arten von Blocks verzeichnet: erstens im entsprechenden Directory-Block – sei es im Volume-Directory oder in einem Subdirectory oder gar Subsubdirectory –, das den Zeiger zum Index-Block einhält (= Hauptzeiger = Key Pointer), zweitens im Index-Block, der die Block-Nummern der Datenblocks umfaßt, und drittens in den Datenblocks, die den eigentlichen Datei-Inhalt ausmachen.

Es gibt zwei Arten von Index-Blocks, den einfachen Index-Block und den Master-Index-Block. Letzterer verzeichnet die Index-Blocks und ersterer verzeichnet die Datenblocks. Beide haben denselben Aufbau (der übrigens im ,,Technical Manual" nicht beschrieben wird): Das niederwertige Byte der Block-Nummer befindet sich in der linken oder ersten Hälfte des Index-Blocks und das höherwertige Byte der Block-Nummer befindet sich in der rechten oder zweiten Hälfte des Index-Blocks. Nehmen wir an, im relativen Byte $000 des Index-Blocks stehe $17 und im relativen Byte $100 des Index-Blocks stehe $01, dann ist die Block-Nummer 17 01 = $0117 = dezimal 279 gemeint. Für die Block-Nummern 0-255 oder $0000-$00FF ist das Low Byte $00, d.h. die zweite Hälfte des Index-Blocks ist dann leer bzw. enthält nur Nullen. Man beachte, daß beim Index-Block das übliche Schema LL – HH (Low Byte – High Byte) insofern durchbrochen wird, als LL und HH nicht unmittelbar aufeinander folgen, sondern durch 256 Bytes getrennt sind.

1.4.5.1. Sämling-Datei

Hinweis: Zu dem Kapitel 1.4.5.1 gehören die ,,Dumps zu Kap. 1.4.5.1": Blocks $0002, $0006 und $0007, die auf den Seiten 135-136 abgebildet sind.

Wir nehmen nun wiederum unsere Testdiskette und initialisieren sie neu mit dem FILER unter den Namen ,,VOLUME". Dann starten wir folgendes Sämling-Testprogramm:

```
100 REM Sämling – Seedling
110 PRINT CHR$ (4) ,,OPEN TEXTFILE": PRINT CHR$ (4) ,,WRITE
    TEXTFILE"
120 REM Nachfolgendes belegt 1 Datenblock
130 FOR X = 1000 TO 1099: PRINT X: NEXT
140 PRINT CHR$ (4) ,,CLOSE"
```

Wenn wir diese Diskette dann mit dem Block-Reader-Programm untersuchen, stellen wir fest, daß gar kein Index-Block existiert. Es handelt sich nämlich hier um eine Sämling-Datei (Seedling-File), die sich als Datei mit maximal 512 Daten-Bytes = 1 Daten-Block charakterisieren läßt. In Block $0002 finden wir vor dem Namen ,,TEXTFILE" als Speichertyp $1 (als linkes Nibble von $18) = Seedling. Ferner finden wir den Hauptzeiger auf Block 07 00 = $0007 eingestellt. Fazit: Eine Sämling-Datei umfaßt genau 1 Datenblock – gleichviel, ob dieser ganz oder teilweise mit Informationen gefüllt ist – und der Hauptzeiger in Verbindung mit dem Speichertyp $1 verweist direkt auf diesen Datenblock. Der Index-Block entfällt.

```
BLOCK: $0002

$1000  ....vVOLUME.....  00 00 03 00 F6 56 4F 4C 55 4D 45 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ..C'.....TEXT     00 00 C3 27 0D 01 00 06 00 18 01 18 54 45 58 54
$1030  FILE............  46 49 4C 45 00 00 00 00 00 00 00 04 07 00 01 00
$1040  t.........c.....  F4 01 00 00 00 00 00 00 00 E3 00 00 00 00 00 00
$1050  ................  02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


BLOCK: $0006

$1000  ................  00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1010  ................  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1020  ................  FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BLOCK: $0007

$1000  1000.1001.1002.1  31 30 30 30 0D 31 30 30 31 0D 31 30 30 32 0D 31
$1010  003.1004.1005.10  30 30 33 0D 31 30 30 34 0D 31 30 30 35 0D 31 30
$1020  06.1007.1008.100  30 36 0D 31 30 30 37 0D 31 30 30 38 0D 31 30 30
$1030  9.1010.1011.1012  39 0D 31 30 31 30 0D 31 30 31 31 0D 31 30 31 32
$1040  .1013.1014.1015.  0D 31 30 31 33 0D 31 30 31 34 0D 31 30 31 35 0D
$1050  1016.1017.1018.1  31 30 31 36 0D 31 30 31 37 0D 31 30 31 38 0D 31
$1060  019.1020.1021.10  30 31 39 0D 31 30 32 30 0D 31 30 32 31 0D 31 30
$1070  22.1023.1024.102  32 32 0D 31 30 32 33 0D 31 30 32 34 0D 31 30 32
$1080  5.1026.1027.1028  35 0D 31 30 32 36 0D 31 30 32 37 0D 31 30 32 38
$1090  .1029.1030.1031.  0D 31 30 32 39 0D 31 30 33 30 0D 31 30 33 31 0D
$10A0  1032.1033.1034.1  31 30 33 32 0D 31 30 33 33 0D 31 30 33 34 0D 31
$10B0  035.1036.1037.10  30 33 35 0D 31 30 33 36 0D 31 30 33 37 0D 31 30
$10C0  38.1039.1040.104  33 38 0D 31 30 33 39 0D 31 30 34 30 0D 31 30 34
$10D0  1.1042.1043.1044  31 0D 31 30 34 32 0D 31 30 34 33 0D 31 30 34 34
$10E0  .1045.1046.1047.  0D 31 30 34 35 0D 31 30 34 36 0D 31 30 34 37 0D
$10F0  1048.1049.1050.1  31 30 34 38 0D 31 30 34 39 0D 31 30 35 30 0D 31
$1100  051.1052.1053.10  30 35 31 0D 31 30 35 32 0D 31 30 35 33 0D 31 30
$1110  54.1055.1056.105  35 34 0D 31 30 35 35 0D 31 30 35 36 0D 31 30 35
$1120  7.1058.1059.1060  37 0D 31 30 35 38 0D 31 30 35 39 0D 31 30 36 30
$1130  .1061.1062.1063.  0D 31 30 36 31 0D 31 30 36 32 0D 31 30 36 33 0D
$1140  1064.1065.1066.1  31 30 36 34 0D 31 30 36 35 0D 31 30 36 36 0D 31
$1150  067.1068.1069.10  30 36 37 0D 31 30 36 38 0D 31 30 36 39 0D 31 30
$1160  70.1071.1072.107  37 30 0D 31 30 37 31 0D 31 30 37 32 0D 31 30 37
$1170  3.1074.1075.1076  33 0D 31 30 37 34 0D 31 30 37 35 0D 31 30 37 36
$1180  .1077.1078.1079.  0D 31 30 37 37 0D 31 30 37 38 0D 31 30 37 39 0D
$1190  1080.1081.1082.1  31 30 38 30 0D 31 30 38 31 0D 31 30 38 32 0D 31
$11A0  083.1084.1085.10  30 38 33 0D 31 30 38 34 0D 31 30 38 35 0D 31 30
$11B0  86.1087.1088.108  38 36 0D 31 30 38 37 0D 31 30 38 38 0D 31 30 38
$11C0  9.1090.1091.1092  39 0D 31 30 39 30 0D 31 30 39 31 0D 31 30 39 32
$11D0  .1093.1094.1095.  0D 31 30 39 33 0D 31 30 39 34 0D 31 30 39 35 0D
$11E0  1096.1097.1098.1  31 30 39 36 0D 31 30 39 37 0D 31 30 39 38 0D 31
$11F0  099.............  30 39 39 0D 00 00 00 00 00 00 00 00 00 00 00 00
```

Hätten Sie's gewußt?

SYS-Files werden stets ab $2000 in den Speicher eingelesen. SYS-Files, die in BIN-Files umgewandelt wurden, booten nicht mehr; umgekehrt lassen sich SYS-Files ohne Parameter nicht BLOADen.

1.4.5.2. Schößling-Datei

Hinweis: Zu dem Kapitel 1.4.5.2 gehören die ,,Dumps zu Kap. 1.4.5.2'': Blocks $0002 und $0006-$0009, die auf den Seiten 139-141 abgebildet sind.

Als nächstes überschreiben wir unsere bereits vorhandene Sämling-Datei ,,TEXTFILE'' mit folgendem Programm:

```
100 REM Schößling – Sapling
110 PRINT CHR$ (4) ,,OPEN TEXTFILE'': PRINT CHR$ (4) ,,WRITE
    TEXTFILE''
120 REM Nachfolgendes belegt 2 Datenblocks und 1 Index-Block
130 FOR X = 1000 TO 1199: PRINT X: NEXT
140 PRINT CHR$ (4) ,,CLOSE''
```

Wenn wir unsere Testdiskette nunmehr untersuchen, stellen wir fest, daß im Block $0002 der Speichertyp von $1 zu $2 (linkes Nibble von Byte $28) geändert wurde und der Hauptzeiger nunmehr statt auf Block $0007 (dies ist nach wie vor der eigentliche erste Datenblock) jetzt auf Block $0008 zeigt, d.h. auf den Index-Block. Aus der Sämling-Datei ist eine Schößling-Datei (Sapling-File) geworden. Eine Schößling-Datei hat den Speichertyp $2 und der Hauptzeiger verweist auf den ersten und einzigen Index-Block, der bis zu 256 Datenblocks verwalten kann. In unserem Index-Block $0008 findet wir 07 00 und 09 00 – je durch 256 Bytes getrennt! –, d.h. die Datenblocks sind die Block-Nummern $0007 und $0009. Eine Schößling-Datei umfaßt minimal 2 und maximal 256 Datenblocks zuzüglich 1 Index-Block, nimmt mithin minimal 3 und maximal 257 physische Diskettenblocks ein. Es gibt also unter ProDOS keine Datei, die exakt 2 physische Blocks einnehmen würde, denn eine Sämling-Datei nimmt exakt 1 physischen Block und eine Schößling-Datei exakt mindestens 3 physische Blocks ein.

Auf einer ,,sauberen'' Diskette, d.h. einer Diskette, bei der die Dateien nicht durch wiederholtes Löschen und Neuspeichern blockmäßig stark verstreut sind, werden die Dateien gemäß der Volume Bit Map streng nach aufsteigenden, freien Block-Nummern von 0-279 gespeichert. Für eine Schößling-Datei ist dann die typische Anordnung gewährleistet: erster Datenblock – Index-Block – restliche Datenblocks. Für unsere ersten zwei Beispiele gilt:

a) Sämling-Datei:

Block $0006: Volume Bit Map
Block $0007: Erster und einziger Datenblock

b) Schößling-Datei

Block $0006: Volume Bit Map
Block $0007: Erster Datenblock
Block $0008: Erster und einziger Index-Block
Block $0009: Zweiter (und bei unserem Beispiel zugleich letzter) Datenblock

Wenn eine Datei mit DELETE gelöscht wird, dann wird erstens das Byte in der Catalog-Spur, das den Speichertyp und die Namenslänge beinhaltet, auf Null gesetzt und zweitens der Index-Block mit Ausnahme der ersten Block-Nummer (in unserem Beispiel 07 00) mit Nullen aufgefüllt. Ein UNDELETE-Programm könnte dann auf alle Fälle noch den ersten Datenblock rekonstruieren. Die restlichen Datenblocks würden sich hingegen nur dann aufgrund der Dateilängenangabe im Catalog-Dateieintrag rekonstruieren lassen, wenn es sich um eine „saubere" Diskette handelt, bei der sich die restlichen Datenblocks unmittelbar an den Index-Block anschließen. Die praktische Konsequenz daraus ist, daß gelöschte Dateien in der Regel nicht mehr gerettet werden können.

BLOCK: $0002

```
$1000  ....vVOLUME.....  00 00 03 00 F6 56 4F 4C 55 4D 45 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ..C'.....(TEXT    00 00 C3 27 0D 01 00 06 00 18 01 28 54 45 58 54
$1030  FILE............  46 49 4C 45 00 00 00 00 00 00 00 04 08 00 03 00
$1040  h........c......  E8 03 00 00 00 00 00 00 00 E3 00 00 00 00 00 00
$1050  ................  02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

BLOCK: $0006

```
$1000  .?..............  00 3F FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1010  ................  FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
$1020  ................  FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BLOCK: $0007

$1000  1000.1001.1002.1  31 30 30 30 0D 31 30 30 31 0D 31 30 30 32 0D 31
$1010  003.1004.1005.10  30 30 33 0D 31 30 30 34 0D 31 30 30 35 0D 31 30
$1020  06.1007.1008.100  30 36 0D 31 30 30 37 0D 31 30 30 38 0D 31 30 30
$1030  9.1010.1011.1012  39 0D 31 30 31 30 0D 31 30 31 31 0D 31 30 31 32
$1040  .1013.1014.1015.  0D 31 30 31 33 0D 31 30 31 34 0D 31 30 31 35 0D
$1050  1016.1017.1018.1  31 30 31 36 0D 31 30 31 37 0D 31 30 31 38 0D 31
$1060  019.1020.1021.10  30 31 39 0D 31 30 32 30 0D 31 30 32 31 0D 31 30
$1070  22.1023.1024.102  32 32 0D 31 30 32 33 0D 31 30 32 34 0D 31 30 32
$1080  5.1026.1027.1028  35 0D 31 30 32 36 0D 31 30 32 37 0D 31 30 32 38
$1090  .1029.1030.1031.  0D 31 30 32 39 0D 31 30 33 30 0D 31 30 33 31 0D
$10A0  1032.1033.1034.1  31 30 33 32 0D 31 30 33 33 0D 31 30 33 34 0D 31
$10B0  035.1036.1037.10  30 33 35 0D 31 30 33 36 0D 31 30 33 37 0D 31 30
$10C0  38.1039.1040.104  33 38 0D 31 30 33 39 0D 31 30 34 30 0D 31 30 34
$10D0  1.1042.1043.1044  31 0D 31 30 34 32 0D 31 30 34 33 0D 31 30 34 34
$10E0  .1045.1046.1047.  0D 31 30 34 35 0D 31 30 34 36 0D 31 30 34 37 0D
$10F0  1048.1049.1050.1  31 30 34 38 0D 31 30 34 39 0D 31 30 35 30 0D 31
$1100  051.1052.1053.10  30 35 31 0D 31 30 35 32 0D 31 30 35 33 0D 31 30
$1110  54.1055.1056.105  35 34 0D 31 30 35 35 0D 31 30 35 36 0D 31 30 35
$1120  7.1058.1059.1060  37 0D 31 30 35 38 0D 31 30 35 39 0D 31 30 36 30
$1130  .1061.1062.1063.  0D 31 30 36 31 0D 31 30 36 32 0D 31 30 36 33 0D
$1140  1064.1065.1066.1  31 30 36 34 0D 31 30 36 35 0D 31 30 36 36 0D 31
$1150  067.1068.1069.10  30 36 37 0D 31 30 36 38 0D 31 30 36 39 0D 31 30
$1160  70.1071.1072.107  37 30 0D 31 30 37 31 0D 31 30 37 32 0D 31 30 37
$1170  3.1074.1075.1076  33 0D 31 30 37 34 0D 31 30 37 35 0D 31 30 37 36
$11B0  .1077.1078.1079.  0D 31 30 37 37 0D 31 30 37 38 0D 31 30 37 39 0D
$1190  1080.1081.1082.1  31 30 38 30 0D 31 30 38 31 0D 31 30 38 32 0D 31
$11A0  083.1084.1085.10  30 38 33 0D 31 30 38 34 0D 31 30 38 35 0D 31 30
$11B0  86.1087.1088.108  38 36 0D 31 30 38 37 0D 31 30 38 38 0D 31 30 38
$11C0  9.1090.1091.1092  39 0D 31 30 39 30 0D 31 30 39 31 0D 31 30 39 32
$11D0  .1093.1094.1095.  0D 31 30 39 33 0D 31 30 39 34 0D 31 30 39 35 0D
$11E0  1096.1097.1098.1  31 30 39 36 0D 31 30 39 37 0D 31 30 39 38 0D 31
$11F0  099.1100.1101.11  30 39 39 0D 31 31 30 30 0D 31 31 30 31 0D 31 31


BLOCK: $0008

$1000  ................  07 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BLOCK: $0009

$1000  02.1103.1104.110  30 32 0D 31 31 30 33 0D 31 31 30 34 0D 31 31 30
$1010  5.1106.1107.1108  35 0D 31 31 30 36 0D 31 31 30 37 0D 31 31 30 38
$1020  .1109.1110.1111.  0D 31 31 30 39 0D 31 31 31 30 0D 31 31 31 31 0D
$1030  1112.1113.1114.1  31 31 31 32 0D 31 31 31 33 0D 31 31 31 34 0D 31
$1040  115.1116.1117.11  31 31 35 0D 31 31 31 36 0D 31 31 31 37 0D 31 31
$1050  18.1119.1120.112  31 38 0D 31 31 31 39 0D 31 31 32 30 0D 31 31 32
$1060  1.1122.1123.1124  31 0D 31 31 32 32 0D 31 31 32 33 0D 31 31 32 34
$1070  .1125.1126.1127.  0D 31 31 32 35 0D 31 31 32 36 0D 31 31 32 37 0D
$1080  1128.1129.1130.1  31 31 32 38 0D 31 31 32 39 0D 31 31 33 30 0D 31
$1090  131.1132.1133.11  31 33 31 0D 31 31 33 32 0D 31 31 33 33 0D 31 31
$10A0  34.1135.1136.113  33 34 0D 31 31 33 35 0D 31 31 33 36 0D 31 31 33
$10B0  7.1138.1139.1140  37 0D 31 31 33 38 0D 31 31 33 39 0D 31 31 34 30
$10C0  .1141.1142.1143.  0D 31 31 34 31 0D 31 31 34 32 0D 31 31 34 33 0D
$10D0  1144.1145.1146.1  31 31 34 34 0D 31 31 34 35 0D 31 31 34 36 0D 31
$10E0  147.1148.1149.11  31 34 37 0D 31 31 34 38 0D 31 31 34 39 0D 31 31
$10F0  50.1151.1152.115  35 30 0D 31 31 35 31 0D 31 31 35 32 0D 31 31 35
$1100  3.1154.1155.1156  33 0D 31 31 35 34 0D 31 31 35 35 0D 31 31 35 36
$1110  .1157.1158.1159.  0D 31 31 35 37 0D 31 31 35 38 0D 31 31 35 39 0D
$1120  1160.1161.1162.1  31 31 36 30 0D 31 31 36 31 0D 31 31 36 32 0D 31
$1130  163.1164.1165.11  31 36 33 0D 31 31 36 34 0D 31 31 36 35 0D 31 31
$1140  66.1167.1168.116  36 36 0D 31 31 36 37 0D 31 31 36 38 0D 31 31 36
$1150  9.1170.1171.1172  39 0D 31 31 37 30 0D 31 31 37 31 0D 31 31 37 32
$1160  .1173.1174.1175.  0D 31 31 37 33 0D 31 31 37 34 0D 31 31 37 35 0D
$1170  1176.1177.1178.1  31 31 37 36 0D 31 31 37 37 0D 31 31 37 38 0D 31
$1180  179.1180.1181.11  31 37 39 0D 31 31 38 30 0D 31 31 38 31 0D 31 31
$1190  82.1183.1184.118  38 32 0D 31 31 38 33 0D 31 31 38 34 0D 31 31 38
$11A0  5.1186.1187.1188  35 0D 31 31 38 36 0D 31 31 38 37 0D 31 31 38 38
$11B0  .1189.1190.1191.  0D 31 31 38 39 0D 31 31 39 30 0D 31 31 39 31 0D
$11C0  1192.1193.1194.1  31 31 39 32 0D 31 31 39 33 0D 31 31 39 34 0D 31
$11D0  195.1196.1197.11  31 39 35 0D 31 31 39 36 0D 31 31 39 37 0D 31 31
$11E0  98.1199.........  39 38 0D 31 31 39 39 0D 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Hätten Sie's gewußt?

Wenn man einen SYS-Files BLOADen will, dann gebe man stets TSYS und A$2000 an, z.B.
BLOAD PRODOS, TSYS, A$2000
BLOAD BASIC.SYSTEM, TSYS, A$2000
BLOAD FILER, TSYS, A$2000

1.4.5.3. Baum-Datei

Hinweis: Zu dem Kapitel 1.4.5.3 gehören die ,,Dumps zu Kap. 1.4.5.3'': Blocks $0002, $0006-$0009 sowie $0107-$010A, die auf den Seiten 144-148 abgebildet sind.

Als nächstes überschreiben wir unsere bereits vorhandene Schößling-Datei ,,TEXTFILE'' mit folgendem Programm:

```
100 REM Baum - Tree
110 PRINT CHR$ (4) ,,OPEN TEXTFILE'': PRINT CHR$ (4) ,,WRITE
    TEXTFILE''
120 REM Nachfolgendes belegt 257 Datenblocks, 1 Master-Index-Block und 2
    Index-Blocks
130 FOR X = 1000000 TO 1016384: PRINT X: NEXT
140 PRINT CHR$ (4) ,,CLOSE''
```

Wenn wir nun den Directory-Block $0002 untersuchen, dann stellen wir fest, daß sich der Speichertyp von $2 (Schößling-Datei) in $3 (Baum-Datei) gewandelt hat ($3 ist linkes Nibble von $38). Ferner zeigt der Hauptzeiger nunmehr auf 08 01 = Block $0108, wo sich der Master-Index-Block (Hauptindexblock, Key Index Block) befindet. In diesem Block $0108 sind zwei Index-Blocks verzeichnet, nämlich $0008 und $0109. In ersten Index-Block - dem physischen Block $0008 - finden wir als ersten Datenblock die Block-Nummer $0007. Im zweiten (und in unserem Beispiel letzten) Index-Block - dem physischen Block $0109 - finden wir als letzten Datenblock die Block-Nummer $010A. Aus der Schößling-Datei ist damit eine Baum-Datei geworden, deren Speichertyp $3 ist und deren Hauptzeiger auf den Master-Index-Block zeigt, der seinerseits auf die Index-Blocks verweist. Ein Master-Index-Block könnte theoretisch auf 256 Index-Blocks zeigen, doch werden unter ProDOS Version 1.0 nur jeweils die ersten 128 Bytes der linken und rechten Hälfte des Master-Index-Blocks genutzt. Eine Baum-Datei kann damit ,,nur'' maximal 128 · 256 = 32768 Datenblocks (= 16 Megabytes) umfassen. Für unser Beispiel, die die kleinstmögliche Baum-Datei darstellt, gilt folgende Anordnung der Blocks:

Block $0006: Volume Bit Map
Block $0007: 1. Datenblock
Block $0008: Erster Index-Block
Block $0009: 2. Datenblock
usw.
Block $0107: 256. Datenblock
Block $0108: Master-Index-Block (Key Index Block)
Block $0109: Zweiter Index-Block
Block $010A: 257. Datenblock (hier letzter Datenblock)

Der CATALOG-Befehl würde für diese Baum-Datei 260 belegte Blocks ausweisen: 257 Datenblocks + 1 Master-Index-Block + 2 Index-Blocks = 260 physische Blocks.

Wenn die nunmehr ,,ausgewachsene" Baum-Datei ,,TEXTFILE" durch eine kürzere Datei desselben Namens *per MLI* überschrieben wird, dann schrumpft sie zu einer Schößling- oder gar Sämling-Datei, d.h. die nicht mehr benötigten, alten Datenblocks werden wieder freigegeben. Hierin unterscheidet sich ProDOS von DOS 3.3, welch letzteres in diesem Fall die TSL nicht reduziert (Das BASIC-SYSTEM läßt zwar Binärfiles, nicht jedoch Textfiles schrumpfen.)

```
BLOCK: $0002

$1000  ....vVOLUME.....  00 00 03 00 F6 56 4F 4C 55 4D 45 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ..C'.....8TEXT    00 00 C3 27 0D 01 00 06 00 18 01 38 54 45 58 54
$1030  FILE............  46 49 4C 45 00 00 00 00 00 00 00 04 08 01 04 01
$1040  .........c......  08 00 02 00 00 00 00 00 00 E3 00 00 00 00 00 00
$1050  ................  02 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

BLOCK: $0006

$1000  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ................  00 1F FF 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

```
BLOCK: $0007

$1000   1000000.1000001.   31 30 30 30 30 30 30 0D 31 30 30 30 30 30 31 0D
$1010   1000002.1000003.   31 30 30 30 30 30 32 0D 31 30 30 30 30 30 33 0D
$1020   1000004.1000005.   31 30 30 30 30 30 34 0D 31 30 30 30 30 30 35 0D
$1030   1000006.1000007.   31 30 30 30 30 30 36 0D 31 30 30 30 30 30 37 0D
$1040   1000008.1000009.   31 30 30 30 30 30 38 0D 31 30 30 30 30 30 39 0D
$1050   1000010.1000011.   31 30 30 30 30 31 30 0D 31 30 30 30 30 31 31 0D
$1060   1000012.1000013.   31 30 30 30 30 31 32 0D 31 30 30 30 30 31 33 0D
$1070   1000014.1000015.   31 30 30 30 30 31 34 0D 31 30 30 30 30 31 35 0D
$1080   1000016.1000017.   31 30 30 30 30 31 36 0D 31 30 30 30 30 31 37 0D
$1090   1000018.1000019.   31 30 30 30 30 31 38 0D 31 30 30 30 30 31 39 0D
$10A0   1000020.1000021.   31 30 30 30 30 32 30 0D 31 30 30 30 30 32 31 0D
$10B0   1000022.1000023.   31 30 30 30 30 32 32 0D 31 30 30 30 30 32 33 0D
$10C0   1000024.1000025.   31 30 30 30 30 32 34 0D 31 30 30 30 30 32 35 0D
$10D0   1000026.1000027.   31 30 30 30 30 32 36 0D 31 30 30 30 30 32 37 0D
$10E0   1000028.1000029.   31 30 30 30 30 32 38 0D 31 30 30 30 30 32 39 0D
$10F0   1000030.1000031.   31 30 30 30 30 33 30 0D 31 30 30 30 30 33 31 0D
$1100   1000032.1000033.   31 30 30 30 30 33 32 0D 31 30 30 30 30 33 33 0D
$1110   1000034.1000035.   31 30 30 30 30 33 34 0D 31 30 30 30 30 33 35 0D
$1120   1000036.1000037.   31 30 30 30 30 33 36 0D 31 30 30 30 30 33 37 0D
$1130   1000038.1000039.   31 30 30 30 30 33 38 0D 31 30 30 30 30 33 39 0D
$1140   1000040.1000041.   31 30 30 30 30 34 30 0D 31 30 30 30 30 34 31 0D
$1150   1000042.1000043.   31 30 30 30 30 34 32 0D 31 30 30 30 30 34 33 0D
$1160   1000044.1000045.   31 30 30 30 30 34 34 0D 31 30 30 30 30 34 35 0D
$1170   1000046.1000047.   31 30 30 30 30 34 36 0D 31 30 30 30 30 34 37 0D
$1180   1000048.1000049.   31 30 30 30 30 34 38 0D 31 30 30 30 30 34 39 0D
$1190   1000050.1000051.   31 30 30 30 30 35 30 0D 31 30 30 30 30 35 31 0D
$11A0   1000052.1000053.   31 30 30 30 30 35 32 0D 31 30 30 30 30 35 33 0D
$11B0   1000054.1000055.   31 30 30 30 30 35 34 0D 31 30 30 30 30 35 35 0D
$11C0   1000056.1000057.   31 30 30 30 30 35 36 0D 31 30 30 30 30 35 37 0D
$11D0   1000058.1000059.   31 30 30 30 30 35 38 0D 31 30 30 30 30 35 39 0D
$11E0   1000060.1000061.   31 30 30 30 30 36 30 0D 31 30 30 30 30 36 31 0D
$11F0   1000062.1000063.   31 30 30 30 30 36 32 0D 31 30 30 30 30 36 33 0D


BLOCK: $0008

$1000   ................   07 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17
$1010   ........ !"#$%&'   18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27
$1020   ()*+,-./01234567   28 29 2A 2B 2C 2D 2E 2F 30 31 32 33 34 35 36 37
$1030   89:;<=>?@ABCDEFG   38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47
$1040   HIJKLMNOPQRSTUVW   48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57
$1050   XYZÄÖÜ↑_`abcdefg   58 59 5A 5B 5C 5D 5E 5F 60 61 62 63 64 65 66 67
$1060   hijklmnopqrstuvw   68 69 6A 6B 6C 6D 6E 6F 70 71 72 73 74 75 76 77
$1070   xyzäöüß.........   78 79 7A 7B 7C 7D 7E 7F 80 81 82 83 84 85 86 87
$1080   ................   88 89 8A 8B 8C 8D 8E 8F 90 91 92 93 94 95 96 97
$1090   ........ !"#$%&'   98 99 9A 9B 9C 9D 9E 9F A0 A1 A2 A3 A4 A5 A6 A7
$10A0   ()*+,-./01234567   A8 A9 AA AB AC AD AE AF B0 B1 B2 B3 B4 B5 B6 B7
$10B0   89:;<=>?@ABCDEFG   B8 B9 BA BB BC BD BE BF C0 C1 C2 C3 C4 C5 C6 C7
$10C0   HIJKLMNOPQRSTUVW   C8 C9 CA CB CC CD CE CF D0 D1 D2 D3 D4 D5 D6 D7
$10D0   XYZÄÖÜ↑_`abcdefg   D8 D9 DA DB DC DD DE DF E0 E1 E2 E3 E4 E5 E6 E7
$10E0   hijklmnopqrstuvw   E8 E9 EA EB EC ED EE EF F0 F1 F2 F3 F4 F5 F6 F7
$10F0   xyzäöüß.........   F8 F9 FA FB FC FD FE FF 00 01 02 03 04 05 06 07
$1100   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0   ................   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0   ................   00 00 00 00 00 00 00 00 01 01 01 01 01 01 01 01
```

BLOCK: $0009

$1000	1000064.1000065.	31	30	30	30	30	36	34	0D	31	30	30	30	30	36	35	0D
$1010	1000066.1000067.	31	30	30	30	30	36	36	0D	31	30	30	30	30	36	37	0D
$1020	1000068.1000069.	31	30	30	30	30	36	38	0D	31	30	30	30	30	36	39	0D
$1030	1000070.1000071.	31	30	30	30	30	37	30	0D	31	30	30	30	30	37	31	0D
$1040	1000072.1000073.	31	30	30	30	30	37	32	0D	31	30	30	30	30	37	33	0D
$1050	1000074.1000075.	31	30	30	30	30	37	34	0D	31	30	30	30	30	37	35	0D
$1060	1000076.1000077.	31	30	30	30	30	37	36	0D	31	30	30	30	30	37	37	0D
$1070	1000078.1000079.	31	30	30	30	30	37	38	0D	31	30	30	30	30	37	39	0D
$1080	1000080.1000081.	31	30	30	30	30	38	30	0D	31	30	30	30	30	38	31	0D
$1090	1000082.1000083.	31	30	30	30	30	38	32	0D	31	30	30	30	30	38	33	0D
$10A0	1000084.1000085.	31	30	30	30	30	38	34	0D	31	30	30	30	30	38	35	0D
$10B0	1000086.1000087.	31	30	30	30	30	38	36	0D	31	30	30	30	30	38	37	0D
$10C0	1000088.1000089.	31	30	30	30	30	38	38	0D	31	30	30	30	30	38	39	0D
$10D0	1000090.1000091.	31	30	30	30	30	39	30	0D	31	30	30	30	30	39	31	0D
$10E0	1000092.1000093.	31	30	30	30	30	39	32	0D	31	30	30	30	30	39	33	0D
$10F0	1000094.1000095.	31	30	30	30	30	39	34	0D	31	30	30	30	30	39	35	0D
$1100	1000096.1000097.	31	30	30	30	30	39	36	0D	31	30	30	30	30	39	37	0D
$1110	1000098.1000099.	31	30	30	30	30	39	38	0D	31	30	30	30	30	39	39	0D
$1120	1000100.1000101.	31	30	30	30	31	30	30	0D	31	30	30	30	31	30	31	0D
$1130	1000102.1000103.	31	30	30	30	31	30	32	0D	31	30	30	30	31	30	33	0D
$1140	1000104.1000105.	31	30	30	30	31	30	34	0D	31	30	30	30	31	30	35	0D
$1150	1000106.1000107.	31	30	30	30	31	30	36	0D	31	30	30	30	31	30	37	0D
$1160	1000108.1000109.	31	30	30	30	31	30	38	0D	31	30	30	30	31	30	39	0D
$1170	1000110.1000111.	31	30	30	30	31	31	30	0D	31	30	30	30	31	31	31	0D
$1180	1000112.1000113.	31	30	30	30	31	31	32	0D	31	30	30	30	31	31	33	0D
$1190	1000114.1000115.	31	30	30	30	31	31	34	0D	31	30	30	30	31	31	35	0D
$11A0	1000116.1000117.	31	30	30	30	31	31	36	0D	31	30	30	30	31	31	37	0D
$11B0	1000118.1000119.	31	30	30	30	31	31	38	0D	31	30	30	30	31	31	39	0D
$11C0	1000120.1000121.	31	30	30	30	31	32	30	0D	31	30	30	30	31	32	31	0D
$11D0	1000122.1000123.	31	30	30	30	31	32	32	0D	31	30	30	30	31	32	33	0D
$11E0	1000124.1000125.	31	30	30	30	31	32	34	0D	31	30	30	30	31	32	35	0D
$11F0	1000126.1000127.	31	30	30	30	31	32	36	0D	31	30	30	30	31	32	37	0D

BLOCK: $0107

$1000	1016320.1016321.	31	30	31	36	33	32	30	0D	31	30	31	36	33	32	31	0D
$1010	1016322.1016323.	31	30	31	36	33	32	32	0D	31	30	31	36	33	32	33	0D
$1020	1016324.1016325.	31	30	31	36	33	32	34	0D	31	30	31	36	33	32	35	0D
$1030	1016326.1016327.	31	30	31	36	33	32	36	0D	31	30	31	36	33	32	37	0D
$1040	1016328.1016329.	31	30	31	36	33	32	38	0D	31	30	31	36	33	32	39	0D
$1050	1016330.1016331.	31	30	31	36	33	33	30	0D	31	30	31	36	33	33	31	0D
$1060	1016332.1016333.	31	30	31	36	33	33	32	0D	31	30	31	36	33	33	33	0D
$1070	1016334.1016335.	31	30	31	36	33	33	34	0D	31	30	31	36	33	33	35	0D
$1080	1016336.1016337.	31	30	31	36	33	33	36	0D	31	30	31	36	33	33	37	0D
$1090	1016338.1016339.	31	30	31	36	33	33	38	0D	31	30	31	36	33	33	39	0D
$10A0	1016340.1016341.	31	30	31	36	33	34	30	0D	31	30	31	36	33	34	31	0D
$10B0	1016342.1016343.	31	30	31	36	33	34	32	0D	31	30	31	36	33	34	33	0D
$10C0	1016344.1016345.	31	30	31	36	33	34	34	0D	31	30	31	36	33	34	35	0D
$10D0	1016346.1016347.	31	30	31	36	33	34	36	0D	31	30	31	36	33	34	37	0D
$10E0	1016348.1016349.	31	30	31	36	33	34	38	0D	31	30	31	36	33	34	39	0D
$10F0	1016350.1016351.	31	30	31	36	33	35	30	0D	31	30	31	36	33	35	31	0D
$1100	1016352.1016353.	31	30	31	36	33	35	32	0D	31	30	31	36	33	35	33	0D
$1110	1016354.1016355.	31	30	31	36	33	35	34	0D	31	30	31	36	33	35	35	0D
$1120	1016356.1016357.	31	30	31	36	33	35	36	0D	31	30	31	36	33	35	37	0D
$1130	1016358.1016359.	31	30	31	36	33	35	38	0D	31	30	31	36	33	35	39	0D
$1140	1016360.1016361.	31	30	31	36	33	36	30	0D	31	30	31	36	33	36	31	0D
$1150	1016362.1016363.	31	30	31	36	33	36	32	0D	31	30	31	36	33	36	33	0D
$1160	1016364.1016365.	31	30	31	36	33	36	34	0D	31	30	31	36	33	36	35	0D
$1170	1016366.1016367.	31	30	31	36	33	36	36	0D	31	30	31	36	33	36	37	0D
$1180	1016368.1016369.	31	30	31	36	33	36	38	0D	31	30	31	36	33	36	39	0D
$1190	1016370.1016371.	31	30	31	36	33	37	30	0D	31	30	31	36	33	37	31	0D
$11A0	1016372.1016373.	31	30	31	36	33	37	32	0D	31	30	31	36	33	37	33	0D
$11B0	1016374.1016375.	31	30	31	36	33	37	34	0D	31	30	31	36	33	37	35	0D
$11C0	1016376.1016377.	31	30	31	36	33	37	36	0D	31	30	31	36	33	37	37	0D
$11D0	1016378.1016379.	31	30	31	36	33	37	38	0D	31	30	31	36	33	37	39	0D
$11E0	1016380.1016381.	31	30	31	36	33	38	30	0D	31	30	31	36	33	38	31	0D
$11F0	1016382.1016383.	31	30	31	36	33	38	32	0D	31	30	31	36	33	38	33	0D

BLOCK: $0108

$1000	.....	08 09 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100	.....	00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

BLOCK: $0109

$1000	.....	0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1010	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100	.....	01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0	.....	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```
BLOCK: $010A

$1000  1016384.........  31 30 31 36 33 38 34 0D 00 00 00 00 00 00 00 00
$1010  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1020  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1030  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1040  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1050  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1060  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1070  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1080  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1090  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$10F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1100  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1110  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1120  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1130  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1140  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1150  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1160  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1170  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1180  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$1190  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11A0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11B0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11C0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11D0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11E0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
$11F0  ................  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Hätten Sie's gewußt?

Mit dem Befehl
BLOAD/VOLUME, TDIR, A$1000
lassen sich alle 4 Blocks des Volume-Directory namens "VOLUME" ab $1000 in den Speicher einlesen. Auch noch nicht belegte Volume-Directory-Blocks werden auf diese Weise eingelesen.
Mit dem Befehl
BLOAD/VOLUME/SUBDIRECTORY, A$2000
läßt sich ein komplettes Subdirectory einlesen, selbst wenn es eine Vielzahl von Blocks umfaßt.

1.5. MLI (Machine Language Interface)

Das MLI ist die Schnittstelle für Assembler-Programmierer zum PRODOS-Betriebssystem in der Language Card. Insgesamt gibt es bei ProDOS Version 1.0 z.Zt. 26 verschiedene Befehle, von denen 2 offenbar nachträglich eingeflickt und deshalb im „Technical Manual" nicht ($65) oder nicht mehr vollständig ($82) behandelt wurden.

Beispiel für einen MLI-Aufruf

```
MLI-AUFRUF      JSR  MLI         ;$BF00
BEFEHLNUMMER    HEX  80          ;hier Read Block
BEFEHLSLISTE    DA   PARAMETER   ;LL-HH
FEHLER1         BNE  FEHLER2     ;Akku ungleich 0
ENDE1           RTS              ;Okay-Exit
FEHLER2         JSR  $FDDA       ;Hexout
ENDE2           RTS              ;Nicht-Okay-Exit
PARAMETER       HEX  03          ;Parameter-Anzahl
GERÄTENUMMER    HEX  60          ;Slot 6, Drive 1
PUFFER          DA   $7000       ;$7000-$71FF
BLOCKNUMMER     HEX  0100        ;Block $0001
```

1. Der Aufruf eines MLI-Befehls (MLI-Call) erfolgt stets durch JSR MLI = JSR $BF00. Ein JSR statt eines JMP ist deshalb erforderlich, weil PRODOS über den Stack erstens die Rücksprungadresse und zweitens die Befehlsnummer sowie die Adresse der Parameter-Liste ermittelt.

2. Unmittelbar an den JSR MLI muß sich die hexadezimale Befehlsnummer (Command Number) anschließen, z.B. $80 für Read Block.

3. Danach folgt die Adresse der Parameter-Liste in der üblichen LL-HH-Form.

4. Die Speicherstelle, die der Adresse der Parameter-Liste folgt, ist die MLI-Rücksprungadresse, die stets das sechste Byte nach dem JSR-MLI-Aufruf ist. Die zwischen JSR MLI und der Rücksprungspeicherstelle liegenden 3 Byte für Befehlsnummer und Adresse der Parameter-Liste werden also von PRODOS übersprungen, indem die Stackadresse entsprechend erhöht wird.

5. Nach der Rückkehr von der MLI-Routine sind das X- und Y-Register unverändert, deren Werte übrigens in der PRODOS Global Page zwischengespeichert werden. Der Akkumulator enthält $00 und Carry-Flag ist zurückgesetzt (BCC), falls kein Fehler auftrat, so daß der MLI-Aufruf mit BEQ oder BCC verlassen werden kann. Andernfalls enthält der Akkumulator die Fehlernummer und das Carry-Flag ist gesetzt, so daß mit BNE oder BCS in die eigene Error-Handling-Routine gesprungen werden kann.

6. Die Parameter-Liste kann sich an einer beliebigen Stelle in den unteren 48K befinden, doch wird man sie in der Regel in der Nähe des MLI-Aufrufs ansiedeln. Für jeden Befehl gibt es eine eigene Parameter-Liste. Das obige Beispiel bezieht sich auf die Readblock-Routine.

Die Parameter lassen sich in zwei Arten gliedern: Zur ersten Art gehören Vorgabe-Parameter, die an das MLI übergeben werden sollen und deshalb entsprechend initialisiert werden müssen. Zur zweiten Art gehören Ergebnis-Parameter, deren Werte von PRODOS in die Parameter-Liste gepokt werden und die deshalb nicht initialisiert werden müssen.

Eine Parameter-Liste umfaßt maximal 17 Bytes. Da sich nicht alle Informationen in einer solch kurzen Liste unterbringen lassen, enthält sie teilweise auch Pointer zu entsprechenden Stellen, z.B. zu Puffern, Dateinamen usw., die sich außerhalb der Parameter-Liste befinden. Das erste Byte der Parameter-Liste ist stets die Parameter-Anzahl (Parameter-Count), die jedoch nicht mit der Anzahl der Parameter-Bytes identisch ist, da zusammengehörige Bytes, z.B. Pointer, Datum-Bytes usw., als jeweils 1 Parameter aufgefaßt werden. Eine Parameter-Liste kann bis zu 10 Parameter enthalten.

Die MLI-Befehle lassen sich in drei Gruppen einteilen:

a) System-Befehle: Hierzu gehören Befehle auf der untersten Systemebene, z.B. READ BLOCK, WRITE BLOCK usw.

b) Allgemeine Datei-Befehle: Hierzu gehören Befehle, die den Namen, die Struktur und die Existenz von Dateien betreffen, z.B. CREATE, DELETE, RENAME usw.

c) Spezielle Datei-Befehle: Hierzu gehören alle Befehle, die mit der Änderung, Erweiterung usw. von Dateien zu tun haben, z.B. OPEN, READ, WRITE, CLOSE usw.

Ausgerüstet mit den theoretischen Kenntnissen über die interne und externe Speicherorganisation, die das Thema der beiden vorangehenden Hauptkapitel bildete, ist die Anwendung der MLI-Befehle unproblematisch. Trotzdem sollen die Befehle in den nachfolgenden Teilkapiteln zunächst an einfachen Beispielen erläutert werden. Im zweiten Band von ,,ProDOS für Aufsteiger'' werden kompliziertere, größere MLI-Anwender-Programme enthalten sein, die aus Platzgründen im ersten Band nicht mehr untergebracht werden konnten.

Der entscheidende Vorteil von ProDOS gegenüber DOS 3.3 ist in meinen Augen der Umstand, daß nicht nur auf der untersten Block-Ebene (z.B. RWTS bei DOS), sondern auch darüber hinaus auf der File-Manager-Ebene ein klar definiertes Benutzer-Interface für Assembler-Programmierer geschaffen wurde, so daß man in Anwenderprogrammen u.U. auf das BASIC.SYSTEM völlig verzichten kann.

1.5.1. Parameter-Liste

Nachfolgend werden zunächst die englischen Bezeichnungen der MLI-Befehle mit deutscher Übersetzung aufgelistet. In Anschluß daran wird gezeigt, welche Parameter bei welchen MLI-Befehlen vorkommen und welche Bedeutung sie haben. Die englischen Bezeichnungen weichen in Einzelfällen vom ,,Technical Manual'' ab. Beispielsweise sprechen wir von File-Reference-Number statt nur Reference-Number, um die File-Reference-Number von der Interrupt-Reference-Number abzugrenzen.

Zusammenfassung der MLI-Befehle

$40 ALLOCATE INTERRUPT (Interrupt-Vektor-Adresse zuweisen)
$41 DEALLOCATE INTERRUPT (Interrupt-Vektor-Adresse entfernen)
$65 REBOOT (von mir so bezeichnet, da von Apple nicht dokumentiert)
$80 READ BLOCK (Block von Diskette lesen)
$81 WRITE BLOCK (Block auf Diskette schreiben)
$82 GET TIME (Uhr-Befehl)
$C0 CREATE (neue Datei oder neues Subdirectory anlegen)
$C1 DESTROY (Datei oder Subdirectory löschen; DELETE)
$C2 RENAME (Datei, Subdirectory oder Volume umbenennen)
$C3 SET FILE INFO (Dateieintrag = Catalog-Eintrag ändern)
$C4 GET FILE INFO (Dateieintrag abrufen)
$C5 ON LINE (Volume-Namen der angeschlossenen Drives ermitteln)

$C6 SET PREFIX (Präfix neu festlegen)
$C7 GET PREFIX (aktives Präfix ermitteln)
$C8 OPEN (Datei beliebigen Typs öffnen; nicht nur bei Textfiles!)
$C9 NEWLINE (Record- oder Datei-Endmarker als bestimmtes Byte definieren)
$CA READ (Datei beliebigen Typs von Diskette einlesen)
$CB WRITE (Datei beliebigen Typs auf Diskette schreiben)
$CC CLOSE (Datei schließen)
$CD FLUSH (I/O-Puffer auf Datei übertragen)
$CE SET MARK (Positionszeiger definieren)
$CF GET MARK (momentanen Positionszeiger ermitteln)
$D0 SET EOF (End of File = Dateilänge definieren bzw. kürzen)
$D1 GET EOF (End of File ermitteln)
$D2 SET BUF (I/O-Pufferadresse festlegen)
$D3 GET BUF (momentane I/O-Pufferadresse ermitteln)

Zusammenfassung der Parameter

1. PARAMETER-COUNT – 1 Byte
2. NAME-POINTER – 2 Bytes LL HH
3. NAME – 1 Längebyte + 1-64 Zeichen
4. DATA-BUFFER-POINTER – 2 Bytes LL HH
5. DATA-BUFFER – nicht notwendigerweise auf Page Boundary beginnend
6. I/O-BUFFER-POINTER – 2 Bytes LL HH
7. I/O-BUFFER – auf Page Boundary beginnend und $0400 Bytes lang
8 CREATION-DATE – 2 Bytes
9. CREATION-TIME – 2 Bytes
10. MODIFICATION-DATE – 2 Bytes
11. MODIFICATION-TIME – 2 Bytes
12. ACCESS – 1 Byte
13. FILE-TYPE – 1 Byte
14. STORAGE-TYPE – 1 Byte (rechtes Nibble!)
15. AUXILIARY-TYPE – 2 Bytes
16. BLOCKS USED – 2 Bytes LL HH
17. NULL-FIELD – 3 Bytes (leer bei SET FILE INFO)
18. EOF – 3 Bytes LL MM HH
19. POSITION – 3 Bytes LL MM HH
20. NEWLINE-CHARACTER – 1 Byte
21. ENABLE-MASK – 1 Byte

22. FILE-REFERENCE-NUMBER – 1 Byte
23. REQUEST-COUNT – 2 Bytes LL HH
24. TRANSFER-COUNT – 2 Bytes LL HH
25. UNIT-NUMBER – 1 Byte
26. BLOCK-NUMBER – 2 Bytes LL HH
27. INTERRUPT-CODE-POINTER – 2 Bytes LL HH
28. INTERRUPT-REFERENCE-NUMBER – 1 Byte

1. PARAMETER-COUNT (Parameter-Anzahl) – 1 Byte

Anzahl der Parameter im Bereich $00-$0A. Kommt bei allen Befehlen mit Ausnahme von GET TIME vor.

2. NAME-POINTER (Zeiger zu NAME) – 2 Bytes LL HH

Kommt bei CREATE, DESTROY, RENAME, SET FILE INFO, GET FILE INFO und OPEN vor. Hier bezieht sich der NAME-POINTER auf den Pathname, Volume-Name oder File-Name. Kommt ferner bei SET PREFIX vor. Hier bezieht sich der NAME-POINTER auf den Präfix-Namen.

3. NAME (PATHNAME oder PREFIX NAME) – 1-65 Bytes

Byte $00: Namenslänge
Byte $01-$40: Name selbst (in hexadezimal mit Bit 7 off)

Beispiel: 09 2F 56 31 2F 44 45 4D 4F 2F = ,,/V1/DEMO/", 9 Bytes lang

Kommt bei denselben Befehlen wie bei NAME-POINTER vor. Für RENAME gibt es 2 NAME-POINTER und 2 NAMEN = OLD and NEW PATHNAME (alter Name und neuer Name). Soweit möglich, gebe man sicherheitshalber immer den vollständigen Namen mit Schrägstrichen an.

4. DATA-BUFFER-POINTER (Zeiger zum Datenpuffer) – 2 Bytes LL HH

Kommt bei ON LINE, GET PREFIX, READ, WRITE, SET BUF, GET BUF, READ BLOCK und WRITE BLOCK vor.

5. DATA-BUFFER (Datenpuffer) – mindestens 1 Byte

Der Datenpuffer muß nicht auf einer Seitengrenze (Page Boundary) beginnen. Er darf sich nur in einem aufgrund der System Bit Map zulässigen Speicherbereich der unteren 48K befinden. Die Größe des Datenpuffers hängt vom Befehl ab. Beispielsweise könnte sich der WRITE-Befehl auf den Bereich $0800-$BEFF erstrecken, falls das BASIC.SYSTEM ausgeschaltet ist. Daten im Sinne des Datenpuffers sind unterschiedlicher Natur: Disketten-Blocks (bei READ BLOCK), Teile einer Datei (bei READ), Teile des Catalogs (bei ON LINE) usw.

Kommt bei denselben Befehlen wie der DATA-BUFFER-POINTER vor.

6. I/O-BUFFER-POINTER (Zeiger zum Input-Output-Puffer) – 2 Bytes LL HH

Kommt explizit bei OPEN, SET BUF und GET BUF sowie implizit bei allen anderen sich an OPEN anschließenden Befehlen (z.B. NEWLINE, READ, WRITE usw.) vor.

7. I/O-BUFFER (Input-Output-Puffer) – $0400 (1024) Bytes

Der I/O-Puffer muß stets auf einer Seitengrenze beginnen und $0400 Bytes lang sein und darf mit der System Bit Map nicht kollidieren. Beim BASIC.SYSTEM liegt z.B. der erste I/O-Puffer im Bereich $9600-$99FF. Man beachte, daß ein I/O-Puffer nur bei den von OPEN abhängigen Befehlen benötigt wird. So geht z.B. die BLOCK-READ-Routine nicht über einen I/O-Puffer.

Kommt bei denselben Befehlen wie der I/O-BUFFER-POINTER vor.

Man beachte, daß NAME, DATA-BUFFER und I/O-BUFFER nicht Bestandteile der Parameter-Liste sind, da diese lediglich die entsprechenden Pointer enthält.

8. CREATION-DATE (Datum der Anlage der Datei) – 2 Bytes

9. CREATION-TIME (Uhrzeit der Anlage der Datei) – 2 Bytes

CREATION-DATE und CREATION-TIME kommen bei CREATE und GET FILE INFO vor. Zur Kodierung siehe PRODOS Global Page Assemblerlisting.

10. MODIFICATION-DATE (Datum der Änderung der Datei) – 2 Bytes

11. MODIFICATION-TIME (Uhrzeit der Änderung der Datei) – 2 Bytes

MODIFICATION-DATE und MODIFICATION-TIME kommen bei SET FILE INFO und GET FILE INFO vor. Da der GET-TIME-Befehl keine Parameter hat, kommen Datum und Uhrzeit hier nicht als Parameter vor, wenngleich dieser Befehl die Uhr-Daten (aufgrund einer Hardware-Uhr) in die Global Page pokt.

12. ACCESS (Zugriffsbefugnis) – 1 Byte

Kommt bei CREATE, SET FILE INFO und GET FILE INFO vor. Zum Begriff ACCESS siehe Kapitel 1.4.2.1.

13. FILE-TYPE (Dateityp) – 1 Byte

Kommt bei CREATE, SET FILE INFO und GET FILE INFO vor. Zum Begriff FILE-TYPE siehe Kapitel 1.4.3.1.

14. STORAGE-TYPE (Speichertyp) – 1 Byte

Kommt bei CREATE und GET FILE INFO vor. Zum Begriff STORAGE-TYPE siehe Kapitel 1.4.2.1. Man beachte, daß auf der Diskette in ein einziges Byte Speichertyp (linkes Nibble) und Namenslänge (rechtes Nibble) gepackt werden. Bei CREATE und GET FILE INFO befindet sich jedoch der Speichertyp im *rechten* Nibble des einen Bytes und die Namenslänge wird nicht hinzugepackt.

15. AUXILIARY-TYPE (Zusatzinfo) – 2 Bytes

Kommt bei CREATE, SET FILE INFO UND GET FILE INFO vor. Zum Begriff AUX-TYPE siehe Kapitel 1.4.3.1.

16. BLOCKS USED (Anzahl der belegten Blocks) – 2 Bytes LL HH

Kommt bei GET FILE INFO vor. Die Anzahl der belegten Blocks einer *Datei* ist nicht mit (1) der möglichen Blockgesamtanzahl einer Diskette (z.B. 280 bei Disk II), die im Volume-Directory-Kopf steht, und nicht mit (2) der Gesamtzahl aller belegten Blocks einer Diskette, die GET FILE INFO berechnen kann, zu verwechseln.

17. NULL-FIELD (Leerfeld) – 3 Bytes

Dieser Leerparameter kommt nur bei SET FILE INFO vor, um die Symmetrie zu GET FILE INFO zu wahren. Das Leerfeld steht dort in der Parameter-Liste, wo bei GET FILE INFO der Speichertyp und die Anzahl der belegten Blocks gepokt werden.

18. EOF (End of File, Dateilänge) – 3 Bytes LL MM HH

Kommt bei SET EOF und GET EOF vor. Zum Begriff EOF siehe Kapitel 1.4.3.1.

19. POSITION (MARK, Positionszeiger) – 3 Bytes LL MM HH

Kommt bei SET MARK und GET MARK vor.

20. NEWLINE-CHARACTER („Neue Zeile", Endmarker) – 1 Byte

Kommt bei NEWLINE vor. Unterscheide: Unter EOF versteht man die Gesamtdateilänge, z.B. 2000 Bytes, oder die absolute Position des letzten Bytes der Datei. Unter dem Positionszeiger versteht man die Stelle des momentan von der Diskette einzulesenden oder auf die Diskette zu schreibenden Bytes, z.B. das 10. Byte. Der Positionszeiger wird stets absolut vom Dateibeginn (= $000000) aus gerechnet. Unter Endmarker im Sinne eines besonderen Zeichens, z.B. Return = $0D, verstehen wir ein Byte, das ein Feld- oder Record-Ende oder u.U. auch ein Datei-Ende markiert. Beim READ-Befehl besteht beispielsweise die Möglichkeit, bei einer bestimmten Datei ab etwa der absoluten Position des 10. Bytes *entweder* eine definierte Anzahl, z.B. 100 Bytes, *oder* genauso viele Bytes einzulesen, bis ein definierter Endmarker, z.B. Return, erreicht ist.

21. ENABLE-MASK (Undierungs-Byte, UND-Maske) – 1 Byte

Kommt bei NEWLINE vor. Die UND-Maske ist ein Byte, mit dem der z.B. eingelesene Buchstabe mit

```
LDA BUCHSTABE
AND ENABLE-MASK
CMP NEWLINE-CHAR
BEQ ENDE
BNE WEITERLESEN
```

undiert wird. Falls der Inhalt des Akkumulators danach dem NEWLINE-CHAR entspricht, wird der Einlesevorgang beendet. Die UND-Maske ist üblicherweise $7F (dann wird Bit 7 ignoriert, d.h. $0D und $8D sind gleichwertig) oder $FF (dann ist jedes Bit von Bedeutung). Der NEWLINE-Modus wird durch UND-Maske = $00 abgeschaltet.
Man beachte, daß EOF auf der Diskette gespeichert ist, nicht hingegen POSITION, NEWLINE-CHAR und ENABLE-MASK. Letztere spielen also nur MLI-intern eine Rolle.

22. FILE-REFERENCE-NUMBER (File-Verweisnummer) – 1 Byte

Kommt bei OPEN, NEWLINE, READ, WRITE, CLOSE, FLUSH, SET MARK, GET MARK, SET EOF, GET EOF, SET BUF und GET BUF vor. Der OPEN-Befehl muß mit NAME erfolgen, doch danach wird in die OPEN-Parameter-Liste eine File-Verweisnummer gepokt, die dann für alle soeben erwähnten MLI-Befehle verwendet werden muß. Mit anderen Worten setzen diese anderen MLI-Befehle wie NEWLINE, READ usw. voraus, daß die Datei zuvor geöffnet worden ist.

23. REQUEST-COUNT (Anzahl der zu übertragenden Bytes; Sollwert)
– 2 Bytes LL HH

Kommt bei READ und WRITE vor.

24. TRANSFER-COUNT (Anzahl der übertragenen Bytes; Istwert) – 2 Bytes LL HH

Kommt bei READ und WRITE vor. Der REQUEST-COUNT als Sollwert ist die Anzahl der einzulesenden oder zu speichernden Bytes, wohingegen der TRANSFER-COUNT als Istwert die Summe der tatsächlich übertragenen Bytes beinhaltet.

25. UNIT-NUMBER (Geräte-Nummer) – 1 Byte

Kommt bei READ BLOCK und WRITE BLOCK vor. Die UNIT-NUMBER enthält bitmäßig verschlüsselt Slot- und Drive-Nummer (siehe Kapitel 1.3.2 sowie 1.5.2).

26. BLOCK-NUMBER (Block-Nummer) – 2 Bytes LL HH

Kommt bei READ BLOCK und WRITE BLOCK vor.

27. INTERRUPT-CODE-POINTER (Startadresse des Interrupt-Handlers)
– 2 Bytes LL HH

Kommt bei ALLOCATE INTERRUPT vor.

28. INTERRUPT-REFERENCE-NUMBER (Interrupt-Verweisnummer) – 1 Byte

Kommt bei ALLOCATE INTERRUPT und DEALLOCATE INTERRUPT vor.

1.5.2. System-Befehle

Hinweis: Zu diesem Kapitel gehören die ,,Programmbeispiele zu Kap. 1.5.2'' ab Seite 165.

Als System-Befehle, die im ,,Technical Manual'' als ,,System Calls'' bezeichnet werden und die die Hardware am innigsten tangieren, gelten folgende:

$80 = READ BLOCK = Einlesen eines Blocks (= 2 Sektoren) vom externen Datenspeicher
$81 = WRITE BLOCK = Schreiben eines Blocks auf einen externen Datenspeicher
$40 = ALLOCATE INTERRUPT = Interrupt-Driver-Adresse in MLI-IRQ-Tabelle eintragen
$41 = DEALLOCATE INTERRUPT = Interrupt-Adresse wieder entfernen
$82 = GET TIME = Hardware-Uhrzeit in PRODOS Global Page eintragen
$65 = REBOOT = anderes Systemprogramm einladen

READ BLOCK ($80) – Parameter

Parameter-Count = $03 – 1 Byte
Unit-Number = z.B. $60 = Slot 6, Drive 1 – 1 Byte
Data-Buffer-Pointer = z.B. 00 10 = $1000 – LL HH – 2 Bytes
Block-Number = z.B. 20 00 = $0020 = dezimal 32 – LL HH – 2 Bytes

WRITE BLOCK ($81) – Parameter

Parameter-Count = $03 – 1 Byte
Unit-Number = z.B. $E0 = Slot 6, Drive 2 – 1 Byte
Data-Buffer-Pointer = z.B. 11 22 = $2211 – LL HH – 2 Bytes
Block-Number = z.B. 17 01 = $0117 = dezimal 279 – LL HH – 2 Bytes

Die Unit-Number ist im linken Nibble verschlüsselt.

7 6 5 4 3 2 1 0
D S S S 0 0 0 0

Drive D als Bit 7 in der Form: D = 0 = Drive 1; D = 1 = Drive 2

0SSS0000 = Slot-Nummer S in der Hex-Form: S0

Der Data-Buffer muß $0200 Bytes umfassen und braucht nicht auf einer Seitengrenze beginnen, darf aber andererseits nicht mit der System Bit Map kollidieren.
Die Block-Number darf die aufgrund der Disk-Driver-Prüfroutine zulässige Höchstzahl (von 279) nicht überschreiten.

Die READ BLOCK und WRITE BLOCK Routinen wurden bereits in den früher gelisteten Programmen „Speed Test 1" (s.S. 21), „Speed Test 2" (s.S. 23) und „Block-Reader" (s.S. 81) benutzt.

ALLOCATE INTERRUPT ($40) – Parameter

Parameter-Count = $02 – 1 Byte
Interrupt-Reference-Number = 1 Byte; wird vom MLI gepokt
Interrupt-Code-Pointer = z.B. 00 03 = $0300 – LL HH – 2 Bytes

DEALLOCATE INTERRUPT ($41) – Parameter

Parameter-Count = $01 – 1 Byte
Interrupt-Reference-Number = z.B. $02 – 1 Byte

Der Interrupt-Code-Pointer zeigt zur Startadresse des Interrupt-Drivers oder Interrupt-Handlers. Diese Adresse, die meist mit einer Slot-Adresse ($CS00) identisch ist, da interruptfähige Karten wie Hardware-Uhr usw. in Slots gesteckt werden müssen, wird in der PRODOS Global Page eingetragen.
Bei ALLOCATE INTERRUPT wird dem Interrupt eine Interrupt-Reference-Number im Bereich 1-4 zugewiesen, d.h. ProDOS kann bis zu 4 Interrupts nacheinander bearbeiten, wobei der ersteingetragene Interrupt jeweils zuerst abgefragt wird (Polling). Diese Interrupt-Reference-Number muß vom Anwenderprogramm zwischengespeichert werden, da bei DEALLOCATE INTERRUPT diese Verweisnummer angegeben werden muß.

GET TIME ($82) – ohne Parameter

Dieser Befehl ist von der Firma Apple nachträglich eingeflickt worden und hat keine Parameter (siehe Programmbeispiel). Zweck dieses Befehls ist der Abruf der Uhr-Routine zur Übertragung von Datum und Uhrzeit in die Global Page. Wer keine Hardware-Uhr besitzt, kann statt dessen das in Kapitel 1.3.9 gelistete Programm ,,Datum-Eingabe für PRODOS'' benutzen.

REBOOT ($65) – mit unvollständigen Parametern

Auch dieser Befehl ist nachträglich eingeflickt worden und hat als einzigen Parameter den Parameter-Count $04 (siehe Programmbeispiel). Der Reboot-Befehl ist im PRODOS Global Page Listing näher beschrieben.

1.5.3. Allgemeine Datei-Befehle

Hinweis: Zu diesem Kapitel gehören die ,,Programmbeispiele zu Kap. 1.5.3'' ab Seite 169.

Diese Gruppe von MLI-Befehlen, die im ,,Technical Manual'' als ,,Housekeeping Calls'' bezeichnet werden, betreffen die Neuanlage und das Löschen von Dateien, die Änderung der Dateieinträge durch RENAME und SET FILE INFO sowie die Ermittlung der angeschlossenen Drives einschließlich deren Volume-Namen.

$C0 = CREATE = Neue Datei oder neues Subdirectory anlegen
$C1 = DESTROY = Datei oder Subdirectory löschen (DELETE)
$C2 = RENAME = Name ändern
$C3 = SET FILE INFO = bestimmte Dateieintrag-Informationen ändern
$C4 = GET FILE INFO = bestimmte Dateieintrag-Informationen abrufen
$C5 = ON LINE = angeschlossene Volumes ermitteln
$C6 = SET PREFIX = Präfix neu bestimmen
$C7 = GET PREFIX = momentan aktives Präfix ermitteln

CREATE ($C0) – Parameter

Parameter-Count = $07 – 1 Byte
Name-Pointer = z.B. 80 02 = $0280 – LL HH – 2 Bytes
Access = z.B. $C3 (UNLOCK-File) – 1 Byte
File-Type = z.B. $FC (Applesoft-Programm) – 1 Byte

Auxiliary Type = z.B. 01 08 = $0801 (Applesoft-Startadresse) – 2 Bytes
Storage-Type = entweder $01 (Seedling) oder $0D (Subdirectory) – 1 Byte
Creation-Date = z.B. 00 00 (kein Datum) – 2 Bytes
Creation-Time = z.B. 00 00 (keine Uhrzeit) – 2 Bytes

Sofern mit CREATE kein neues Subdirectory, sondern ein normaler File angelegt wird, ist der Storage-Type stets $01 = Sämling-Datei. ,,CREATE Subdirectory'' bewirkt den Dateieintrag im Volume-Directory und legt den ersten Subdirectory-Block mit dem Subdirectory-Kopf an. ,,CREATE File'' bewirkt den Dateieintrag und legt den ersten Datenblock an, der jedoch noch leer bleibt.
CREATE setzt automatisch das Backup-Bit des Access-Bytes.

DESTROY ($C1) – Parameter

Parameter-Count = $01 – 1 Byte
Name-Pointer = z.B. 80 02 = $0280 – LL HH – 2 Bytes

DESTROY ist nur möglich, wenn der File nicht OPEN ist. Deshalb ist die von DOS 3.3 gewohnte Befehlsfolge OPEN – DELETE – OPEN – WRITE nicht mehr möglich und muß beim BASIC.SYSTEM durch OPEN – CLOSE – DELETE – OPEN – WRITE ersetzt werden, was übrigens 8 - 10 Sekunden dauert.
Ein Subdirectory kann nur mit DESTROY entfernt werden, wenn alle darin enthaltenen Dateieinträge bereits gelöscht sind.
DESTROY setzt das Namenslänge-Speichertyp-Byte im Catalog-Eintrag auf Null und löscht die Index-Blocks.

RENAME ($C2) – Parameter

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

Parameter-Count = $07 – 1 Byte
Name-Pointer – LL HH – 2 Bytes
Access – 1 Byte
File-Type – 1 Byte
Auxiliary Type – 2 Bytes

Null-Field – 00 00 00 (unbenutzt) – 3 Bytes
Modification-Date – z.B. 00 00 (kein Datum) – 2 Bytes
Modification-Time – z.B. 00 00 (keine Uhrzeit) – 2 Bytes

GET FILE INFO ($C4) – Parameter

Parameter-Count = $0A – 1 Byte
Name-Pointer – LL HH – 2 Bytes
(Nachfolgende Parameter werden allesamt vom MLI gepokt)
Access – 1 Byte; wird vom MLI gepokt
File-Type – 1 Byte; wird vom MLI gepokt
Auxiliary Type – 2 Bytes; werden vom MLI gepokt
Storage-Type – 1 Byte; wird vom MLI gepokt (Null-Field bei SET FILE INFO)
Blocks used – LL HH – 2 Bytes; werden vom MLI gepokt (Null-Field bei SET FILE INFO)
Modification-Date – 2 Bytes; werden vom MLI gepokt
Modification-Time – 2 Bytes; werden vom MLI gepokt
Creation-Date – 2 Bytes; werden vom MLI gepokt
Creation-Time – 2 Bytes; werden vom MLI gepokt

Üblicherweise wird man zunächst mit GET FILE INFO die Dateieintrag-Informationen einlesen, im Speicher ändern und dann wieder mit SET FILE INFO auf die Diskette zurückschreiben

Mit RENAME und SET FILE INFO, die beide zusammen zur Änderung der Dateieintrag-Informationen dienen, lassen sich die folgenden Werte *nicht* ändern (siehe hierzu Kapitel 1.4.3.1): Datum und Uhrzeit der Dateianlage, Hauptzeiger zum Dateianfang, Anzahl der belegten Blocks, ProDOS-Versionen, Pointer zum Volume- oder Subdirectory-Kopf. Der EOF (Dateilänge), der ebenfalls zum Dateieintrag gehört, kann mit SET EOF geändert werden. Die Befehle RENAME, SET FILE INFO und SET EOF hätte man zweckmäßigerweise zu einem einzigen Befehl zusammenfassen können. Lästig ist fernerhin, daß kein MLI-Befehl zum Einlesen der Index-Blocks einer Datei sowie der Volume Bit Map existiert, womit man Informationen über die Blockverteilung einer Datei sowie der Diskette selbst erhalten könnte. Darüber hinaus fehlt der von Apple Pascal bekannte KRUNCH-Befehl.

SET FILE INFO könnte zwar auch ausgeführt werden, während ein File OPEN ist, doch würde dies den sog. Dateieintragblock = File Control Block = FCB, der übrigens in der LC im Bereich $F6CF-$F7FE liegt, nicht berühren. Deshalb verwende

man SET FILE INFO nur bei nicht-geöffneten Files.
SET FILE INFO setzt automatisch das Backup-Bit des Access-Bytes.

Wenn sich GET FILE INFO auf das Volume selbst bezieht, dann wird vom MLI in dem Aux-Type-Feld die Gesamtanzahl der möglichen Blocks der Diskette und in dem Blocks-used-Feld die Gesamtanzahl der belegten Blocks gepokt. Man beachte, daß letztere Zahl nicht auf der Diskette physisch gespeichert ist, sondern errechnet werden muß.

ON LINE ($C5) – Parameter

Parameter-Count = $02 – 1 Byte
Unit-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes

ON LINE ermittelt Slot und Drive sowie Volume-Name und Namenslänge des oder der angeschlossenen externen Datenspeicher. Wenn man eine konkrete Unit-Number angibt, genügt ein 16 Bytes langer Datenpuffer. Wenn man hingegen als Unit-Number $00 angibt, werden alle Volumes ermittelt und der Datenpuffer muß 256 Bytes umfassen. Die Struktur der im Datenpuffer abgelegten Volume-Informationen ist folgende:

```
Erster Eintrag                                          Endmarker

00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F ... 00 01
ULV  O  L  U  M  E  N  A  M  E  00 00 00 00 00      00 00
U0 F  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00          (Bei Fehler)
```

Byte $00: Unit-Number U (Drive/Slot) sowie Länge L des Volume-Namens in folgender Form:

```
7 6 5 4 3 2 1 0
D S S S L L L L
1 1 1 0 1 1 0 0   Beispiel: Drive 2, Slot 6, Länge 12 Bytes
```

Byte $01: Falls ein MLI-Fehler auftrat, ist die Namenslänge = $0 (DSSS0000) und die Fehler-Nummer F wird im Byte $01 abgelegt.

Byte $01-$0F: Volume-Name (ohne Schrägstriche!)

Nach dem letzten Eintrag enthalten Byte $00 und Byte $01 beide Nullen als Endmarker.

SET PREFIX ($C6) – Parameter

Parameter-Count = $01 – 1 Byte
Name-Pointer – LL HH – 2 Bytes

GET PREFIX ($C7) – Parameter

Parameter-Count = $01 – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes

Bei SET PREFIX gebe man sicherheitshalber das vollständige Präfix in der Form „/VOLUME/SUBDIRECTORY/" an (maximale Länge 1 Längenbyte + 64 Namensbytes . GET PREFIX erwartet einen 128 Bytes langen Datenpuffer, in dem die Länge und der Name einschließlich Schrägstriche abgelegt wird.

Bei SET PREFIX kann man mit Namenslänge = $00 ein sog. Null-Präfix definieren. Sinngemäß enthält dann der Datenpuffer bei GET PREFIX als Länge eine Null.

```
                        1                ORG   $6000
                        2    *
                        3    * Read Block $80
                        4    * ==============
                        5    *
                        6    MLI         EQU   $BF00
                        7    BELL        EQU   $FF3A
                        8    HEXOUT      EQU   $FDDA
                        9    *
6000: 20 00 BF          10   RDBLOCK     JSR   MLI
6003: 80                11   COMMAND1    HEX   80          ;Read
6004: 09 60             12               DA    COUNT1
6006: D0 07             13               BNE   ERROR
6008: 60                14               RTS
                        15   *
6009: 03                16   COUNT1      HEX   03
                        17   *
600A: 60                18   UNITNUM1    DFB   #%01100000 ;D1,S6
                        19   *
                        20   * Bit 7 = 0 = Drive 2
                        21   * Bit 7 = 1 = Drive 1
                        22   *
                        23   * Bits 6-4: Slot: 1-7
                        24   *
                        25   * Beispiele Unit-Number:
                        26   *
                        27   *         76543210
                        28   * $60 = %01100000 = D1,S6
                        29   * $E0 = %11100000 = D2,S6
                        30   * $B0 = %10110000 = D2,S3
                        31   *
                        32   * 512-Bytes-Puffer
                        33   *
600B: 00 70             34   BUFFER1     DA    $7000       ;-$71FF
                        35   *
600D: 01                36   BLOCKL1     HEX   01
600E: 00                37   BLOCKH1     HEX   00          ;Block 1
                        38   *
600F: 20 DA FD          39   ERROR       JSR   HEXOUT
6012: 4C 3A FF          40               JMP   BELL
                        41   *
                        42   * Write Block $81
                        43   * ===============
                        44   *
6015: 20 00 BF          45   WRBLOCK     JSR   MLI
6018: 81                46   COMMAND2    HEX   81          ;Write
6019: 1E 60             47               DA    COUNT2
601B: D0 F2             48               BNE   ERROR
601D: 60                49               RTS
601E: 03                50   COUNT2      HEX   03
601F: B0                51   UNITNUM2    DFB   #%10110000 ;D2,S3
6020: 00 70             52   BUFFER2     DA    $7000       ;-$71FF
6022: 17                53   BLOCKL2     HEX   17
6023: 01                54   BLOCKH2     HEX   01          ;B1.279
```

```
                        1            ORG  $6000
                        2    *
                        3    * Alloc-Dealloc-Demo
                        4    * ==================
                        5    *
                        6    * Dieses Demo zeigt die
                        7    * IR-Vektor-Tabelle
                        8    * vor Alloc, nach Alloc
                        9    * und nach Dealloc,
                        10   * wobei die Werte für vor
                        11   * Alloc und nach Dealloc
                        12   * identisch sein müssen.
                        13   *
                        14   MLI       EQU  $BF00
                        15   HEXOUT    EQU  $FDDA
                        16   PRINT     EQU  $FDED
                        17   IRVEKTOR  EQU  $BF80       ;-$BF87
                        18   *
                        19   * Allocate Interrupt $40
                        20   * ======================
                        21   *
6000: 20 1E 60          22   ALLOC     JSR  IRDISP
6003: 20 00 BF          23             JSR  MLI
6006: 40                24   COMMAND1  HEX  40          ;ALLOC
6007: 0C 60             25             DA   COUNT1
6009: 4C 10 60          26             JMP  DEALLOC
                        27   *
600C: 02                28   COUNT1    HEX  02
                        29   *
                        30   * Nummer des Interrupts 1-4
                        31   *
600D: 01                32   INTNUM1   HEX  01
                        33   *
                        34   * Startadresse des Interrupt-
                        35   * programms hier fiktiv $0300
                        36   *
600E: 00 03             37   INTPROG1  DA   $0300
                        38   *
                        39   * Dealloate Interrupt $41
                        40   * =======================
                        41   *
6010: 20 1E 60          42   DEALLOC   JSR  IRDISP
6013: 20 00 BF          43             JSR  MLI
6016: 41                44   COMMAND2  HEX  41          ;DEALLOC
6017: 1C 60             45             DA   COUNT2
6019: 4C 1E 60          46             JMP  IRDISP
                        47   *
601C: 01                48   COUNT2    HEX  01
                        49   *
                        50   * Nummer des Interrupts 1-4
                        51   *
601D: 01                52   INTNUM2   HEX  01
                        53   *
```

```
                    55   * Alle 4 IR-Vektoren anzeigen
                    56   *
601E: AC 81 BF      57   IRDISP   LDY  IRVEKTOR+1
6021: AE 80 BF      58            LDX  IRVEKTOR+0
6024: 20 3F 60      59            JSR  IRDISP1
6027: AC 83 BF      60            LDY  IRVEKTOR+3
602A: AE 82 BF      61            LDX  IRVEKTOR+2
602D: 20 3F 60      62            JSR  IRDISP1
6030: AC 85 BF      63            LDY  IRVEKTOR+5
6033: AE 84 BF      64            LDX  IRVEKTOR+4
6036: 20 3F 60      65            JSR  IRDISP1
6039: AC 87 BF      66            LDY  IRVEKTOR+7
603C: AE 86 BF      67            LDX  IRVEKTOR+6
603F: A9 8D         68   IRDISP1  LDA  #$8D
6041: 20 ED FD      69            JSR  PRINT
6044: A9 A4         70            LDA  #"$"
6046: 20 ED FD      71            JSR  PRINT
6049: 98            72            TYA
604A: 20 DA FD      73            JSR  HEXOUT
604D: 8A            74            TXA
604E: 20 DA FD      75            JSR  HEXOUT
6051: 60            76            RTS

--End assembly--

82 bytes

Errors: 0
```

Hätten Sie's gewußt?

Wenn man kürzere Subdirectory- und Subsubdirectory-Namen durch RENAME in längere Namen umwandelt, kann die Summe aller Präfix-Namen größer als 64 Zeichen werden mit der Folge, daß man ohne Tricks nicht mehr auf eine ,,Sub-Sub-Sub-Datei'' zugreifen kann. Fazit: Erstens Subdirectories nicht zu tief verschachteln und zweitens kurze Subdirectory-Namen wählen.

```
                        1               ORG  $6000
                        2    *
                        3    * Reboot und Get Time
                        4    * ====================
                        5    *
                        6    * Diese Befehle haben eine
                        7    * unvollständige Parameter-
                        8    * Liste, da sie nachträglich
                        9    * von Apple eingeflickt wurden
                        10   * und deshalb vom MLI-Interpreter
                        11   * vor den anderen, normalen
                        12   * Befehlen abgefangen werden.
                        13   *
                        14   * Siehe Language Card, $D03D ff.
                        15   *
                        16   * Reboot $65
                        17   * ==========
                        18   *
                        19   MLI       EQU  $BF00
                        20   *
6000: 20 00 BF          21             JSR  MLI
6003: 65                22   COMMAND1  HEX  65          ;Reboot
6004: 06 60             23             DA   COUNT1
                        24   *
                        25   * Kehrt nicht mehr hier zurück!
                        26   * Daher nach COUNT1 nichts mehr!
                        27   *
6006: 04                28   COUNT1    HEX  04
                        29   *
                        30   * Get Time $82
                        31   * ============
                        32   *
6007: 20 00 BF          33             JSR  MLI
600A: 82                34   COMMAND2  HEX  82          ;Uhr
                        35   *
                        36   * Die Stelle für die sonst
                        37   * übliche Adresse von COUNT
                        38   * beliebig auffüllen.
                        39   * Der COUNT entfällt. In der
                        40   * internen MLI-Prüfroutine
                        41   * steht $00, doch wird dieser
                        42   * Wert mit COUNT nicht mehr
                        43   * verglichen!
                        44   *
600B: 00 00             45   FÜLLER    HEX  0000
                        46   *
                        47   * Der Exit ist dort, wo sonst
                        48   * der COUNT wäre.
                        49   *
600D: 60                50   EXIT      RTS
```

```
                        1               ORG   $6000
                        2    *
                        3    MLI        EQU   $BF00
                        4    BELL       EQU   $FF3A
                        5    HEXOUT     EQU   $FDDA
                        6    *
                        7    * Create $C0
                        8    * ==========
                        9    *
6000: 20 00 BF         10               JSR   MLI
6003: C0               11    COMMAND1   HEX   C0          ;Create
6004: 0F 60            12               DA    COUNT1
6006: D0 01            13               BNE   ERROR1
6008: 60               14               RTS
6009: 20 DA FD         15    ERROR1     JSR   HEXOUT
600C: 4C 3A FF         16               JMP   BELL
                       17    *
                       18    * Parameter-Liste
                       19    *
600F: 07               20    COUNT1     HEX   07
6010: 1B 60            21    NAME1      DA    LÄNGE
6012: C3               22    ACCESS     HEX   C3          ;UNLOCK
6013: 04               23    FILETYPE   HEX   04          ;Textfile
6014: 00 00            24    AUXTYPE    HEX   0000        ;keiner
6016: 01               25    STORAGE    HEX   01          ;Sämling
6017: 00 00            26    CDATE      HEX   0000        ;leer
6019: 00 00            27    CTIME      HEX   0000        ;leer
                       28    *
                       29    * Vollständiger Dateiname
                       30    *
601B: 11               31    LÄNGE      DFB   #STRICH2-STRICH1+1
601C: 2F               32    STRICH1    ASC   '/'
601D: 56 4F 4C         33               ASC   'VOLUME/TEXTFILE'
6020: 55 4D 45 2F 54 45 58 54
6028: 46 49 4C 45
602C: 2F               34    STRICH2    ASC   '/'
                       35    *
                       36    * Destroy $C1
                       37    * ===========
                       38    *
                       39    * Achtung: Bei OPEN-Files
                       40    *           kein Destroy möglich!
                       41    *
602D: 20 00 BF         42               JSR   MLI
6030: C1               43    COMMAND2   HEX   C1          ;Destroy
6031: 3C 60            44               DA    COUNT2
6033: D0 01            45               BNE   ERROR2
6035: 60               46               RTS
6036: 20 DA FD         47    ERROR2     JSR   HEXOUT
6039: 4C 3A FF         48               JMP   BELL
                       49    *
                       50    * Parameter-Liste
                       51    *
603C: 01               52    COUNT2     HEX   01
603D: 1B 60            53    NAME2      DA    LÄNGE
```

```
                      1                ORG   $6000
                      2    *
                      3    MLI         EQU   $BF00
                      4    BELL        EQU   $FF3A
                      5    HEXOUT      EQU   $FDDA
                      6    *
                      7    * Rename $C2
                      8    * ==========
                      9    *
6000: 20 00 BF        10               JSR   MLI
6003: C2              11   COMMAND     HEX   C2          ;Rename
6004: 0F 60           12               DA    COUNT
6006: D0 01           13               BNE   ERROR
6008: 60              14               RTS
6009: 20 DA FD        15   ERROR       JSR   HEXOUT
600C: 4C 3A FF        16               JMP   BELL
                      17   *
                      18   * Parameter-Liste
                      19   *
600F: 02              20   COUNT       HEX   02
6010: 14 60           21   NAMEALT     DA    LANGEALT
6012: 1D 60           22   NAMENEU     DA    LANGENEU
                      23   *
                      24   * Alter Name
                      25   *
6014: 08              26   LANGEALT    DFB   #STRICHA2-STRICHA1+1
6015: 2F              27   STRICHA1    ASC   '/'
6016: 56 4F 4C        28               ASC   'VOLUME'
6019: 55 4D 45
601C: 2F              29   STRICHA2    ASC   '/'
                      30   *
                      31   * Neuer Name
                      32   *
601D: 08              33   LANGENEU    DFB   #STRICHN2-STRICHN1+1
601E: 2F              34   STRICHN1    ASC   '/'
601F: 53 54 49        35               ASC   'STIEHL'
6022: 45 48 4C
6025: 2F              36   STRICHN2    ASC   '/'

--End assembly--

38 bytes

Errors: 0
```

```
:ASM                    1               ORG   $6000
                        2    *
                        3    MLI        EQU   $BF00
                        4    BELL       EQU   $FF3A
                        5    HEXOUT     EQU   $FDDA
                        6    *
                        7    * Get File Info $C4
                        8    * ==================
                        9    *
                       10    * Liest Dateieintrag von
                       11    * Datei "TEXTFILE" ein, die
                       12    * man zu Testzwecken vorher
                       13    * anlegen und LOCKen sollte
                       14    *
6000: A9 0A            15               LDA   #$0A
6002: 8D 3C 60         16               STA   COUNT
6005: 20 00 BF         17               JSR   MLI
6008: C4               18    COMMAND1   HEX   C4          ;Get Info
6009: 3C 60            19               DA    COUNT
600B: D0 29            20               BNE   ERROR
                       21    *
                       22    * Set File Info $C3
                       23    * ==================
                       24    *
                       25    * Verwandelt Textfile ($04)
                       26    * in Binärfile ($06)
                       27    * mit Startadresse $1000
                       28    * und entfernt ein ggf.
                       29    * vorhandenes LOCK-Flag.
                       30    *
600D: A9 07            31               LDA   #$07
600F: 8D 3C 60         32               STA   COUNT
                       33    *
6012: AD 40 60         34               LDA   FILETYPE
6015: C9 04            35               CMP   #$04         :text
6017: D0 20            36               BNE   ERROR1
6019: A9 06            37               LDA   #$06         ;binär
601B: 8D 40 60         38               STA   FILETYPE
                       39    *
601E: A9 00            40               LDA   #$00
6020: 8D 41 60         41               STA   AUXTYPE
6023: A9 10            42               LDA   #$10         ;$1000
6025: 8D 42 60         43               STA   AUXTYPE+1
                       44    *
6028: A9 C3            45               LDA   #$C3         ;UNLOCK
602A: 8D 3F 60         46               STA   ACCESS
                       47    *
602D: 20 00 BF         48               JSR   MLI
6030: C3               49    COMMAND2   HEX   C3          ;Set Info
6031: 3C 60            50               DA    COUNT
6033: D0 01            51               BNE   ERROR
6035: 60               52               RTS
                       53    *
```

```
6036: 20 DA FD  55   ERROR    JSR  HEXOUT
6039: 4C 3A FF  56   ERROR1   JMP  BELL
                57   *
                58   * Kombinierte Parameter-Liste
                59   * für Get und Set File Info
                60   *
603C: 00        61   COUNT    HEX  00          ;Poke
603D: 4E 60     62   NAMEPTR  DA   LÄNGE
603F: 00        63   ACCESS   HEX  00          ;Poke
6040: 00        64   FILETYPE HEX  00          ;Poke
6041: 00 00     65   AUXTYPE  HEX  0000        ;Poke
                66   *
                67   * Storage + Blocks used sind
                68   * Nullfield bei Set File Info
                69   *
6043: 00        70   STORAGE  HEX  00          ;Poke
6044: 00 00     71   BLOCKS   HEX  0000        ;Poke
                72   *
6046: 00 00     73   MDATE    HEX  0000        ;Poke
6048: 00 00     74   MTIME    HEX  0000        ;Poke
                75   *
                76   * Creation Date und Time
                77   * entfallen bei Set File Info
                78   *
604A: 00 00     79   CDATE    HEX  0000        ;Poke
604C: 00 00     80   CTIME    HEX  0000        ;Poke
                81   *
604E: 11        82   LÄNGE    DFB  #STRICH2-STRICH1+1
604F: 2F        83   STRICH1  ASC  '/'
6050: 56 4F 4C  84            ASC  'VOLUME/TEXTFILE'
6053: 55 4D 45 2F 54 45 58 54
605B: 46 49 4C 45
605F: 2F        85   STRICH2  ASC  '/'

--End assembly--

96 bytes

Errors: 0
```

```
:ASM                    1           ORG   $6000
                        2   *
                        3   IND     EQU   $CE         ;-$CF
                        4   MLI     EQU   $BF00
                        5   BELL    EQU   $FF3A
                        6   PRINT   EQU   $FDED
                        7   HEXOUT  EQU   $FDDA
                        8   *
                        9   * On Line $C5
                       10   * ============
                       11   *
6000: A9 0B             12   ONLINE  LDA   #<NAMEN
6002: 85 CE             13           STA   IND
6004: A9 61             14           LDA   #>NAMEN
6006: 85 CF             15           STA   IND+1
                       16   *
                       17   * 256-Byte-Puffer löschen
                       18   * Status-Register auf Null!
                       19   *
6008: A0 00            20           LDY   #0
600A: 84 48            21           STY   $48         ;P-Reg.
600C: 98               22           TYA
600D: 91 CE            23   ONLINE1 STA   (IND),Y
600F: C8               24           INY
6010: D0 FB            25           BNE   ONLINE1
                       26   *
6012: B9 02 61         27   ONLINE2 LDA   ONLINE4,Y
6015: F0 06            28           BEQ   ONLINE3
6017: 20 ED FD         29           JSR   PRINT
601A: C8               30           INY
601B: D0 F5            31           BNE   ONLINE2
                       32   *
                       33   * On Line ausführen
                       34   *
601D: 20 00 BF         35   ONLINE3 JSR   MLI
6020: C5               36   COMMAND1 HEX  C5          ;Online
6021: 28 60            37           DA    COUNT1
6023: F0 07            38           BEQ   LOOP0
6025: 4C AC 60         39           JMP   ERROR
                       40   *
6028: 02               41   COUNT1  HEX   02
6029: 00               42   UNITNUM HEX   00          ;alle
602A: 0B 61            43           DA    NAMEN
                       44   *
                       45   * Endmarker erreicht?
                       46   *
602C: A0 00            47   LOOP0   LDY   #0
602E: B1 CE            48           LDA   (IND),Y
6030: D0 08            49           BNE   LOOP1
6032: C8               50           INY
6033: B1 CE            51           LDA   (IND),Y
6035: D0 03            52           BNE   LOOP1
6037: 4C B2 60         53           JMP   PREFIX      ;fertig
```

```
                        55   *
603A: A9 8D             56   LOOP1    LDA  #$8D
603C: 20 ED FD          57            JSR  PRINT
603F: A0 00             58            LDY  #0
                        59   *
                        60   * Slot anzeigen
                        61   *
6041: A9 D3             62            LDA  #"S"
6043: 20 ED FD          63            JSR  PRINT
6046: B1 CE             64            LDA  (IND),Y
6048: 29 70             65            AND  #%01110000
604A: 4A                66            LSR
604B: 4A                67            LSR
604C: 4A                68            LSR
604D: 4A                69            LSR
604E: 09 B0             70            ORA  #$B0        ;0
6050: 20 ED FD          71            JSR  PRINT       ;Slot
6053: A9 AC             72            LDA  #","
6055: 20 ED FD          73            JSR  PRINT
6058: A9 A0             74            LDA  #$A0
605A: 20 ED FD          75            JSR  PRINT
605D: B1 CE             76            LDA  (IND),Y
605F: 29 0F             77            AND  #%00001111
6061: 8D 0B 62          78            STA  LENGTH
                        79   *
                        80   * Drive anzeigen
                        81   *
6064: A9 C4             82            LDA  #"D"
6066: 20 ED FD          83            JSR  PRINT
6069: B1 CE             84            LDA  (IND),Y
606B: 10 04             85            BPL  LOOP2
606D: A9 B2             86            LDA  #"2"        ;D2
606F: D0 02             87            BNE  LOOP3
6071: A9 B1             88   LOOP2    LDA  #"1"        ;D1
6073: 20 ED FD          89   LOOP3    JSR  PRINT
6076: A9 BA             90            LDA  #":"
6078: 20 ED FD          91            JSR  PRINT
                        92   *
607B: AD 0B 62          93            LDA  LENGTH      ;Len = 0
607E: F0 17             94            BEQ  LOOP6       ;Fehler!
6080: A9 AF             95            LDA  #"/"
6082: 20 ED FD          96            JSR  PRINT
                        97   *
                        98   * Name anzeigen
                        99   *
6085: C8                100  LOOP4    INY
6086: CC 0B 62          101           CPY  LENGTH
6089: 90 02             102           BCC  LOOP5
608B: D0 0F             103           BNE  LOOP7
                        104  *
608D: B1 CE             105  LOOP5    LDA  (IND),Y
608F: 09 80             106           ORA  #$80
6091: 20 ED FD          107           JSR  PRINT
```

```
6094: 4C 85 60  109           JMP  LOOP4
                110  *
                111  * Name leer, d.h. Diskette
                112  * nicht eingelegt: I/O-Error
                113  *
6097: A9 BF     114  LOOP6     LDA  #"?"
6099: 20 ED FD  115           JSR  PRINT
                116  *
                117  * Nächster Name
                118  *
609C: 18        119  LOOP7     CLC
609D: A5 CE     120           LDA  IND
609F: 69 10     121           ADC  #16
60A1: 85 CE     122           STA  IND
60A3: A5 CF     123           LDA  IND+1
60A5: 69 00     124           ADC  #0
60A7: 85 CF     125           STA  IND+1
60A9: 4C 2C 60  126           JMP  LOOP0
                127  *
60AC: 20 DA FD  128  ERROR     JSR  HEXOUT
60AF: 4C 3A FF  129           JMP  BELL
                130  *
                131  * Get Prefix $C7
                132  * ==============
                133  *
60B2: 20 00 BF  134  PREFIX    JSR  MLI
                135  *
60B5: C7        136  COMMAND2 HEX  C7          ;Get P.
60B6: BC 60     137           DA   COUNT2
60B8: F0 05     138           BEQ  PREFIX0
60BA: D0 F0     139           BNE  ERROR
                140  *
60BC: 01        141  COUNT2    HEX  01
60BD: 0B 61     142           DA   NAMEN
                143  *
60BF: A0 00     144  PREFIX0   LDY  #0
60C1: B9 F8 60  145  PREFIX1   LDA  PREFIX3,Y
60C4: F0 06     146           BEQ  PREFIX2
60C6: 20 ED FD  147           JSR  PRINT
60C9: C8        148           INY
60CA: D0 F5     149           BNE  PREFIX1
                150  *
60CC: A9 0B     151  PREFIX2   LDA  #<NAMEN
60CE: 85 CE     152           STA  IND
60D0: A9 61     153           LDA  #>NAMEN
60D2: 85 CF     154           STA  IND+1
60D4: A0 00     155           LDY  #0
60D6: B1 CE     156           LDA  (IND),Y
60D8: 8D 0B 62  157           STA  LENGTH
60DB: D0 08     158           BNE  DISPLAY
                159  *
                160  * Kein Präfix!
                161  *
```

```
60DD: A9 BF     163           LDA  #"?"
60DF: 20 ED FD  164           JSR  PRINT
60E2: 4C ED 60  165           JMP  EXIT
                166  *
60E5: C8        167  DISPLAY  INY
60E6: CC 0B 62  168           CPY  LENGTH
60E9: 90 03     169           BCC  DISPLAY1
60EB: F0 01     170           BEQ  DISPLAY1
60ED: 60        171  EXIT     RTS              ;Exit
                172  *
60EE: B1 CE     173  DISPLAY1 LDA  (IND),Y
60F0: 09 80     174           ORA  #$80
60F2: 20 ED FD  175           JSR  PRINT
60F5: 4C E5 60  176           JMP  DISPLAY
                177  *
60F8: 8D 8D     178  PREFIX3  HEX  8D8D
60FA: D0 F2 FB  179           ASC  "Präfix:"
60FD: E6 E9 F8 BA
6101: 00        180           HEX  00
                181  *
6102: 8D        182  ONLINE4  HEX  8D
6103: CF EE EC  183           ASC  "Online"
6106: E9 EE E5
6109: 8D 00     184           HEX  8D00
                185  *
                186  * 256-Byte-Puffer
                187  *
                188  NAMEN     DS   256
                189  *
620B: 00        190  LENGTH   HEX  00

--End assembly--

524 bytes

Errors: 0
```

```
                        1               ORG   $6000
                        2    *
                        3    PROMPT     EQU   $33
                        4    PUFFER     EQU   $0200
                        5    MLI        EQU   $BF00
                        6    BELL       EQU   $FF3A
                        7    PRINT      EQU   $FDED
                        8    HEXOUT     EQU   $FDDA
                        9    GETLN      EQU   $FD6A
                       10    *
                       11    * Set Prefix $C6
                       12    * ==============
                       13    *
6000: A5 33            14    GETP1      LDA   PROMPT
6002: 8D 71 60         15               STA   PROMPT1
6005: A9 AF            16               LDA   #"/"
6007: 85 33            17               STA   PROMPT
6009: A2 00            18               LDX   #0
600B: BD 72 60         19    GETP2      LDA   PREFIX1,X
600E: F0 06            20               BEQ   GETP3
6010: 20 ED FD         21               JSR   PRINT
6013: E8               22               INX
6014: D0 F5            23               BNE   GETP2
6016: 20 6A FD         24    GETP3      JSR   GETLN
6019: E0 01            25               CPX   #1
601B: 90 E3            26               BCC   GETP1       ;nur Rtn
601D: A2 00            27               LDX   #0
                       28    *
                       29    * Legales Präfix ?
                       30    *
601F: BD 00 02         31    GETP4      LDA   PUFFER,X    ;Loop
6022: 29 7F            32               AND   #%01111111
                       33    *
6024: C9 0D            34               CMP   #$0D        ;Return
6026: F0 28            35               BEQ   GETP8
                       36    *
6028: C9 2E            37               CMP   #'.'        ;Punkt
602A: F0 16            38               BEQ   GETP6
602C: C9 2F            39               CMP   #'/'
602E: F0 12            40               BEQ   GETP6       ;Strich
                       41    *
                       42    * A-Z ?
                       43    *
6030: C9 41            44               CMP   #'A'
6032: 90 06            45               BCC   GETP5       ;<A
6034: C9 5B            46               CMP   #'Z'
6036: B0 C8            47               BCS   GETP1       ;>Z
6038: 90 0C            48               BCC   GETP7       ;A-Z!
                       49    *
                       50    * 0-9 ?
                       51    *
603A: C9 30            52    GETP5      CMP   #'0'
603C: 90 C2            53               BCC   GETP1       ;<0
```

```
603E: C9 3A      55            CMP  #':'
6040: B0 BE      56            BCS  GETP1       ;>9
                 57   *
                 58   * Ziffer oder Punkt
                 59   * erstes Zeichen ?
                 60   *
6042: E0 00      61   GETP6    CPX  #0
6044: F0 BA      62            BEQ  GETP1
                 63   *
6046: 9D 7D 60   64   GETP7    STA  NAME1,X
6049: E8         65            INX
604A: E0 80      66            CPX  #128
604C: D0 D1      67            BNE  GETP4
604E: F0 B0      68            BEQ  GETP1
                 69   *
6050: A9 2F      70   GETP8    LDA  #'/'
6052: 9D 7D 60   71            STA  NAME1,X
6055: E8         72            INX              ;siehe
6056: E8         73            INX              ;unten
6057: 8E 7B 60   74            STX  LÄNGE
                 75   *
                 76   * Set Prefix
                 77   *
605A: 20 00 BF   78            JSR  MLI
605D: C6         79   COMMAND  HEX  C6          ;Set P.
605E: 68 60      80            DA   COUNT
6060: D0 09      81            BNE  ERROR
6062: AD 71 60   82            LDA  PROMPT1
6065: 85 33      83            STA  PROMPT
6067: 60         84            RTS              ;Exit
6068: 01         85   COUNT    HEX  01
6069: 7B 60      86            DA   LÄNGE
                 87   *
606B: 20 DA FD   88   ERROR    JSR  HEXOUT
606E: 4C 3A FF   89            JMP  BELL
                 90   *
6071: 00         91   PROMPT1  HEX  00
                 92   *
6072: 8D         93   PREFIX1  HEX  8D
6073: D0 F2 FB   94            ASC  "Präfix "
6076: E6 E9 F8 A0
607A: 00         95            HEX  00
                 96   *
                 97   * 00 01 02 03 04 05 06  Länge-1
                 98   * N  A  M  E  8D        Puffer
                 99   * /  N  A  M  E  /       Name
                100   *
607B: 00        101   LÄNGE    HEX  00
607C: 2F        102   NAME     ASC  '/'
                103   NAME1    DS   127
```

1.5.4. Spezielle Datei-Befehle

Hinweis: Zu diesem Kapitel gehören die ,,Programmbeispiele zu Kap. 1.5.4'' ab Seite 183.

Die speziellen Datei-Befehle, die im ,,Technical Manual'' als ,,Filing Calls'' bezeichnet werden, dienen dem Einlesen und der Speicherung der eigentlichen Datei, nicht dem Dateieintrag, der – mit Ausnahme von SET EOF – durch das MLI automatisch geändert wird (z.B. Hauptzeiger zum Dateianfang, Sämling-Schößling-Baum-Vermerk, Anzahl der belegten Blocks usw.).

$C8 = OPEN = Datei öffnen und I/O-Puffer anlegen
$C9 = NEWLINE = Endmarker definieren
$CA = READ = eine bestimmte Anzahl von Bytes einlesen
$CB = WRITE = eine bestimmte Anzahl von Bytes speichern
$CC = CLOSE = Datei definitiv schließen
$CD = FLUSH = I/O-Puffer-Inhalt speichern, ohne Datei zu schließen
$CE = SET MARK = Positionszeiger definieren
$CF = GET MARK = momentanen Positionszeiger ermitteln
$D0 = SET EOF = EOF definieren
$D1 = GET EOF = EOF ermitteln
$D2 = SET BUF = I/O-Pufferadresse neu festlegen
$D3 = GET BUF = momentane I/O-Pufferadresse ermitteln

OPEN ($C8) – Parameter

Parameter-Count = $03 – 1 Byte
Name-Pointer – LL HH – 2 Bytes
I/O-Buffer-Pointer – z.B. 00 96 = $9600 – LL HH – 2 Bytes
File-Reference-Number – 1 Byte; wird vom MLI gepokt

OPEN ist der grundlegende Befehl, da er allen anderen speziellen Dateibefehlen vorausgehen muß. Mit OPEN wird der I/O-Puffer festgelegt, der auf einer Seitengrenze beginnen und $0400 Bytes umfassen muß (z.B. $9600-$99FF) und der bis zum CLOSE (oder SET BUF) aktiv bleibt. Die erste Hälfte des I/O-Puffers nimmt den jeweiligen Datenblock auf, während in der zweiten Hälfte der jeweils benötigte Index-Block abgelegt wird. Das MLI pokt nach OPEN in die Parameter-Liste eine File-Verweisnummer im Bereich 1-8 (höchstens Maxfiles 8) und überträgt den momentanen Dateieintrag vom Disketten-Catalog in den File Control Block. Die File-

Verweisnummer muß bei allen nachfolgenden speziellen Dateibefehlen verwendet werden.

Wenn man PRODOS *und* BASIC.SYSTEM gleichzeitig benutzt, sollte man vor OPEN das Level-Byte in der PRODOS Global Page mit

LDA #$00
STA $BF94

auf Null setzen, damit man später mit CLOSE + File-Reference-Number = $00 *alle* OPEN-Files schließen oder ,,flushen'' kann.

NEWLINE ($C9) – Parameter

Paramter-Count = $03 – 1 Byte
File-Reference-Number – 1 Byte
Enable-Mask = z.B. $7F – 1 Byte
Newline-Character = z.B. $0D – 1 Byte

Wenn die UND-Maske = $00 ist, ist NEWLINE inaktiv (Näheres siehe Kapitel 1.5.1, laufende Nr. 21).

READ ($CA) – Parameter

Parameter-Count = $04 – 1 Byte
File-Reference-Number – 1 Byte
Data-Buffer-Pointer – LL HH – 2 Bytes
Request-Count – z.B. 00 20 = $ 2000 Bytes – LL HH – 2 Bytes
Transfer-Count – LL HH – 2 Bytes; werden vom MLI gepokt

Wenn das Newline-Byte gesetzt ist, dann werden alle Bytes ab Positionszeiger bis einschließlich zum Newline-Byte übertragen. Andernfalls werden die durch den Request-Count vorgegebenen Bytes übertragen.

WRITE ($CB) – Parameter

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; werden vom MLI gepokt

CLOSE ($CC) – Parameter

Parameter-Count = $01 – 1 Byte
File-Reference-Number – 1 Byte

FLUSH ($CD) – Parameter

Parameter-Count = $01 – 1 Byte
File-Reference-Number – 1 Byte

FLUSH überträgt durch MLI-internes WRITE den I/O-Puffer auf den OPEN-File und aktualisiert den Dateieintrag. CLOSE gibt zusätzlich den FCB-Eintrag und die File-Verweisnummer frei. Außerdem wird das Backup-Bit im Access-Byte gesetzt. File-Verweisnummer = $00 + Level-Byte = $00 schließt *alle* OPEN-Files.

SET MARK ($CE) – Parameter

Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
Position – LL MM HH – 3 Bytes

GET MARK ($CF) – Parameter

Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
Position – LL MM HH – 3 Bytes; werden vom MLI gepokt

Der Positionszeiger ist die absolute File-Position des momentan mit READ oder WRITE übertragenen Bytes. Der Positionszeiger ist nach OPEN auf Null gesetzt und kann das Dateiende (EOF) nicht überschreiten, d.h. formelmäßig:

0 > = Position < = EOF

Mit GET MARK kann der Positionszeiger willkürlich, d.h. ohne READ/WRITE, wodurch der Zeiger automatisch erhöht wird, herauf- oder herabgesetzt werden.

SET EOF ($D0) – Parameter

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

GET EOF ($D1) – Parameter

Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
EOF – LL MM HH – 3 Bytes; werden vom MLI gepokt

Wenn das monentane EOF reduziert wird, dann werden die entsprechenden Datenblocks auf der Diskette freigegeben. Wenn umgekehrt das momentane EOF erhöht wird, entstehen die berüchtigten nicht-vorformatierten ,,Loch-Dateien'' oder ,,Sparse Files'' (siehe hierzu ,,DOS-Buch'', S. 58ff. sowie ,,Technical Manual'', S. 149ff.), die unbedingt zu vermeiden sind, da sie mit dem ,,Rest der Welt'' nicht kompatibel sind. Beispielsweise kann das CONVERT-Programm derartige Dateien nicht konvertieren.

SET BUF ($D2) – Parameter

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

GET BUF ($D3) – Parameter

Parameter-Count = $02 – 1 Byte
File-Reference-Number – 1 Byte
I/O-Buffer-Pointer – LL HH – 2 Bytes; werden vom MLI gepokt

Mit SET BUF kann *nach* OPEN ein neuer I/O-Puffer festgelegt werden, wobei der Inhalt des alten in den neuen übertragen und der alte freigegeben wird.

```
                        1               ORG  $6000
                        2    *
                        3    * Write-Demo/16.05.84/U.Stiehl
                        4    * ============================
                        5    *
                        6    * CREATE-OPEN-NEWLINE-WRITE-CLOSE
                        7    *
                        8    * Der per MLI erzeugte Textfile
                        9    * kann zu Testzwecken mit dem
                        10   * folgenden Applesoft-Programm
                        11   * eingelesen werden.
                        12   * Weitere, größere MLI-Programme
                        13   * finden sich im Band 2 von
                        14   * "ProDOS für Aufsteiger".
                        15   *
                        16   * 10 PRINT CHR$ (4) "OPEN TEXTFILE"
                        17   * 20 PRINT CHR$ (4) "READ TEXTFILE"
                        18   * 30 FOR X = 1 TO 64
                        19   * 40 INPUT X$: PRINT X$: NEXT
                        20   * 50 PRINT CHR$ (4) "CLOSE"
                        21   *
                        22   IND       EQU  $CE          ;-$CF
                        23   MLI       EQU  $BF00
                        24   LEVEL     EQU  $BF94
                        25   BELL      EQU  $FF3A
                        26   HEXOUT    EQU  $FDDA
                        27   *
                        28   * Beginn + Ende des Datenpuffers
                        29   * Dateilänge = EOF = $2000 = 8192
                        30   *
                        31   PUFBEG    EQU  $1000
                        32   PUFEND    EQU  $3000
                        33   *
                        34   * Create $C0
                        35   * ==========
                        36   *
6000: 20 00 BF          37             JSR  MLI
6003: C0                38   COMMAND1  HEX  C0          ;Create
6004: 0B 60             39             DA   COUNT1
6006: F0 21             40             BEQ  OPEN1
6008: 4C B2 60          41             JMP  ERROR1
                        42   *
600B: 07                43   COUNT1    HEX  07
600C: 17 60             44   NAME1     DA   LÄNGE
600E: C3                45   ACCESS    HEX  C3          ;UNLOCK
600F: 04                46   FILETYPE  HEX  04          ;Textfile
6010: 00 00             47   AUXTYPE   HEX  0000        ;keiner
6012: 01                48   STORAGE   HEX  01          ;Sämling
6013: 00 00             49   CDATE     HEX  0000        ;leer
6015: 00 00             50   CTIME     HEX  0000        ;leer
                        51   *
                        52   * Vollständiger Dateiname
                        53   *
```

```
6017: 11          55   LÄNGE    DFB  #STRICH2-STRICH1+1
6018: 2F          56   STRICH1  ASC  '/'
6019: 56 4F 4C    57            ASC  'VOLUME/TEXTFILE'
601C: 55 4D 45 2F 54 45 58 54
6024: 46 49 4C 45
6028: 2F          58   STRICH2  ASC  '/'
                  59   *
                  60   * Open $C8
                  61   * ========
                  62   *
6029: A9 00       63   OPEN1    LDA  #0
602B: 8D 94 BF    64            STA  LEVEL
602E: 20 00 BF    65            JSR  MLI
6031: C8          66   COMMAND2 HEX  C8          ;Open
6032: 39 60       67            DA   COUNT2
6034: F0 09       68            BEQ  NEWLINE1
6036: 4C B2 60    69            JMP  ERROR1
                  70   *
6039: 03          71   COUNT2   HEX  03
603A: 17 60       72            DA   LÄNGE
603C: 00 96       73   IOPUFER  DA   $9600       ;-$99FF
603E: 00          74   FILENO1  HEX  00          ;poken
                  75   *
                  76   * Newline $C9
                  77   * ===========
                  78   *
603F: AD 3E 60    79   NEWLINE1 LDA  FILENO1
6042: 8D 51 60    80            STA  FILENO2
                  81   *
6045: 20 00 BF    82            JSR  MLI
6048: C9          83   COMMAND3 HEX  C9          ;Newline
6049: 50 60       84            DA   COUNT3
604B: F0 07       85            BEQ  WRITE1
604D: 4C B2 60    86            JMP  ERROR1
                  87   *
6050: 03          88   COUNT3   HEX  03
6051: 00          89   FILENO2  HEX  00          ;poken
6052: 00          90   ENABLE   HEX  00          ;NEWLINE
6053: 00          91   ENDCHAR  HEX  00          ;entfällt
                  92   *
                  93   * Write $CB
                  94   * =========
                  95   *
                  96   * Datenpuffer mit Test-Strings
                  97   * = 127 "-" + Return vollschreiben
                  98   *
6054: A9 00       99   WRITE1   LDA  #<PUFBEG
6056: 85 CE       100           STA  IND
6058: A9 10       101           LDA  #>PUFBEG
605A: 85 CF       102           STA  IND+1
605C: A0 00       103           LDY  #0
                  104  *
605E: A9 2D       105  WRITE2   LDA  #'-'         ;Loop
6060: 91 CE       106           STA  (IND),Y
6062: C0 7F       107           CPY  #127
```

```
6064: F0 11      109            BEQ  WRITE4
6066: C0 FF      110            CPY  #255
6068: F0 0D      111            BEQ  WRITE4
                 112  *
606A: C8         113  WRITE3    INY
606B: D0 F1      114            BNE  WRITE2
606D: E6 CF      115            INC  IND+1
606F: A5 CF      116            LDA  IND+1
6071: C9 30      117            CMP  #>PUFEND
6073: 90 E9      118            BCC  WRITE2
6075: B0 06      119            BCS  WRITE5
                 120  *
6077: A9 0D      121  WRITE4    LDA  #$0D        ;Return
6079: 91 CE      122            STA  (IND),Y
607B: D0 ED      123            BNE  WRITE3
                 124  *
                 125  * Write
                 126  *
607D: AD 3E 60   127  WRITE5    LDA  FILENO1
6080: 8D 99 60   128            STA  FILENO3
6083: A9 00      129            LDA  #<PUFBEG
6085: 8D 9A 60   130            STA  DATABUF
6088: A9 10      131            LDA  #>PUFBEG
608A: 8D 9B 60   132            STA  DATABUF+1
                 133  *
608D: 20 00 BF   134            JSR  MLI
6090: CB         135  COMMAND4  HEX  CB          ;Write
6091: 98 60      136            DA   COUNT4
6093: F0 0B      137            BEQ  CLOSE1
6095: 4C B2 60   138            JMP  ERROR1
                 139  *
6098: 04         140  COUNT4    HEX  04
6099: 00         141  FILENO3   HEX  00          ;poken
609A: 00 00      142  DATABUF   HEX  0000        ;poken
609C: 00 20      143  REQUEST   HEX  0020        ;$2000
609E: 00 00      144  TRANSFER  HEX  0000        ;poken
                 145  *
                 146  * Close $CC
                 147  * =========
                 148  *
60A0: AD 3E 60   149  CLOSE1    LDA  FILENO1
60A3: 8D B1 60   150            STA  FILENO4
                 151  *
60A6: 20 00 BF   152            JSR  MLI
60A9: CC         153  COMMAND5  HEX  CC          ;Close
60AA: B0 60      154            DA   COUNT5
60AC: F0 0A      155            BEQ  EXIT1
60AE: D0 02      156            BNE  ERROR1
                 157  *
60B0: 01         158  COUNT5    HEX  01
60B1: 00         159  FILENO4   HEX  00          ;poken
                 160  *
                 161  *
```

```
60B2: 20 DA FD   163  ERROR1   JSR  HEXOUT
60B5: 4C 3A FF   164           JMP  BELL
                 165  *
                 166  * MLI-Programme, die man vom
                 167  * Monitor aus startet, sollten
                 168  * das Status-Register auf Null
                 169  * zurücksetzen, da es speziell
                 170  * im Falle einer Fehlermeldung
                 171  * nicht initialisiert wird.
                 172  *
60B8: A9 00      173  EXIT1     LDA  #0
60BA: 85 48      174           STA  $48        ;P-Reg.
60BC: 60         175           RTS

--End assembly--

189 bytes

Errors: 0
```

Hätten Sie's gewußt?

Bei einem Apple IIe mit 80-Zeichen-Karte gebe man folgendes ein:

PR#3	(Karte einschalten)
ESC	(ESC-Taste tippen)
CTRL-L	(Ctrl- plus L-Taste gleichzeitig tippen)
Folge:	Absturz in den Monitor

Frage: Welches Byte im ROM hätte anders lauten müssen, damit die 80-Zeichen-Routine nicht abstürzt?

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)

2. ProDOS für Applesoft-Programmierer

In diesem Kapitel wird aus Platzgründen eine stark gestraffte Zusammenfassung der BASIC.SYSTEM-Befehle für weit fortgeschrittene Applesoft-Programmierer gegeben, wobei Kenntnisse von DOS 3.3 und dem ,,DOS-Buch'' vorausgesetzt werden.

Hinweis: Der zweite Band von ,,ProDOS für Aufsteiger'' wird die dem Applesoft-Programmierer zur Verfügung stehenden und hier nur kursorisch und mehr theoretisch behandelten Befehle ausführlich mit Beispielprogrammen erläutern.

2.1. Alte DOS-Befehle: FP, INT, MON, NOMON, MAXFILES, INIT

Diese ehemaligen DOS-Befehle sind unter ProDOS verschwunden. FP und INT existieren nicht mehr, weil Integer-Basic unter ProDOS nicht implementiert wurde und von daher ein Umschalten zwischen Applesoft- und Integer-Basic sinnlos geworden ist.

MON und NOMON wurden aus systemimmanenten Gründen nicht in das BASIC.SYSTEM aufgenommen. Besonders bei der Entwicklung von Textfiles bearbeitenden Programmen wird man diese Befehle zum Austesten vermissen.
MAXFILES ist wegen der Garbage Collection und wegen HIMEM verschwunden (siehe unten).

Der INIT-Befehl ist nunmehr nur noch in dem FILER-Programm enthalten. Wenn man Disketten aus einem Anwenderprogramm heraus initialisieren will, muß man sich seine eigene INIT-Routine schreiben (siehe PRODOS.INIT, S. 70).

Von den alten DOS-Befehlen ist NOMON in der BASIC.SYSTEM-Befehlstabelle stehengeblieben. Dies ist damit der einzige der Altbefehle, der keine Fehlermeldung verursacht, aber andererseits auch nichts bewirkt.

2.2. Neue BASIC.SYSTEM-Befehle: PREFIX, CAT, CREATE, FLUSH, STORE, RESTORE, CHAIN, Strich-Befehl

Dies sind die neuen BASIC.SYSTEM-Befehle. Mit PREFIX (= Volume-Directory- und ggf. Subdirectory-Name) kann man ein anderes Präfix definieren. Die Befehle CAT und CATALOG in Verbindung mit Drive und/oder Slot definieren ebenfalls das Präfix neu. Nach den Booten gilt zunächst der Volume-Name der Boot-Diskette als Präfix.

CATALOG ist der neue Catalog-Befehl für 80-Zeichen-Darstellung und CAT ist ein 40-Zeichen-Catalog-Befehl. Der Catalog enthält nunmehr genaue Angaben über die belegten und freien Disketten-Blocks, über die Längen und Startadressen der Files, über das Datum, wann ein File neu angelegt bzw. geändert wurde usw.

FLUSH leert zwar den I/O-Puffer, indem es ihn auf den Textfile überträgt, doch schließt es nicht den File. CLOSE macht beides. Insofern wendet man CLOSE dann an, wenn man eine Datei endgültig schließt und FLUSH dann, wenn man sicherstellen will, daß dem Anwender eines Dateiprogramms durch unsachgemäße Bedienung, z.B. durch Vertauschen von Datendisketten, keine Daten verloren gehen.

STORE speichert *alle* Variablen eines Applesoft-Programms und RESTORE lädt sinngemäß alle Variablen wieder von Diskette ein. Da STORE/RESTORE vom BASIC.SYSTEM abgefangen werden, müßte man erst ProDOS „abhängen", bevor man die gleichnamigen Kassettenrecorder-Befehle ausführen könnte.

Der CHAIN-Befehl, der bislang nur bei Integer-Basic implementiert war, funktioniert nunmehr unter Applesoft und BASIC.SYSTEM, allerdings mit der Einschränkung, daß ein später eingeladenes Modul keinen Array dimensionieren darf, der im vorangehenden Modul bereits definiert worden war. Deshalb kann man das arraydefinierende Modul (= Hauptmodul) nur ein einziges Mal einlesen. Beispiel:

```
10 REM MODUL.1
20 DIM A (100)
30 FOR X = 1 TO 100: A (X) = X: NEXT
40 PRINT CHR$ (4) ,,CHAIN MODUL.2"

10 REM MODUL.2
20 FOR X = 1 TO 100: PRINT A (X): NEXT: REM Zahlen wurden nicht gelöscht
30 PRINT CHR$ (4) ,,CHAIN MODUL.3"

10 REM MODUL.3
20 DIM A (100) : REM bewirkt Redimension-Error
```

Der Strich-Befehl, z.B. ,,–PRODOS", dient als Sammelbefehl für RUN, BRUN und EXEC sowie zum Starten von System-Files.

2.3. Geänderte Befehle: HIMEM, FRE und HGR, HGR2, TEXT

Das BASIC.SYSTEM übernimmt jetzt selbst einen Teil der Speicherorganisation und -verwaltung (Memory Management). HIMEM sollte jetzt normalerweise nicht mehr vom Applesoft-Programmierer definiert oder geändert werden, da das BASIC.SYSTEM entsprechend der benötigten (Textfile-) Puffer HIMEM selbst vermindert und später wieder erhöht. Neben dem weiterhin existierenden Applesoft-FRE-Befehl, der in der alten Form PRINT FRE (0) oder X = FRE (0) gegeben wird, ist im BASIC.SYSTEM eine eigene, übrigens vielfach schnellere Garbage-Collection-Routine implementiert, die jeweils vor dem Anlegen eines neuen Textfile-Puffers und der damit verbundenen Änderung von HIMEM automatisch aufgerufen wird. Darüber hinaus besteht die Möglichkeit, mit FRE (im Direktmodus) oder mit PRINT CHR$ (4) ,,FRE" (im Indirektmodus) die Garbage Collection gewollt herbeizuführen.

HIMEM muß sich auf einer Seitengrenze (durch 256 ohne Rest teilbar) befinden, wie überhaupt die I/O-Puffer sowohl beim BASIC.SYSTEM wie auch beim PRODOS-MLI auf einer Seitengrenze beginnen müssen. HIMEM hat normalerweise den Wert $9600 (dezimal 38400 = PRINT PEEK (115) + PEEK (116) · 256), d.h. von $9600-$99FF liegt der erste und zugleich allgemeine BASIC.SYSTEM-Puffer für Befehle wie CATALOG, BLOAD usw. Würde man HIMEM mit POKE 115, 0 : POKE 116, 146 (= $9200) ändern und dann einen Textfile mit OPEN-WRITE neu

anlegen, dann würde der Bereich $8E00-$91FF als eigentlicher Textfile-Puffer sowie der Bereich $9200-$95FF als weiterer I/O-Puffer benutzt, während der Bereich $9600-$99FF nunmehr nicht mehr angetastet würde. Zusätzlich zur HIMEM-Änderung sollte man jedoch auch die System Bit Map korrigieren, wenn man sichergehen will, daß der oberhalb von HIMEM liegende Speicherbereich nicht angetastet wird.

Nach den Befehlen HGR und HGR2 sollten ursprünglich* das Applesoft-Programm und/oder die Variablen speichermäßig verschoben bzw. gesplittet und nach dem TEXT-Befehl wieder in die alte Speicherposition zurückgebracht werden. Dies war in den Pre-Release-Versionen von ProDOS versuchsweise implementiert. In der ProDOS-Version 1.0 vom 1.1.84 wurden diese BASIC-SYSTEM-Befehle wieder entfernt, d.h. HGR, HGR2 und TEXT sind wieder wie bisher reine Applesoft-Befehle. Ob eine spätere ProDOS-Version hier Änderungen vornimmt, bleibt offen. Auf alle Fälle ist evident, daß diesbezügliche Änderungen jeden Compiler ,,aufs Kreuz legen'' würden.

2.3.1. TRACE, NOTRACE, PR#S, IN#S, Ctrl-D (I/O-Vektoren)

Die Befehle PR#S und IN#S (S = Slot) funktionieren zwar im Prinzip wie unter DOS 3.3, doch wird das Applesoft-Programm nunmehr viel stärker vom BASIC.SYSTEM ,,überwacht'', als dies unter DOS 3.3 der Fall war. Der Grund liegt darin begründet, daß sich Applesoft unter dem BASIC.SYSTEM quasi permanent im Trace-Zustand befindet, wenngleich die Zeilennummern unterdrückt werden, es sei denn, daß der Benutzer das ,,echte'' Trace mit dem TRACE-Befehl einschaltet. Der Leser wird sich fragen, wie dieses ,,unechte'' Pseudo-Trace funktioniert. Dies habe ich auch erst durch Zufall gefunden, als ich mit dem Einzeiler

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

das BASIC.SYSTEM abzuhängen versuchte. Plötzlich erschien am Bildschirm nach RUN

```
# 10
```

und bei erneutem RUN

```
# 10 # 10
```

* Die Angaben im "BASIC Programming with ProDOS", S. 154 und 206 sind falsch bzw. hinfällig.

Nach Disassemblierung der entsprechenden Stellen im BASIC.SYSTEM war die Lösung gefunden: Die Trace-Flag-Speicherstelle $00F2 (dezimal 242) erhält vom BASIC.SYSTEM nach jedem über die normalen Output-Vektoren ausgegebenen Return den Wert $A5 (binär 10100101, Bit 7 gesetzt; bedeutet für Applesoft Trace eingeschaltet), falls $00F2 nicht bereits $A5 enthält. Damit versucht der Applesoft-Interpreter (durch JSR $D810, womit D8 12 auf den Stack geschoben wird, der vom BASIC.SYSTEM auf diese zwei Bytes untersucht wird) bei jeder neuen Programmzeile sowie bei jedem Doppelpunkt (Statement-Separator) innerhalb einer Zeile die entsprechende Zeilennummer über die Output-Vektoren anzuzeigen. Das BASIC.SYSTEM fängt diesen PRINT ab und unterdrückt ihn, falls der echte Trace-Befehl nicht zuvor von ihm registiert wurde. Mit dem Einzeiler

```
10 POKE 242, 0: PR#0: IN#0
```

könnte man das BASIC.SYSTEM definitiv abhängen mit der Folge, daß nunmehr #10 nicht mehr angezeigt wird. Poke 242, 0 allein würde nicht genügen, weil diese Speicherstelle sofort wieder von BASIC.SYSTEM korrigiert werden würde.

In Ergänzung zu den „normalen" Input-Output-Vektoren (siehe Kapitel 1.3.6.2) gibt es im BASIC.SYSTEM noch weitere „geheime", d.h. von der Firma Apple nicht dokumentierte Vektoren, die das Applesoft-Programm durch das Pseudo-Trace sowie bei Ctrl-D noch stärker an die Kandare nehmen:

```
0: $B851: 2F  9A  = $9A2F (O)  Restore I/O-Handler
   $BE53: BA  9A  = $9ABA (I)

4: $B855: 5B  9E  = $9E5B (O)  Trace-Handler 1
   $B857: A5  9B  = $9BA5 (I)

8: $B859: FB  9D  = $9DFB (O)  Trace-Handler 2
   $B85B: A5  9B  = $9BA5 (I)
```

Die Kopierroutine ab $9FAD wird mit dem X-Register (X = 0 oder X = 4 oder X = 8) von den Stellen $9A17 (X = 0), $9A46 (X = 4), $9B60 (X = 0), $9DE8 (X = 8), $9DF0 (X = 4), $9E25 (X = 4) und $AC28 (X = 4) angesprungen, wobei je nach X-Wert die entsprechenden 4 Bytes der obigen 3 je 4 Bytes umfassenden I/O-Vektoren nach $BE38-$BE3B (= eigentlicher I/O-Handler) kopiert werden.

Das obligatorische Ctrl-D ist auch unter ProDOS erforderlich. Allerdings unterdrückt das Semikolon jetzt nicht mehr immer das Erkennen des BASIC.SYSTEM-Befehls. Beispiel:

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

In Zeile 10 folgt Ctrl-D unmittelbar auf einen String, der nicht mit Return abgeschlossen wird. Fazit: Der CATALOG-Befehl wird nicht erkannt.
In Zeile 20 ist ein Doppelpunkt eingefügt. Deshalb wird das Pseudo-Trace aktiv. Fazit: Der CATALOG-Befehl wird trotz des fehlenden Returns ausgeführt.

In dem Moment, da Ctrl-D erkannt wird, werden die obigen ,,geheimem" I/O-Vektoren dergestalt aktiviert, daß das normale COUT ($FDED) lahmgelegt wird mit der Folge, daß erstens MON nicht mehr möglich ist und zweitens BASIC.SYSTEM-Befehle nicht mehr über COUT ausgegeben, sondern statt dessen im Eingabe-Puffer ab $0200 abgelegt werden. Die ,,geheimem" Vektoren werden indes nur dann aktiviert, wenn Ctrl-D von einem normalen Applesoft-PRINT-Statement herrührt. Eine Assembler-COUT-Routine würde nicht erkannt und demzufolge nur über den Bildschirm ausgegeben, ohne daß der BASIC.SYSTEM-Befehl ausgeführt würde. Beispiel:

```
START   LDX   #$00
LOOP    LDA   BEFEHL,X
        BEQ   EXIT
        JSR   COUT
        INX
        BNE   LOOP
EXIT    RTS
BEFEHL  HEX   8D84
        ASC   ,,CATALOG"
        HEX   8D00
```

Daß sich das BASIC.SYSTEM bei jedem Statement-Doppelpunkt und bei jeder neuen Programmzeile einschaltet, bleibt nicht ohne Folgen für die Verarbeitungsgeschwindigkeit. Zu diesem Zweck wurden zwei kleine Tests A und B durchgeführt, die exakt in der vorliegenden Form eingegeben werden müssen, wenn man zu denselben Testergebnissen gelangen will:

Test A

```
10 FOR X = 1 TO 25000
11 REM
12 REM
13 REM
14 REM
15 REM
16 REM
17 REM
18 REM
20 NEXT
```

Test B

```
10 FOR X = 1 TO 25000
15 ::::::::::::::
20 NEXT
```

Test	BASIC.SYSTEM	DOS 3.3	,,abgehängt''
A	100 s	81 s	81 s
B	100 s	68 s	68 s

Die Tests A und B wurden so gewählt, daß sie unter dem BASIC.SYSTEM mit aktiven I/O-Vektoren exakt 100 Sekunden dauerten. Es zeigte sich dann, daß im Vergleich dazu unter DOS 3.3 sowie mit ,,abgehängten'' BASIC.SYSTEM bzw. DOS 3.3 (PR#0: IN#0) die Zeitersparnis 19% bzw. 32% beträgt, d.h. Applesoft ist unter dem BASIC.SYSTEM aufgrund dieser Pseudo-Trace-Aktivitäten u.U. ein Drittel langsamer als normales Applesoft! Nun muß man jedoch berücksichtigen, daß die Beispiele konstruiert sind und deshalb die Verzögerung durch das Pseudo-Trace besonders deutlich zutage tritt. Daher wurde als praxisnahes Beispiel ein von mir in Applesoft geschriebenes, weitgehend exakt ,,geeichtes'' Lores-Digital-Uhr-Programm herangezogen. Hier zeigte sich, daß unter dem BASIC.SYSTEM diese Uhr nach 3 Minuten bereits 7 Sekunden nachging. Dies sind immerhin eine Geschwindigkeitsreduzierung von 4%.

Die Garbage Collection sowie das oben skizzierte Pseudo-Trace haben zur Folge, daß die bisherigen Applesoft-Compiler (TASC, Expediter, Speedstar, Hayden) unter dem BASIC.SYSTEM nicht mehr lauffähig sind, insbesondere weil BASIC.SYSTEM-Befehle nicht erkannt und deshalb ignoriert werden und weil das BASIC.SYSTEM die String-Pointer des Compilers falsch interpretieren würde.

2.4. Neue Befehlsparameter

Das BASIC.SYSTEM ist gegenüber DOS 3.3 um mehrere zum Teil sehr mächtige Befehlsparameter erweitert worden. Im einzelnen gibt es folgende Parameter:

Name (/n)

Name im Sinne von vollständigem Dateinamen (= Pathname), d.h. Kombination aus Präfix (= ,,/'' + ,,Volume-Directory-Name'' + ,,/'' + ,,Subdirectory-Name'' + ,,/'' + ,,Subsubdirectory-Name'' + ,,/'') + ,,eigentlichem Dateinamen'' + ,,/'' (fakultativ), z.B.

/VOLUME/SUBDIRECTORY/SUBSUBDIRECTORY/BASIC.PROGRAM/

Slot- und Drive-Nummer (Ss, Dd)

Wie bei DOS 3.3, doch weniger gebräuchlich aufgrund des vollständigen Dateinamens. Slot im Bereich 1-7, Drive im Bereich 1-2.

Feld-Nummer (Ff)

Relatives Feld ab momentaner Feldposition (nur Vorwärtszeiger). Bereich 0-65535.

Record-Nummer (Rr)

Absolute Datensatz-Nummer (ab Dateibeginn gerechnet) beim Random-File, der zuvor mit Länge-Parameter geöffnet worden sein muß. Bereich 0-65535.

Ff und Rr sind aus Gründen der DOS 3.3-Kompatibilität teils austauschbar, z.B. beim EXEC- und POSITION-Befehl.

Byte-Nummer (Bb)

Bei READ und WRITE relative Byte-Nummer ab momentaner Feldposition (nur Vorwärtszeiger). Bereich 0 bis EOF. Bei BRUN, BLOAD und BSAVE absolute Byte-Nummer (ab Dateibeginn gerechnet).

Anfangsadresse (Aa)

Wie bei DOS 3.3, jedoch im Bereich $0000 – $FFFF (theoretisch).

Länge (Ll)

Wie bei DOS 3.3, jedoch im Bereich $0001 – $FFFF (theoretisch).

Endadresse (Ee)

Dieser Parameter ist neu. Statt BSAVE BILD, A$2000, L$2000 kann man jetzt BSAVE BILD A$2000, E$3FFF sagen. Die Endadresse bezieht sich auf das letzte Byte und die Anfangsadresse auf das erste Byte des (Binär-) Files. Ee und Ll schließen sich gegenseitig aus, d.h. man muß entweder den einen oder den anderen Parameter nehmen, nicht aber beide gleichzeitig. Bereich $0000 – $FFFF (theoretisch).

Zeilen-Nummer-Beginn (§z)

Bei RUN und CHAIN wird mit z.B.

RUN BASIC.PROGRAMM, §1000

zwar das ganze Programm eingeladen, jedoch mit Zeile 1000 gestartet. Der Bereich 0 – 65535, d.h. über 63999 hinaus!

Typ des Files (Tt)

Auch dieser Parameter ist neu. Die Syntax ist „T“ + 3stelliger File-Typ, z.B.

TDIR
TTXT
TBIN
TREL
TBAS
TVAR
TSYS

Dieser Parameter spielt besonders bei CREATE sowie bei BSAVE/BLOAD eine Rolle.

Gegenüber DOS 3.3 ist der Volume-Parameter Vv bei ProDOS verschwunden, doch führt er zu keiner Fehlermeldung.

2.5. Kurzbeispiele

Nachfolgend werden die BASIC.SYSTEM-Befehle anhand kurzer Beispiele erläutert. Im übrigen sei auf den in Vorbereitung befindlichen zweiten Band von „ProDOS für Aufsteiger" verwiesen.

CAT und CATALOG (CAT /n, Ss, Dd)

CAT/VOLUME/SUBDIRECTORY
CAT, D2

PREFIX (PREFIX /n, Ss, Dd)

PREFIX/VOLUME/SUBDIRECTORY
PREFIX/ (Null-Präfix)
PREFIX (zeigt momentanes Präfix an)

CREATE (CREATE /n, Tt, Ss, Dd)

CREATE SUBDIRECTORY (ohne T-Parameter stets Subdirectory)
CREATE BINAERFILE, TBIN
CREATE SYSTEMPROGRAMM, TSYS

RENAME (RENAME /n1, /n2, Ss, Dd)

RENAME ALTNAME, NEUNAME

DELETE (DELETE /n, Ss, Dd)

DELETE DEMOS/SCHROTTPROGRAMM

LOCK und UNLOCK (LOCK /n, Ss, Dd)

LOCK BRIEF1
UNLOCK BRIEF2

Strich-Befehl (– /n, Ss, Dd)

– BASIC.SYSTEM
– /VOLUME/PRODOS
– FILER, S6, D2

RUN und CHAIN (RUN /n, §z, Ss, Dd)

RUN DEMO
RUN PROGRAMM, §200
CHAIN MODUL.2

EXEC (EXEC /n, Ff oder Rr, Ss, Dd)

EXEC EXECUTIVE.FILE, R2 (ab Feld 2)

LOAD und SAVE (LOAD /n, Ss, Dd)

LOAD BASIC.PROGRAMM
SAVE TESTPROGRAMM

STORE und RESTORE (STORE /n, Ss, Dd)

STORE VARIABLEN
RESTORE ALLE.WERTE

BRUN (BRUN /n, Aa, Bb, Ll oder Ee, Ss, Dd)

BRUN MACHINE.PROGRAM

BSAVE (BSAVE /n, Aa, Ll oder Ee, Tt, Ss, Dd)

BSAVE BILD, A$4000, E$5FFF
BSAVE TEXT, A$1000, L$2000, TTXT (neu)

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

BLOAD FILER, A$2000, TSYS
BLOAD BASIC.PROGRAMM, A$801, TBAS (neu)
BLOAD TEXTFILE, A$2000, TTXT (neu)

OPEN (OPEN, /n, Ll, Tt, Ss, Dd)

100 PRINT CHR$ (4) ,,OPEN RANDOM, L200"
100 PRINT CHR$ (4) ,,OPEN BINAERFILE, TBIN" (neu)

Ein anderer Dateityp außer TXT setzt voraus, daß zuvor die Datei mit CREATE,. BSAVE usw. erzeugt wurde.

READ, WRITE (READ /n, Rr, Ff, Bb)

100 PRINT CHR$ (4) ,,READ TEXTFILE"
100 PRINT CHR$ (4) ,,WRITE RANDOMFILE,R100"

APPEND (APPEND /n, Ll, Ss, Dd)

100 PRINT CHR$ (4) ,,APPEND GROSSDATEI"

POSITION (POSITION /n, Ff oder Rr)

100 PRINT CHR$ (4) ,,POSITION, TEXTFILE, R5"

CLOSE und FLUSH (CLOSE /n)

CLOSE (schließt alle Textfiles)
FLUSH DATEI1

PR#S und IN#S (PR#S, Aa)

PR#1
PR# A$300 (neu: Output-Vektor wird nach $0300 gelegt)

Die Befehle OPEN, READ, WRITE, APPEND und POSITION sind reine Indirektbefehle (RUN-Modus). Alle anderen Befehle sind sowohl Direkt- wie Indirektbefehle.

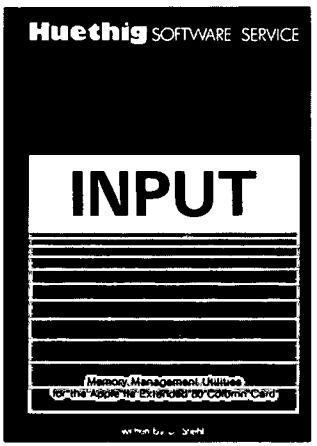
Register

Inhalt der Begleitdiskette zu „ProDOS für Aufsteiger, Band 1“.

AUFSTEIGER.BD1

NAME	TYPE	BLOCKS	MODIFIED	CREATED	ENDFILE	SUBTYPE
VON.U.STIEHL84	BAS	1	25-MAY-84 0:00	<NO DATE>	9	
SPEEDTEST1.S	BIN	4	<NO DATE>	<NO DATE>	1072	A=$0901
SPEEDTEST1	BIN	1	<NO DATE>	<NO DATE>	64	A=$6000
SPEEDTEST2.S	BIN	3	<NO DATE>	<NO DATE>	574	A=$0901
SPEEDTEST2	BIN	1	<NO DATE>	<NO DATE>	52	A=$6000
ZEROTEST.S	BIN	3	<NO DATE>	<NO DATE>	886	A=$0901
ZEROTEST	BIN	1	<NO DATE>	<NO DATE>	106	A=$6000
BEFEHLE.S	BIN	5	<NO DATE>	<NO DATE>	2015	A=$0901
BEFEHLE	BIN	1	<NO DATE>	<NO DATE>	103	A=$0300
OUTPUTVEKTOR.S	BIN	4	<NO DATE>	<NO DATE>	1327	A=$0901
OUTPUTVEKTOR	BIN	1	<NO DATE>	<NO DATE>	91	A=$0300
GLOBAL.PAGE.S	BIN	22	<NO DATE>	<NO DATE>	10676	A=$0901
GLOBAL.PAGE	BIN	1	<NO DATE>	<NO DATE>	256	A=$BF00
RAMDISKOFF.S	BIN	1	<NO DATE>	<NO DATE>	418	A=$0901
RAMDISKOFF	BIN	1	<NO DATE>	<NO DATE>	51	A=$6000
DATUM.S	BIN	6	<NO DATE>	<NO DATE>	2159	A=$0901
DATUM	BIN	1	<NO DATE>	<NO DATE>	267	A=$6000
BLOCKREADER.S	BIN	7	<NO DATE>	<NO DATE>	2577	A=$0901
BLOCKREADER	BIN	1	<NO DATE>	<NO DATE>	319	A=$0803
PRODOS.SKEW.S	BIN	4	<NO DATE>	<NO DATE>	1412	A=$0901
PRODOS.SKEW	BIN	1	<NO DATE>	<NO DATE>	283	A=$6000
PRODOS.DOS3.3.S	BIN	10	<NO DATE>	<NO DATE>	4478	A=$0901
PRODOS.DOS3.3	BIN	3	<NO DATE>	<NO DATE>	639	A=$3000
READADRS.S	BIN	4	<NO DATE>	<NO DATE>	1260	A=$0901
READADRS	BIN	1	<NO DATE>	<NO DATE>	125	A=$1000
EDITOR.S	BIN	3	<NO DATE>	<NO DATE>	756	A=$0901
EDITOR	BIN	1	<NO DATE>	<NO DATE>	58	A=$3FA5
RDWRBLOCK.S	BIN	3	<NO DATE>	<NO DATE>	747	A=$0901
RDWRBLOCK	BIN	1	<NO DATE>	<NO DATE>	36	A=$6000
ALLOC.DEALLOC.S	BIN	4	<NO DATE>	<NO DATE>	1073	A=$0901
ALLOC.DEALLOC	BIN	1	<NO DATE>	<NO DATE>	82	A=$6000
REBOOT.TIME.S	BIN	3	<NO DATE>	<NO DATE>	833	A=$0901
REBOOT.TIME	BIN	1	<NO DATE>	<NO DATE>	14	A=$6000
CREATE.DESTR.S	BIN	3	<NO DATE>	<NO DATE>	725	A=$0901
CREATE.DESTR	BIN	1	<NO DATE>	<NO DATE>	63	A=$6000
RENAME.S	BIN	1	<NO DATE>	<NO DATE>	456	A=$0901
RENAME	BIN	1	<NO DATE>	<NO DATE>	38	A=$6000
FILEINFO.S	BIN	4	<NO DATE>	<NO DATE>	1257	A=$0901
FILEINFO	BIN	1	<NO DATE>	<NO DATE>	96	A=$6000
ONLINE.PREFIX.S	BIN	6	<NO DATE>	<NO DATE>	2061	A=$0901
ONLINE.PREFIX	BIN	3	<NO DATE>	<NO DATE>	524	A=$6000
SETPREFIX.S	BIN	4	<NO DATE>	<NO DATE>	1187	A=$0901
SETPREFIX	BIN	1	<NO DATE>	<NO DATE>	252	A=$6000
WRITE.DEMO.S	BIN	6	<NO DATE>	<NO DATE>	2547	A=$0901
WRITE.DEMO	BIN	1	<NO DATE>	<NO DATE>	189	A=$6000
PRODOS.INIT	BIN	9	<NO DATE>	<NO DATE>	3840	A=$8000
T.SPEEDTEST1	TXT	4	<NO DATE>	<NO DATE>	1072	R= 0
T.SPEEDTEST2	TXT	3	<NO DATE>	<NO DATE>	574	R= 0
T.ZEROTEST	TXT	3	<NO DATE>	<NO DATE>	886	R= 0
T.BEFEHLE	TXT	5	<NO DATE>	<NO DATE>	2015	R= 0
T.OUTPUTVEKTOR	TXT	4	<NO DATE>	<NO DATE>	1327	R= 0
T.GLOBAL.PAGE	TXT	22	<NO DATE>	<NO DATE>	10676	R= 0
T.RAMDISKOFF	TXT	1	<NO DATE>	<NO DATE>	418	R= 0
T.DATUM	TXT	6	<NO DATE>	<NO DATE>	2159	R= 0
T.BLOCKREADER	TXT	7	<NO DATE>	<NO DATE>	2577	R= 0
T.PRODOS.SKEW	TXT	4	<NO DATE>	<NO DATE>	1412	R= 0
T.PRODOS.DOS3.3	TXT	10	<NO DATE>	<NO DATE>	4478	R= 0
T.READADRS	TXT	4	<NO DATE>	<NO DATE>	1260	R= 0
T.EDITOR	TXT	3	<NO DATE>	<NO DATE>	756	R= 0
T.RDWRBLOCK	TXT	3	<NO DATE>	<NO DATE>	747	R= 0
T.ALLOC.DEALLOC	TXT	4	<NO DATE>	<NO DATE>	1073	R= 0
T.REBOOT.TIME	TXT	3	<NO DATE>	<NO DATE>	833	R= 0
T.CREATE.DESTR	TXT	3	<NO DATE>	<NO DATE>	725	R= 0
T.RENAME	TXT	1	<NO DATE>	<NO DATE>	456	R= 0
T.FILEINFO	TXT	4	<NO DATE>	<NO DATE>	1257	R= 0
T.ONLINE.PREFIX	TXT	6	<NO DATE>	<NO DATE>	2061	R= 0
T.SETPREFIX	TXT	4	<NO DATE>	<NO DATE>	1187	R= 0
T.WRITE.DEMO	TXT	6	<NO DATE>	<NO DATE>	2547	R= 0

BLOCKS FREE: 10 BLOCKS USED: 270 TOTAL BLOCKS: 280



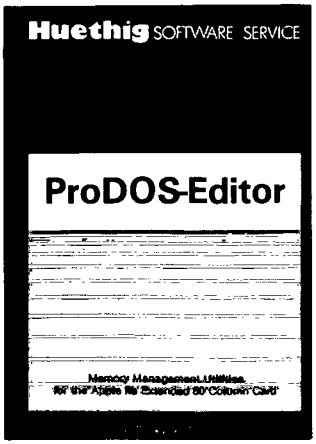
INPUT 2.0
Ein Bildschirm-Maskengenerator für DOS 3.3 und ProDOS

von U. Stiehl

1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1021-5

Gerätevoraussetzung:
Apple IIc/IIe oder
Apple II (im letzteren Fall nur 40 Z/Z)

Der für den Apple II bestimmte Maskengenerator „Input 2.0" basiert auf den früheren Programmen „Input 1.0" und „Input80 1.0" (von denen noch Restbestände lieferbar sind) und ist sowohl unter DOS 3.3 wie auch unter dem neuen ProDOS lauffähig. Der Maskengenerator setzt einen Apple II Plus mit Language Card oder einen Apple IIe voraus. Im 40 Z/Z-Modus funktioniert er auf beiden Gerätetypen, im 80 Z/Z-Modus dagegen nur auf dem Apple IIe mit 80-Zeichen-Karte. (Die alte Videx-Karte für den Apple II wird nicht unterstützt!)
„Input 2.0" liegt wahlweise in der Bank 1 oder Bank 2 der Language Card und wird durch einen kurzen Driver in den unteren 48K aufgerufen. „Input 2.0" läßt sich problemlos in nicht-compilierte und compilierte Applesoft- sowie in Assemblerprogramme einbinden. Die Übergabe der Feldinhalte an das Anwenderprogramm erfolgt durch ein einfaches Verfahren, das auch bei Compilern funktioniert.
Für jedes Feld der Bildschirmmaske lassen sich u. a. definieren: Feldlänge (bis zu 255 Zeichen) – Vtab – Htab – Datentyp (insgesamt 8 Typen) – Scrollflag (starre oder dynamische Maske) – Ctrlflag – Füllflag – Löschflag – Bildschirmflag (40- oder 80-Z-Darstellung). Innerhalb eines Eingabefeldes besteht jeder denkbare Redigierkomfort (Insert, Delete, Rubout, Restore usw.).
Bei der neuen Version des Maskengenerators können jetzt auch Ctrl-Zeichen beim Datentyp String eingegeben werden. Ferner sind – dies gilt nur für IIe – die Apfeltasten als schnelle Cursortasten definiert. Schließlich wurden Features implementiert, die den Einsatz von „Input 2.0" als zeilenorientiertes Textverarbeitungsprogramm ermöglichen. Die „Input 2.0"-Diskette enthält zahlreiche Demos zur Veranschaulichung der Anwendung.



ProDOS-Editor 1.0

Applesoft-Editor
unter ProDOS-Betriebssystem

von U. Stiehl

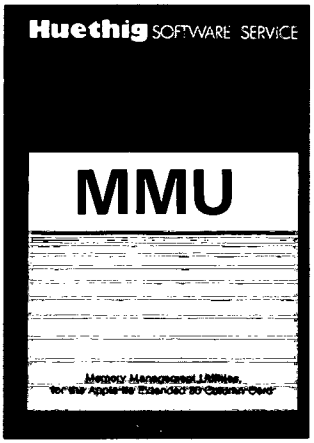
1984, Diskette und Manual, DM 98,–
ISBN 3-7785-1024-X

Gerätevoraussetzung:
Apple IIc/IIe/II+

Mit diesem neuen Editor – übrigens der bislang einzige deutsche ProDOS-Editor – wird dem *Applesoft-Programmierer* ein Werkzeug zur effektiven Programmierung unter dem Betriebssystem ProDOS gegeben, denn die früheren Editoren sind allesamt unter ProDOS nicht mehr lauffähig.
Unter anderen sind folgende Features implementiert worden:

- Zeilenorientierter Editor mit jedem erdenklichen Redigierkomfort (Insert, Delete, Tab, Restore, freie Cursorbewegung in allen vier Richtungen, Eingabe von Ctrl-Buchstaben in Applesoft-Zeilen usw.)
- Renumber (Zeilen-Umnumerierung)
- Xreference (sortierte Variablenliste)
- Suchen von Tokens, Strings und Variablen
- dezimale und hexadezimale Umrechnung
- Ausführung von Monitorbefehlen aus dem Editor heraus
- Listen des Applesoft-Programms in speicherinterner Form als Hex-Dump
- Suchen von Hex-Folgen, Adressen oder Speicherstellen im gesamten RAM-Bereich einschließlich der Language Card
- frei definierbare Tastatur-Macrobefehle

Der Applesoft-Editor liegt in einem von ProDOS geschützten Bereich und läßt sich per Tastendruck vorübergehend abschalten und ebenso einfach wieder aktivieren.



MMU 2.0
Memory Managements Utilities

für die Apple IIe 64K-Karte

DOS 3.3 (und ProDOS)

von U. Stiehl

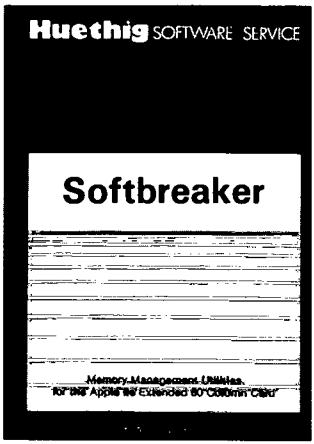
1984, Diskette und Manual, DM 98,–
ISBN 3-7787-1023-1

Gerätevoraussetzung:
Apple IIc/IIe

Bei der Version 2.0 der MMU's sind die Utilities teilweise so umgeschrieben worden, daß sie sowohl unter DOS 3.3 als auch unter ProDOS lauffähig sind. Da dies nicht immer möglich war, sind zusätzlich zu den reinen DOS-Hilfsprogrammen, speziell den RAM-Disk-Drivern, einige reine ProDOS-Utilities aufgenommen worden. Insgesamt enthält die neue „MMU 2.0"-Diskette über 25 Programme, die neue Einsatzmöglichkeiten für die Extended 80 Column Card (erweiterte 80-Z-Karte = 64K-Karte für den Apple IIe) erschließen. Ein Teil der Programme laufen auch auf dem Apple II Plus, doch ist „MMU 2.0" primär für 64K-Karte-Besitzer gedacht.
Im einzelnen umfaßt „MMU 2.0"

- Drei RAM-Disk-Driver für DOS 3.3: „INIT62" benutzt nur die 64K-Karte als RAM-Disk, „INIT78" benutzt zusätzlich die Motherboard-LC als RAM-Karte und „DOSMOVER. INIT62" gilt für den Fall, daß sich das DOS selbst in der Motherboard-LC befindet.
- Eine sehr nützliche Pseudo-Coprocessor-Utility, die das Hin- und Herschalten zwischen zwei Programm-Modulen ermöglicht, von denen sich das eine Modul auf der 64K-Karte befindet.
- Zwei schnelle Kopierprogramme (für DOS 3.3 und ProDOS).
- Mehrere Move-Programme zum Verschieben von Daten auf die 64K-Karte sowie auf die Language Card und umgekehrt.
  Mehrere Hilfsprogramme zum Untersuchen und Löschen bestimmter Speicherbereiche der 64K-Karte und der LC, zur Ermittlung des Softswitch-Status usw.
- Zwei Simulator-Programme zum Simulieren von Apple II und Apple II Plus auf dem Apple IIe.

Schließlich enthält „MMU 2.0" auch verschiedene Move-Utilities für die RAM-Karten AP20 und AP17 der Firma IBS.



Softbreaker 1.0

Eine softwaremäßige Interrupt-Utility für die Apple IIe 64K-Karte

von U. Stiehl

1984, Diskette und Manual, DM 48,–
ISBN 3-7785-1022-3

Gerätevoraussetzung:
Apple IIe (IIc-Version gesondert anfragen)

„Softbreaker" ist ein Assemblerprogramm, mit dessen Hilfe Programme, die sich von der 64K-Karte (= Extended 80 Column Card für den Apple IIe) starten lassen, unterbrochen, gespeichert, geladen und exakt an der Stelle der Unterbrechung fortgeführt werden können. Dadurch ist es auch möglich, Sicherungskopien von sogenannten kopiergeschützten Programmen herzustellen. Es sei hier ausdrücklich darauf hingewiesen, daß laut Urheberrechtsgesetz die Herstellung von Vervielfältigungsstücken nur von rechtmäßig erworbenen Werken und nur zum persönlichen Gebrauch zulässig ist.

„Softbreaker" läßt sich bei folgenden Programmtypen anwenden:

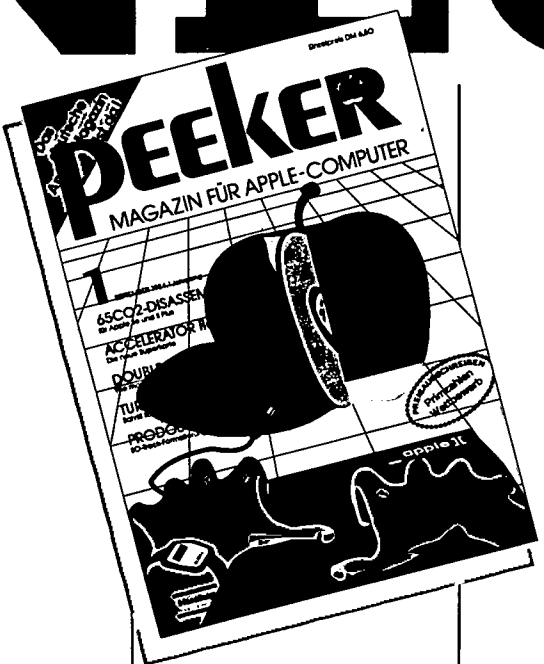
- bei grundsätzlich allen für den Apple II Plus konzipierten Programmen, die die 40-Zeichen-Darstellung und/oder die Grafikseite 1 (also nicht HGR2) benutzen, gleichviel ob sie durch Reset abgesichert sind oder nicht.
- bei grundsätzlich allen durch Reset abgesicherten Programmen, auch wenn sie die 80-Zeichen-Darstellung benutzen.

Softbreaker funktioniert selbstverständlich nicht bei Programmen, die die 64K-Karte des Apple IIe in irgendeiner Form benutzen, z. B. nicht bei Programmen, die die 64K-Karte als RAM-Disk oder als Zusatzspeicher verwenden.

Mit Softbreaker unterbrochene Programme werden komplett, d. h. die ganzen 64K einschließlich Language Card, in nur ca. 11 Sekunden auf einer formatierten Diskette gesichert, wobei die Rücksprungadresse und diverse Status-Werte automatisch mit abgespeichert werden, so daß keinerlei technischen oder Programmierkenntnisse erforderlich sind.

DAS DEUTSCHE APPLE-MAGAZIN IST DA!

NEU



»**peeker**« ist die neue deutschsprachige Computer-Zeitschrift, die sich ausschließlich mit Apple-Computern (Apple II, Macintosh usw.) befaßt. Es gibt viele gute Gründe, »**peeker**« zu abonnieren:

»**peeker**« wendet sich an alle Apple-Besitzer, an Neulinge ebenso wie an Erfahrene.

»**peeker**« ist firmenunabhängig, d. h. wir schreiben auch über die Schwächen und wie man sie behebt.

»**peeker**« bietet in jedem Heft zahlreiche, gut dokumentierte Programme, die auch als Begleitdiskette zur Zeitschrift erhältlich sind.

»**peeker**« bringt ausführliche Software-Erfahrungsberichte sowie Tips und Tricks, damit Sie noch mehr aus Ihren Programmen machen können.

»**peeker**« testet für Sie Original-Geräte, Apple-Nachbauten, Laufwerke, Drucker und andere Hardware.

»**peeker**« erscheint seit September 1984 (2 Ausgaben); ab Januar 1985 monatlich. Der Einführungspreis beträgt DM 58,- (Inland) für 12 Ausgaben pro Jahr. Probehefte senden wir Ihnen gerne kostenlos zu. Postkarte genügt!

Hüthig

»peeker« Leserservice
Postfach 10 28 69
6900 Heidelberg 1

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

Applesoft-Programmierer, die unter DOS 3.3 gearbeitet haben, werden sich schnell an ProDOS gewöhnen, da ProDOS und DOS 3.3 in dieser Hinsicht weitgehend kompatibel sind. Dagegen müssen Assembler-Programmierer völlig umdenken. Deshalb liegt das Schwergewicht dieses ersten Bandes auf der Assemblerprogrammierung und der minutiösen Darstellung der ProDOS-internen Systemadressen, die jedoch auch für Applesoft-Programmierer von großer Bedeutung sind.

Zunächst wird ein allgemeiner Überblick über das neue „Professional Disk Operating System" gegeben. Im Anschluß daran folgt eine Gegenüberstellung der Geschwindigkeit des Diskettenzugriffs bei DOS und ProDOS. Dann wird die interne Speicherorganisation detailliert beschrieben (Boot-Vorgang, Zero-Page, ProDOS-Vektoren, Basic-System-Puffer, Basic-System-Global-Page, Basic-Command-Handler, I/O-Vektoren, ProDOS-Global-Page, Language-Card-Organisation, Interrupt, Disk-Driver, Reboot-Programm usw.). Ebenso ausführlich wird die externe Speicherorganisation geschildert (Spuren, Sektoren, Blocks, Directory-Struktur, Volume Bit Map, Dateistrukturen usw.). Schließlich wird das MLI (Machine Language Interface) mit zahlreichen praktischen Anwendungsbeispielen erläutert. Insgesamt enthält dieser Band 70 Seiten mit eigens für dieses Werk entwickelten Programmen.