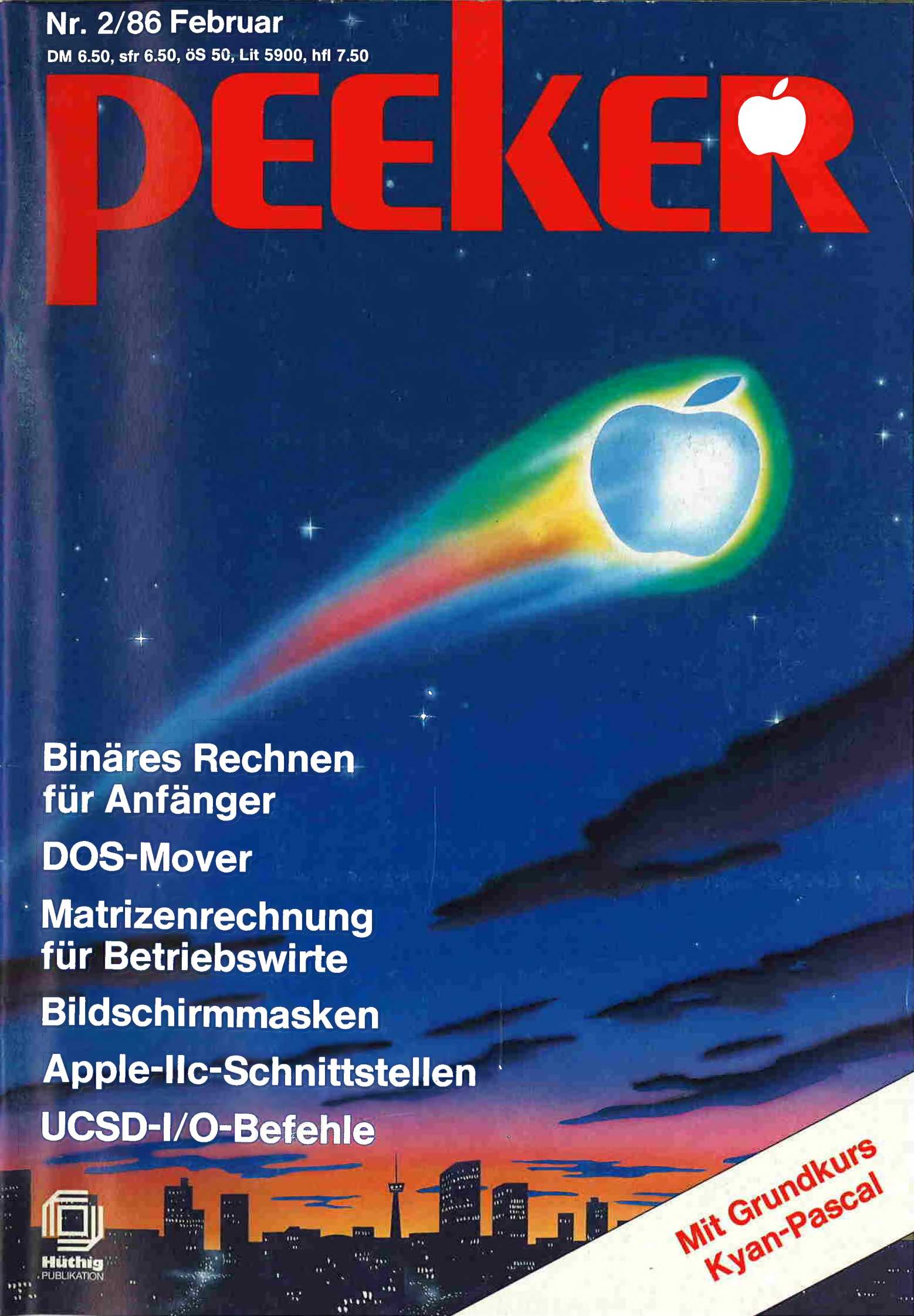
Nr. 2/86 Februar
DM 6.50, sfr 6.50, öS 50, Lit 5900, hfl 7.50

PEEKER



Binäres Rechnen für Anfänger

DOS-Mover

Matrizenrechnung für Betriebswirte

Bildschirmmasken

Apple-IIc-Schnittstellen

UCSD-I/O-Befehle

Mit Grundkurs Kyan-Pascal

Hüthig PUBLIKATION

Computerbücher
die gehen, für Computer die kommen.

Arne Schäpers
ProDOS-Analyse
Versionen 1.0.1, 1.0.2, 1.1.1
1985, 470 S., kart., DM 68,–
ISBN 3-7785-1134-3

Arne Schäpers
Bewegte Apple-Grafik
1985, 305 S., 6 Abb., kart.,
DM 58,—
ISBN 3-7785-1150-5

Frank Bühler
Applesoft Basic
Tips und Tricks
1985, 241 S., 40 Abb., kart.,
DM 38,—
ISBN 3-7785-1094-0

Jürgen Kehrel
Assembler lernen
Band 1: Einführung in die Assembler-Programmierung des 6502
1985, ca. 200 S., kart.,
DM 38,—
ISBN 3-7785-1151-3

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

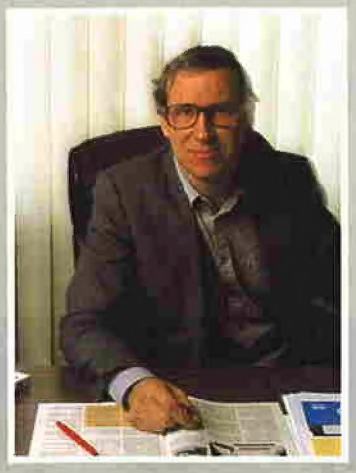
Ulrich Stiehl
Apple DOS 3.3
Tips und Tricks
3., völlig überarbeitete Ausgabe erscheint Anfang 1986

Ulrich Stiehl
ProDOS für Aufsteiger
Band 1
2., geänderte Auflage 1985,
208 S., kart., DM 28,—
ISBN 3-7785-1098-3

Ulrich Stiehl
ProDOS für Aufsteiger
Band 2
1985, 207 S., kart., DM 30,—
ISBN 3-7785-1036-3

Weitere Titel und Informationen finden Sie in unserem Computerbuch-Katalog:
Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1

Hüthig



EDITORIAL

Informatik für jedermann

Mit der Popularisierung der Wissenschaft haben sich die Deutschen immer schwer getan. So begreifen beispielsweise Philosophiestudenten die freie englische Übersetzung des Kantschen Werks besser als das deutsche Original. Ein einfaches Gemüt schlug einmal zufällig die „Reine Vernunft" auf. Nachdem er in diesem Buch eine Zeitlang geblättert hatte, rief er verstört aus: „Deine Sorgen möcht' ich haben!"

Unverständlich und praxisfremd – so stellt sich nicht nur die Philosophie dar. Denn auch um die als kristallklar gerühmte Mathematik ist es nicht zum besten bestellt. Man kann mit Fug und Recht behaupten, daß es kaum einen professionellen Mathematiker gibt, der in der Lage wäre, die Werke von Gauß im Original zu lesen. Nicht weil Gauß in Latein schrieb, wie es damals üblich war, sondern weil er stets die fertige Lösung präsentierte und somit die Spuren seiner mathematischen Gedankengänge sorgfältig verwischte.

Im Peeker wollen wir die Popularisierung der Informatik vorantreiben. Zu ausgewählten Grundsatzthemen werden jetzt regelmäßig didaktisch aufbereitete Beiträge wie etwa der Aufsatz über das „Binäre Rechnen" erscheinen. Hier besteht ein enormer Nachholbedarf. Ich habe mir einmal den Spaß erlaubt, einigen wirklich versierten Assemblerprogrammierern die folgende Frage vorzulegen (Dialog): „Sie kennen doch das Zweierkomplement von Binärzahlen, oder?" – „Logisch!" – „Und Dezimalzahlen kennen Sie auch?" – „Was für eine Frage!" – „Nun, wenn das so ist, wie heißt dann das dezimale Zweierkomplement zur der Dezimalzahl 6789?" – Sendepause! Keiner konnte die Frage ohne Umschweife beantworten, also direkt ohne vorherige Umwandlung von 6789 in eine Binärzahl, Komplementierung und Rückumwandlung in die Dezimalzahl. Dies zeigt aber auch gleichzeitig, daß man über das Einer- und Zweierkomplement nie bewußt nachgedacht hat, sonst wäre der „didaktische Transfer" sofort gelungen. Übrigens: Informatiklehrer, die bereits jetzt den zweiten Teil des Beitrages über „Binäres Rechnen" benötigen, können den Umbruch kostenlos anfordern (bitte mit Schulstempel).

Kyan-Pascal

Soeben haben wir erfahren, daß am 1.2.86 die neue Version 2.0 (bislang Version 1.2) ausgeliefert wird. Wir haben deshalb für diejenigen, die sich an der Sammelbestellaktion beteiligen, die neue Version angefordert, die voraussichtlich am 10.2.86 bei uns eintreffen wird. Sinngemäß haben wir den Anmeldeschluß für die Sammelbestellung um einen Monat bis zum 28.2.86 verlängert. Nach diesem Termin müssen wir den Sonderpreis (DM 170,–) leider erhöhen, da der USA-Bezugspreis inzwischen $ 80.00 beträgt. Bestellen Sie also sofort, denn so preiswert werden Sie Kyan-Pascal nie mehr erhalten.

Kyan-Pascal wird von uns voll unterstützt. Zunächst erscheint ein zweiteiliger Kompaktkurs für Anfänger. Außerdem sind verschiedene Include-Utilities für Hires-Grafik, Stringverarbeitung, Bildschirmmasken und ProDOS-Dateiverwaltung in Arbeit, die im Laufe der nächsten Hefte veröffentlicht werden.

Neuer Macintosh

Die Firma Apple wird Ende Januar in München einen neuen Macintosh vorstellen, der die Bezeichnung „Macintosh Plus" haben wird. Nachdem der alte Macintosh ein „Mauerblümchendasein" (s. „Chip"-Bestseller-Liste, Heft 11/85) führte, hoffen wir, daß das neue Gerät die Forderungen nach einem offenen Systems erfüllt und der Endabnehmerpreis angesichts Atari moderat ausfällt. Wir werden im nächsten Heft darüber berichten.

Ulrich Stiehl

INHALT

Impressum

Peeker
3. Jahrgang 1986
ISSN 0176-9200

Verleger und Herausgeber:
Dipl.-Kfm. Holger Hüthig
Geschäftsführung Zeitschriften:
Heinz Melcher
Chefredakteur: Ulrich Stiehl (us)
Redaktion: Harald Grumser
Sekretariat, Tel. (0 62 21) 48 92 31
Bestellungen, Tel. (0 62 21) 48 92 78
Anzeigen, Tel. (0 62 21) 48 92 18

Vertriebsleitung:
Walter Menzel, Tel. (0 62 21) 48 92 80
Produktionsleitung: Gunter Sokollek
Gestaltung: Rainer Schmitt

Hüthig
PUBLIKATION

PEEKER

Heft 2/1986

Verlag:
Dr. Alfred Hüthig Verlag GmbH
Im Weiher 10, Postfach 10 28 69
6900 Heidelberg
Telefon (0 62 21) 4 89-1
Telex 4-6 17 27 hued d.
Telefax (0 62 21) 489 279
BTX * 51851 #

Erscheinungsweise: 12 Hefte jährlich,
Erscheinungstag jeweils 1 Woche vor Monatsbeginn.
Jahresabonnement DM 72,–, einschließlich MwSt,
im Inland portofrei. Einzelheft DM 6,50
Vertrieb Handel:
MZV – Moderner Zeitschriften Vertrieb GmbH
Breslauer Str. 5, Postfach 1123,
8057 Eching b. München,
Tel. 089/319 10 67, Telex 0 522 656

Zahlungen: an den Dr. Alfred Hüthig Verlag
GmbH, D-6900 Heidelberg 1: Postgiro-
konten: BRD: Karlsruhe 485 45-753;
Österreich: Wien 7555 8 88; Schweiz: Basel
40-24417; Niederlande: Den Haag 1 457 28;
Italien: Mailand 47718; Belgien:
Brüssel 7230 26; Dänemark: Kopenhagen
349 69; Norwegen: Oslo 994 24;
Schweden: Stockholm 5477 76-5

Bankkonten: Landeszentralbank Heidel-
berg 67 207 341; BLZ 672 000 00; Deutsche
Bank Heidelberg 0 2165 041; BLZ
672 700 03; Bezirkssparkasse Heidelberg
204 51, BLZ 672 500 20.

Herstellung: Heidelberger Verlagsanstalt
Printed in Germany

Binäres Rec
mit Papier u

von Ulrich Stiehl

Teil 1: Addition und Subtraktion

Gliederung

1. Einleitung

Das binäre Rechnen mit natürlichen Zahlen ist leicht verständlich, *wenn* es nachvollziehbar erklärt wird. Die üblichen 6502-Lehrbücher von L.A.Leventhal, R.Hyde u.a. beschränken sich indes allesamt auf beiläufige Sätze in der Art „You can perform division on the computer just like you would perform division with pen and paper" und gehen dann sofort zur Tagesordnung über, indem sie Algorithmen präsentieren, die mit „paper and pencil" nur noch wenig gemein haben. Nirgendwo wird das binäre Rechnen einmal wirklich Schritt für Schritt auf das dezimale Rechnen mit Papier und Bleistift zurückgeführt. Folglich bleibt eine Lücke in der Beweisführung, und es ist von daher einsichtig, daß eine nicht geringe Zahl von Assemblerprogrammierern die binären Algorithmen „blind" anwendet.
Ein 6502-Profi wird möglicherweise jetzt naserümpfend ausrufen: „Ich gehöre nicht zu jenen ‚Blind'-Gängern!" Wirklich nicht? Dann prüfen Sie sich jetzt einmal selbst. Sie kennen das *binäre* Einer- und Zweierkomplement, z.B.
1001 → 0110 Einerkomplement
1001 → 0111 Zweierkomplement
Frage: Wie heißt das *dezimale* Zweierkomplement zu der Zahl 6789? Wenn Sie die Lösung spontan und ohne langes Knobeln niederschreiben können, dann sind Sie nicht so blind wie wir und brauchen den nachfolgenden Aufsatz nicht mehr zu lesen.

Im folgenden beschränken wir uns auf das schriftliche Rechnen (Addition, Subtraktion, Multiplikation und Division) mit **natürlichen Zahlen** (0, 1, 2, 3 usw.) im Dezimal- und Binärsystem. Somit werden negative Zahlen (-1, -2, -3 usw.) und Bruchzahlen (1/1, 1/2, 1/3 usw.) bzw. Dezimalzahlen im engeren Sinne (1.0, 0.5, 0.33333 usw.) ausgespart. Das schriftliche Rechnen im Dezimalsystem wird zwar als *praktische Fähigkeit* vorausgesetzt, doch werden wir zudem auf den zugrundeliegenden *theoretischen Algorithmus* eingehen, denn nur wenn man sich die Vorgehensweise beim schriftlichen Rechnen im Dezimalsystem *bewußt* macht, wird man die Rechenregeln auf das Binärsystem übertragen können.
Bei den einzelnen Abschnitten wird säuberlich zwischen Binärtheorie und 6502-Implementierung getrennt, so daß Nicht-Assemblerprogrammierer die 6502-Passagen überspringen können.

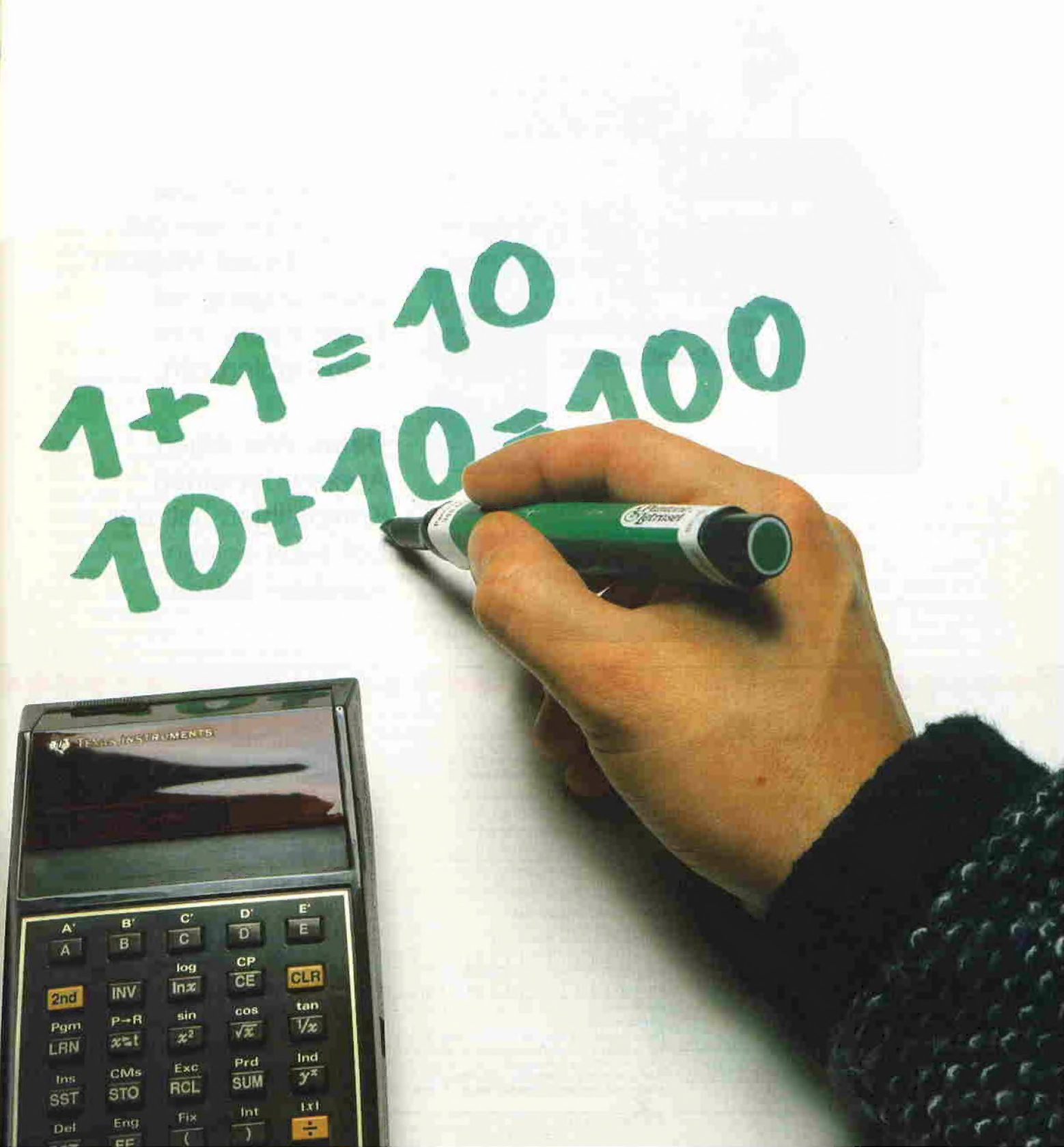
2. Zahlensysteme

Für jede Zahl könnte man theoretisch ein eigenes Zeichen erfinden, z.B. „●" usw. Da es jedoch unendlich viele Zahlen gibt, müßte man sich unendlich viele Zeichen ausdenken. Manche Naturvölker haben sich deshalb auf einige wenige Zahlen beschränkt. Demgegenüber haben Kulturvölker die **Ziffer** eingeführt, die sich als nicht-zusammengesetzte, „einziffrige" Zahl mit eigenem Zahlzeichen definieren läßt, z.B. „●". Zusammengesetzte, „mehrziffrige" Zahlen werden dann durch Verkettung der vorhandenen Ziffern gebildet, z.B. „●●●". Wenn sich der *Wert* einer mehrziffrigen Zahl aus der Summe der Ziffernwerte ergibt, spricht man vom Additionssystem. Wenn hingegen der Wert einer mehrziffrigen Zahl zusätzlich von der Stelle oder Position der Ziffern innerhalb der Ziffernkette abhängt, spricht man vom Positionssystem.

2.1. Additionssysteme

Die römische Zahl
XIII = 13
ist ein Beispiel für ein **Additionssystem**, denn es gilt
X + I + I + I = 13.
(M = 1000, D = 500, C = 100, L = 50, X = 10, V = 5, I = 1.)
Allerdings enthält das römische Zahlensystem bereits Elemente des Positionssy-

nnen
nd Bleistift



Ein Exemplar der hier gezeigten Bücher

Sie haben die Wahl!

erhalten Sie als "Dankeschön" für einen neuen Abonnenten, den Sie uns vermitteln.

Als »**peeker**«-Leser wissen Sie, wie gut Ihnen dieses Magazin beim Umgang mit Ihrem Apple oder Kompatiblen hilft.

Denn: Wer einen Apple oder einen Kompatiblen hat, der soll auch seinen »**peeker**« haben.

Bestellcoupon

Ich habe den neuen Abonnenten geworben und erhalte kostenlos eines der folgenden Bücher (bitte ankreuzen)

- ☐ Bühler, **Applesoft Basic**
- ☐ Meinhold, **Was ist Elektronik**
- ☐ Schilling, **EDV–kein Geheimnis**
- ☐ Stiehl, **Apple ProDOS Bd. 2**
- ☐ Meinhold, **Was ist Mikroelektronik**

Name, Vorname

Straße, Postfach

PLZ, Ort

X
Datum, Unterschrift

Ich bin der neue Abonnent. Bitte liefern Sie mir bis auf Widerruf, zumindest aber für 1 Jahr, »peeker« zum Jahresbezugspreis von DM 72,– (Ausland plus DM 18,– Porto) an folgende Anschrift:

Name, Vorname

Straße, Postfach

PLZ, Ort

X
Datum, Unterschrift

Gewünschte Zahlungsweise
☐ gegen Rechnung
☐ bargeldlos durch Bankeinzug

Konto-Nr. | Bankleitzahl

Geldinstitut

Vertrauensgarantie:
Diese Bestellung kann ich innerhalb einer Woche bei Dr. Alfred Hüthig Verlag GmbH, Im Weiher 10, 6900 Heidelberg 1 widerrufen. Zur Wahrung der Frist genügt die rechtzeitige Absendung. Ich bestätige die Kenntnisnahme mit meiner Unterschrift:

X
2. Unterschrift

Coupon ausschneiden oder kopieren und einsenden an:

»peeker«
Abonnementservice
Im Weiher 10
6900 Heidelberg 1

Hüthig
PUBLIKATION

stems. So hängt bei der Zahl
MCMLXXXVI = 1986
der Wert „CM“ = 900 von der Position oder Stellung der Ziffer „C“ (= 100) *vor* der Ziffer „M“ (= 1000) ab, denn es gilt
CM = -100 + 1000 = 900
MC = 1000 + 100 = 1100
Ähnlich ist es mit dem französischen Zahlwort
quatre-vingts = 80, d.h. 4 · 20
gegenüber dem Zahlwort
vingt-quatre = 24, d.h. 20 + 4.

Wie ersichtlich, gibt es bei älteren Zahlensystemen etliche Sonderfälle und Ungereimtheiten.

2.2. Positionssysteme

Beim reinen **Positionssystem** wird der Wert einer Zahl konsequent durch die Position der Ziffern bestimmt. Positionssysteme werden nach der Anzahl der verschiedenen Ziffern (= Zahlzeichen) benannt.

2.2.1. Dezimalsystem

Das uns vertraute **Dezimalsystem** verfügt über 10 (lateinisch decem) verschiedene Ziffern
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
und geht ursprünglich auf die Inder zurück. Wie die **Abbildung 1** zeigt, wurden die (noch heute in Indien benutzten) indischen Zahlzeichen von den Europäern nicht konsequent übernommen. Beispielsweise entspricht das Zahlzeichen der indischen „4“ dem Zahlzeichen der europäischen „8“; ferner entspricht die indische „5“ der europäischen „4“, die indische „7“ der europäischen „6“, und die europäische „9“ ist spiegelbildlich verdreht.

Abbildung 1: Indische Zahlzeichen, wie sie heute in der Devanagari-Schrift benutzt werden (Hindi u. a.).

Da das Dezimalsystem das uns Europäern zunächst einzig vertraute Zahlensystem ist, verwundert es nicht, daß für den Begriff „Zahl im Dezimalsystem“ kein eigenes Fachwort geprägt wurde. Wir sprechen zwar von Binärzahl (= „Zahl im Binärsystem“) und Hexadezimalzahl (= „Zahl im Hexadezimalsystem“), aber der Terminus „Dezimalzahl“ oder „dekadische Zahl“ bedeutet normalerweise nicht „Zahl im Dezimalsystem“, denn dafür haben wir bereits den allgemeinen, aber hier mißverständlichen Ausdruck „Zahl“. Wir definieren deshalb **Dezimalzahl** (a) im weiteren Sinne als „Zahl im Dezimalsystem“ und (b) im engeren Sinne als „Zahl im Dezimalsystem mit Dezimalpunkt bzw. -komma“, z.B. 123.456 oder 123,456. In diesem Beitrag werden wir „Dezimalzahl“ stets in dem erweiterten Sinne benutzen.

Bei Positionssystemen entspricht die Anzahl der Zahlzeichen auch gleichzeitig der sog. Basis. Betrachten wir hierzu folgende Aufstellung:

3210 Exponent
4321 Dezimalzahl

Die Dezimalzahl 4321 läßt sich auch als die Summe von (1 · 1) + (10 · 2) + (100 · 3) + (1000 · 4) schreiben, d.h. von rechts nach links repräsentiert die 1. Stelle die Einer (E), die 2. Stelle die Zehner (Z), die 3. Stelle die Hunderter (H) und die 4. Stelle die Tausender (T) usw. Die Zahl 4321 entsteht mithin durch die Addition

```
 0001 E
+0020 Z
+0300 H
+4000 T
------
 4321
```

Wir sagen, daß Dezimalzahlen die **Basis** 10 haben, wobei unter Basis die Grundzahl in der Potenzrechnung gemeint ist (Basis ↑ Exponent = Potenz). Es gilt dann in bezug auf unsere Zahl 4321 von rechts nach links:

```
1 · (10 ↑ 0) =    1 (E)
2 · (10 ↑ 1) =   20 (Z)
3 · (10 ↑ 2) =  300 (H)
4 · (10 ↑ 3) = 4000 (T)
```

1 + 20 + 300 + 4000 = 4321. Die Zehnerpotenzen lauten 1, 10, 100, 1000 usw.

Aus philosophischer Sicht sind die obigen „Beweise“ allerdings ein Hysteron-Proteron, denn man kann das Positionssystem nicht auf die Potenzrechnung zurückführen, weil die Potenzrechnung bereits das Positionssystem voraussetzt. Zur Axiomatisierung des Systems der natürlichen Zahlen wird auf die einschlägigen Werke von G.Peano und Whitehead/Russell verwiesen. Als Kurzdarstellung sei empfohlen „Handbuch philosophischer Grundbegriffe“, München 1974, S.1775 ff. Uns geht es jedoch hier nicht um mathematische Grundlagenforschung, sondern um das praktische („vorphilosophische“) Verständnis.

2.2.2. Binärsystem

Nachdem wir nunmehr das Dezimalsystem an Beispielen erläutert haben, fällt es uns leicht, andere Positionssysteme zu definieren. Im **Binärsystem** gibt es 2 Zahlzeichen – 0 und 1 (oder L) –, und die Basis ist 2. Die Zahlen im Binärsystem oder Dualsystem heißen Binärzahlen oder Dualzahlen („binär“ von lateinisch „bis“ = zweimal; „dual“ von lateinisch „duo“ = „zwei“), und eine einzelne Stelle einer x-stelligen Binärzahl wird als **Bit** bezeichnet (von englisch „*b*inary dig*it*“ = „bit“ = „Binärziffer“). Betrachten wir hierzu die folgende Aufstellung:

3210 Exponent
1111 Binärzahl

Auch hier können wir wieder von rechts nach links eine Addition vornehmen:

```
1 · (2 ↑ 0) = dezimal 1
1 · (2 ↑ 1) = dezimal 2
1 · (2 ↑ 2) = dezimal 4
1 · (2 ↑ 3) = dezimal 8
```

1 + 2 + 4 + 8 = 15. Somit entspricht der Binärzahl 1111 die Dezimalzahl 15. Ein weiteres Übungsbeispiel – ebenfalls von rechts nach links ausgerechnet:

```
1101 = (1 · (2 ↑ 0)) +
       (0 · (2 ↑ 1)) +
       (1 · (2 ↑ 2)) +
       (1 · (2 ↑ 3))
1101 = 1 + 0 + 4 + 8
1101 = dezimal 13
```

Die Zweierpotenzen lauten 1, 2, 4, 8, 16, 32, 64, 128, 256 usw. Wir können deshalb auch verkürzt schreiben: 1101 = (1 · 1) + (0 · 2) + (1 · 4) + (1 · 8) = dezimal 13. Bevor Sie nun weiterlesen, sollten Sie sich eine binär-dezimale Umrechnungstabelle (z.B. aus Peeker 7/85, S. 34) oder die nachstehende Kurztabelle zu Gemüte führen und eine Reihe weiterer Umrechnungen mit Papier und Bleistift vornehmen.

Eine Binärzahl läßt sich also in eine Dezimalzahl umrechnen, indem man von rechts nach links die der Position entsprechende Potenz entweder mit 0 oder mit 1 malnimmt, womit das Ergebnis entweder 0 ist oder der Positionspotenz entspricht. Umgekehrt läßt sich eine Dezimalzahl in eine Binärzahl umwandeln, indem man fortlaufend durch 2 teilt und den jeweiligen Rest (0 oder 1) von oben nach unten oder von rechts nach links notiert. Beispiel:

```
20 : 2 = 10 Rest 0 (0.)
10 : 2 =  5 Rest 0 (1.)
 5 : 2 =  2 Rest 1 (2.)
 2 : 2 =  1 Rest 0 (3.)
 1 : 2 =  0 Rest 1 (4.)
```

43210 Exponent
10100 Binärzahl

2.2.3. Hexadezimalsystem

Neben dem Dezimal- und Binärsystem ist noch das **Hexadezimalsystem** gebräuchlich. Hier gibt es 16 Zahlzeichen – 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F –, und die Basis ist 16. Neben der Bezeichnung „Hexadezimalzahl" oder kurz „Hexzahl" („hex" griechisch „sechs", „decem" lateinisch „zehn") gibt es noch den Terminus „Sedezimalzahl" („sedecim" bzw. „sex-decim" lateinisch „sechzehn"), der sich jedoch nicht durchgesetzt hat. Manche Puristen wollen indessen nur die reinlateinische Bezeichnung „sedezimal" gelten lassen. Es sind dies dieselben Puristen, die vom rein-lateinischen „Ipsomobil" oder vom rein-griechischen „Autokinetikon" sprechen, während der Rest der Menschheit weiterhin beim griechisch-lateinischen „Automobil" oder kurz „Auto" bleibt.

Die nachfolgende Kurztabelle dient zum bequemen Umrechnen zwischen den drei Positionssystemen – binär, dezimal, hexadezimal:

```
0000 = 00 = 0
0001 = 01 = 1
0010 = 02 = 2
0011 = 03 = 3
0100 = 04 = 4
0101 = 05 = 5
0110 = 06 = 6
0111 = 07 = 7
1000 = 08 = 8
1001 = 09 = 9
1010 = 10 = A
1011 = 11 = B
1100 = 12 = C
1101 = 13 = D
1110 = 14 = E
1111 = 15 = F
```

Hexadezimalzahlen wurden in der EDV deshalb eingeführt, weil die Speicherstelle, die ein Mikroprozessor bearbeiten kann, in der Regel das x-fache einer 4stelligen Binärzahl und somit das x-fache einer 1stelligen Hexzahl enthält. Im einzelnen unterscheidet man:

Nibble, z.B. 1010 = A
4stellige Binärzahl = 1stellige Hexzahl

Byte, z.B. 10101010 = AA
8stellige Binärzahl = 2stellige Hexzahl

Wort, z.B. 1010101010101010 = AAAA
16stellige Binärzahl = 4stellige Hexzahl

Da der 6502-Prozessor hinsichtlich der Binärmathematik nur über Additions- und Subtraktionsbefehle verfügt, die exakt jeweils 1 ganzes Byte bearbeiten (= 8stellige Binärzahl = 8 Bits = 2stellige Hexzahl), sind hier die Operanden (Summand, Minuend, Multiplikator, Divisor usw.) einer binären Rechenoperation stets ein x-faches eines Bytes. Beispiele: 8-mal-8-Bit-Multiplikation, 16-durch-8-Bit-Division usw.

3. Addition

3.1. Dezimale Addition

Einstellige Dezimalzahlen können wir „im Kopf" addieren, d.h. wir wissen sofort, daß z.B. 2 + 2 = 4 ist. Mehrstellige Dezimalzahlen werden schriftlich addiert, indem man die beiden Summanden oder Addenden rechtsbündig untereinanderschreibt und von rechts nach links die Einer, Zehner, Hunderter usw. zur Summe zusammenzählt. Beispiel:

```
  1234 Summand
+ 2345 Summand
------------
  3579 Summe
```

Dabei kann ein **Übertrag** von der Einer- zur Zehnerposition, von der Zehner- zur Hunderterposition usw. stattfinden. Beispiel:

```
  1234 Summand
+ 5778 Summand
    11 Übertrag
------------
  7012 Summe
```

4 + 8 ergibt 12 (= 10 + 2). Da jedoch die Einerposition nur 1 Ziffer aufnehmen kann, tragen wir bei der Einerposition 2 ein und übernehmen die 1, die für die 1 von 10 steht, in die Zehnerposition. Danach addieren wir in der Zehnerspalte 3 + 7 + 1 = 11 usw. Dies alles ist Ihnen seit Ihrer Kindheit bereits bestens vertraut.

Es gibt folgende Grundregeln, die sowohl für Dezimal- als auch Binärzahlen gelten:

1. Wenn man 2 Summanden addiert, ist der Übertrag in die nächste Position höchstens 1. Dies gilt nicht, wenn man mehr als 2 Summanden auf einmal addiert.

2. Wenn man 2 x-stellige Summanden addiert, ist das Ergebnis höchstens (x+1)-stellig. Dies gilt ebenfalls nicht, wenn man mehr als 2 Summanden auf einmal addiert. Beispiel für den ungünstigsten Fall:

```
  9999 Summand
+ 9999 Summand
  1111 Übertrag
------------
 19998 Summe
```

3.2. Binäre Addition

Bei der binären Addition müssen wir uns zunächst mit den Rechenregeln vertraut machen. Da wir später bei der binären Multiplikation auch mit mehr als 2 Summanden rechnen werden, geben wir auch deren Additionsregeln an:

```
0 + 0 = 0 Übertrag 0
1 + 0 = 1 Übertrag 0
0 + 1 = 1 Übertrag 0
1 + 1 = 0 Übertrag 1
1 + 1 + 1 = 1 Übertrag 1
1 + 1 + 1 + 1 = 0 Übertrag 1 + 1
1 + 1 + 1 + 1 + 1 = 1 Übertrag 1 + 1
usw.
```

Man präge sich ein, daß im Binärsystem 1 + 1 nicht etwa 2 ergibt, denn die 2 existiert hier gar nicht als Zahlzeichen. 1 + 1 ergibt mithin 10 oder 0 Übertrag 1.

Bitte rechnen Sie nun mit Papier und Bleistift folgende Übungen durch:

```
  0000 Summand (00)
+ 1111 Summand (15)
-----------------
  1111 Summe   (15)

  1010 Summand (10)
+ 0101 Summand (05)
-----------------
  1111 Summe   (15)

  0111 Summand (07)
+ 0011 Summand (03)
   111 Übertrag (1 )
-----------------
  1010 Summe   (10)

  1111 Summand (15)
+ 0111 Summand (07)
  1111 Übertrag (1 )
-----------------
 10110 Summe   (22)

  1111 Summand (15)
+ 1111 Summand (15)
  1111 Übertrag (1 )
-----------------
 11110 Summe   (30)
```

Dieses letzte Beispiel zeigt, daß die Addition von 2 x-stelligen Binärzahlen höchstens zu einem (x+1)-stelligen Ergebnis führt. Dies gilt nicht, wenn 3 oder 4 Binärzahlen auf einmal addiert werden:

```
   1110 Summand (14)
+  1111 Summand (15)
+  1100 Summand (12)
   111  Übertrag (1 )
  111   Übertrag
-----------------
 101001 Summe   (41)

   1111 Summand (15)
+  1111 Summand (15)
+  1111 Summand (15)
+  1111 Summand (15)
   1111 Übertrag (2 )
  1111  Übertrag
 1111   Übertrag
-----------------
 111100 Summe   (60)
```

Wie ersichtlich, muß man bei der Addition von mehr als 2 Summanden mehr als eine Übertragszeile einrichten.

3.3. 6502-Addition

Während wir bei der schriftlichen Addition Binärstelle für Binärstelle, also Bit für Bit, zusammengezählt haben, erfolgt die 6502-Addition aus der Sicht des Programmierers stets in 8-Bit-Schüben, d.h. byteweise. Im einfachsten Fall werden 2 Bytes zusammengezählt.

Der Übertrag bzw. das Übertrag-Bit wird im Status-Register des 6502-Prozessors als sog. **Carry-Flag** (= Übertrag-Speicherstelle) registriert. Das Carry-Flag C kann mit **CLC** (= Clear Carry) auf 0 und mit **SEC** (= Set Carry) auf 1 gesetzt werden. Der Additionsbefehl **ADC** addiert den Inhalt der Speicherstelle M (= Memory) zu dem Inhalt des Akkumulators A und legt das Ergebnis (vorübergehend) im Akkumulator A ab. Unter **Vor-Übertrag** versteht man den Inhalt des Carry-Flags *vor* der Addition und unter **Nach-Übertrag** den Inhalt des Carry-Flags *nach* der Addition. Es gibt bei anderen Prozessoren einen Additionsbefehl, der nur den Nach-Übertrag registriert, der bei der Addition von A und M entsteht, symbolisch
A + M → A + C.

Der 6502-Befehl ADC (= Add *with* Carry) berücksichtigt jedoch nicht nur den Nach-Übertrag, sondern auch stets den Vor-Übertrag, symbolisch
A + M + C → A + C.

Deshalb muß eine Addition immer mit CLC eingeleitet werden. Beispiele:

Vorher:
C = 0
A = 00000001
M = 00000001
Befehl:
ADC M
Nachher:
C = 0
A = 00000010
Kommentar: Hier stört das fehlende CLC nicht, weil C bereits vorher *zufällig* 0 war. Darauf kann man sich jedoch nicht verlassen.

Vorher:
C = 1
A = 00000001
M = 00000001
Befehl:
ADC M
Nachher:
C = 0
A = 00000011
Kommentar: Hier fand eine fehlerhafte Addition statt (1 + 1 = 3!), weil der Vor-Übertrag nicht durch CLC gelöscht wurde (effektiv 1 + 1 + 1 = 3).

Vorher:
C = 1
A = 00000001
M = 00000001
Befehl:
CLC
ADC M
Nachher:
C = 0
A = 00000010
Kommentar: Richtige Addition.

Wenn wir für eine 8-und-8-Bit-Addition als Ergebnis nur 1 Byte reservieren, so können wir einen möglichen **Überlauf** (= Nach-Übertrag 1) durch folgende Befehlssequenz abfangen:

```
CLC
ADC M
BCS FEHLER
BCC ENDE
```

Grund: CLC löscht den Vor-Übertrag. Wenn dann *nach* der Addition von A und M der Nach-Übertrag 1 beträgt, so muß die Summe von A + M größer als 11111111, d.h. größer als dezimal 255 gewesen sein. Im Falle eines Überlaufs (Nach-Übertrag 1) wird der Sprung BCS (Branch on Carry set) ausgeführt, andernfalls der Sprung BCC (Branch on Carry clear).
Ein vollständiges Additionsbeispiel für die 8-Bit-Summanden M1 und M2 sieht im Falle einer 8-Bit-Summe S so aus:

```
CLC
LDA M1
ADC M2
BCS FEHLER
STA S
BCC ENDE
```

Gestattet man eine 16-Bit-Summe SL und SH (Low und High Byte = niederwertiges und höherwertiges Byte), so wird ein nunmehr zulässiger Überlauf im höherwertigen Byte der Summe abgefangen. SH enthält nach der Addition entweder 0 (kein Überlauf) oder 1 (Überlauf). Einen größeren Wert als 1 kann SH niemals enthalten, weil der Nach-Übertrag höchstens 1 ist:

```
LDA #0    ;SH auf
STA SH    ;0 setzen.
CLC       ;Vor-Übertrag = 0
LDA M1    ;1. Summand
ADC M2    ;2. Summand
STA SL    ;SL-Summe
BCC ENDE  ;kein Nach-Übertrag.
INC SH    ;SH auf 1 erhöhen
BCS ENDE  ;wegen Nach-Übertrag.
```

Wenn die Summanden mindestens je 2 Bytes umfassen, so muß man vom niederwertigen (Low) Byte zum höherwertigen (High) Byte – also quasi von rechts nach links – addieren. Nehmen wir an, der 1. Summand belege die Speicherstellen M1L und M1H und der 2. Summand M2L und M2H. Als Summe dieser 2-Byte- oder 16-Bit-Addition sei ein 3-Byte- oder 24-Bit-Ergebnis SL, SM, SH (Low/Middle/High Byte) zugelassen. Die Addition sieht dann wie folgt aus:

```
LDA #0    ;SH auf
STA SH    ;0 setzen.
CLC       ;Vor-Übertrag = 0
LDA M1L   ;Low Bytes
ADC M2L   ;der Summanden.
STA SL    ;SL-Summe
LDA M1H   ;High Bytes
ADC M2H   ;der Summanden.
STA SM    ;SM-Summe
BCC ENDE  ;kein Nach-Übertrag.
INC SH    ;SH auf 1 setzen
BCS ENDE  ;wegen Nach-Übertrag.
```

Betrachten wir diese Addition an einem konkreten Zahlenbeispiel:

```
  High     Low
  11111111 11111111 Summand M1
+ 11111111 11111111 Summand M2
  1        1        0 Übertrag
-------------------
  11111111 11111110 Summe S
```

Da wir die Addition mit CLC einleiten, ist der Vor-Übertrag auf 0 gesetzt. Die Addition der beiden niederwertigen Bytes der Summanden läuft also auf
11111111 + 11111111 + 0
hinaus, wodurch ein Nach-Übertrag von 1 entsteht, der zugleich zum Vor-Übertrag bei der nachfolgenden Addition der beiden höherwertigen Bytes der Summanden wird, so daß für diese gilt:
11111111 + 11111111 + 1
Dies führt zu einem Nach-Übertrag von 1 in das 3. Byte der Summe, die nunmehr effektiv so aussieht:

```
High     Middle   Low
00000001 11111111 11111110
```

Wie dieses ungünstigste Zahlenbeispiel zeigt, sind Vor- und Nach-Übertrag jeweils höchstens 1, so daß problemlos eine beliebig genaue binäre Addition implementiert werden kann.

4. Subtraktion

4.1. Dezimale Subtraktion

Während man bei der Addition die beiden Summanden vertauschen kann, muß man bei der Subtraktion die Reihenfolge beachten, denn 7 - 2 entspricht nicht 2 - 7. Bei der Gleichung
7 - 2 = 5
ist die 7 der Minuend, die 2 der Subtrahend und die 5 die Differenz. Der Minuend ist also die Zahl, von der der Subtrahend abgezogen wird. Während man die Subtraktion einstelliger Dezimalzahlen „im Kopf" erledigt, muß man bei mehrstelligen Zahlen Minuend und Subtrahend rechtsbündig untereinanderschreiben. Danach wird von rechts nach links entweder die

Subtrahendenziffer von der jeweiligen Minuendenziffer abgezogen (= **norddeutsches Wegnehmen**) oder die Differenz zwischen Minuendenziffer und Subtrahendenziffer zur Subtrahendenziffer hinzugezählt (= **süddeutsches Ergänzen**).
Wenn kein **Borgen** (= negativer Übertrag = „Vortrag" = „Unterlauf" = „der Borg") stattfindet, d.h. wenn die Minuendenstellen nicht kleiner als die jeweiligen Subtrahendenstellen sind, sind beide Verfahren gleichwertig. Beispiel ohne „Borgen":

```
 5678 Minuend
-1234 Subtrahend
------------------
 4444 Differenz
```

Norddeutsch: 8 weniger 4 = 4, 7 weniger 3 = 4, 6 weniger 2 = 4 und 5 weniger 1 = 4.
Süddeutsch: 4 bis 8 = 4, 3 bis 7 = 4, 2 bis 6 = 4 und 1 bis 5 = 4.

Norddeutsches Borgen:

```
 lll  Übertrag
 7234 Minuend
-5678 Subtrahend
------------------
 1556 Differenz
```

1. Stelle: 4 - 8 geht nicht, also 14 - 8 = 6, 1 geborgt;
2. Stelle: 3 - 1 = 2, 2 - 7 geht nicht, also 12 - 7 = 5, 1 geborgt;
3. Stelle: 2 - 1 = 1, 1 - 6 geht nicht, also 11 - 6 = 5, 1 geborgt;
4. Stelle: 7 - 1 = 6, 6 - 5 = 1; fertig.
Beim norddeutschen Verfahren wird also der geborgte Übertrag von der *nächsten Minuendenstelle abgezogen.*

Süddeutsches Borgen:

```
 7234 Minuend
-5678 Subtrahend
  lll  Übertrag
------------------
 1556 Differenz
```

1. Stelle: 8 bis 4 geht nicht, also 8 bis 14 = 6, 1 geborgt;
2. Stelle: 7 + 1 = 8, 8 bis 3 geht nicht, also 8 bis 13 = 5, 1 geborgt;
3. Stelle: 6 + 1 = 7, 7 bis 2 geht nicht, also 7 bis 12 = 5, 1 geborgt;
4. Stelle: 5 + 1 = 6, 6 bis 7 = 1; fertig.
Beim süddeutschen Verfahren wird also der geborgte Übertrag zur nächsten *Subtrahendenstelle hinzugezählt.*

4.2. Binäre Subtraktion

Zunächst müssen wir uns mit den Rechenregeln der binären Subtraktion vertraut machen, wobei wir uns auf den einfachsten Fall (1 Minuend minus 1 Subtrahend) beschränken, weil die kombinierte Subtraktion (1 Minuend minus *mehrere* Subtrahenden) „zuviel Kopf nimmt" (Adam Ries) und deshalb hier unnötig verwirren würde.

1. – 3. Regel

```
 0   1   1
-0  -1  -0
---------
 0   0   1
```

Dies sind die drei einfachen Regeln, weil sie zu keinem negativen Übertrag führen.

4. Regel

```
 0
-1
--
 1
```

Hier muß geborgt werden, indem anstelle des effektiven Minuenden 0 der Schein-

Minuend 10 benutzt wird, wobei die „1“ der „10“ von der nächsten Stelle geborgt wird, also so:

```
 10 (2)
- 1 (1)
-------
  1 (1)
```

Je nachdem, ob die nächste Minuendenstelle 1 oder 0 beträgt und ob bei der Subtraktion der momentanen Stelle bereits ein negativer Übertrag der vorangehenden Stelle zu berücksichtigen ist, sind einige Spezialfälle zu unterscheiden, wobei wir uns aus Platzgründen auf das süddeutsche Verfahren (Subtrahendenstelle + Übertrag statt Minuendenstelle - Übertrag) beschränken wollen:

```
 321 1., 2., 3. Stelle
 100 Minuend
-011 Subtrahend
  11 Übertrag
-------------------
 001 Differenz
```

1. Stelle: 1 bis 0 geht nicht, also 1 bis 10 = 1, 1 geborgt;
2. Stelle: 1 + 1 = 10, 10 bis 0 geht nicht, also 10 bis 10 = 0, 1 geborgt;
3. Stelle: 0 + 1 = 1, 1 bis 1 = 0; fertig.

```
 321 1., 2., 3. Stelle
 110 Minuend
-011 Subtrahend
  11 Übertrag
-------------------
 011 Differenz
```

1. Stelle: 1 bis 0 geht nicht, also 1 bis 10 = 1, 1 geborgt;
2. Stelle: 1 + 1 = 10, 10 bis 1 geht nicht, also 10 bis 11 = 1, 1 geborgt;
3. Stelle: 0 + 1 = 1, 1 bis 1 = 0; fertig.

Wie ersichtlich, kann die Auflösung der Minuendenstelle höchstens zu 10 (dezimal 2) oder 11 (dezimal 3) führen („Zwei-Drei-Regel“). Die nach dem süddeutschen Verfahren vorgenommene Zusammenfassung von Subtrahend + Übertrag (= Gesamtsubtrahend) kann 1 (dezimal 1) oder 10 (dezimal 2) ergeben. Daraus lassen sich alle denkbaren Kombinationen ableiten. Versuchen Sie nun, die folgenden Beispiele mit Papier und Bleistift zu lösen.

```
 1101 (13)
-1010 (10)
   1
---------
 0011 (03)

 1100 (12)
-1011 (11)
   11
---------
 0001 (01)

 1000 (08)
-0001 (01)
  111
---------
 0111 (07)
```

Wer sich bei den obigen Aufgaben unterfordert fühlen sollte, probiere einmal die folgende kombinierte Subtraktion. Der Umweg über die dezimale Umrechnung wäre gemogelt!

```
 llllllll Minuend
-00000001 Subtrahend 1
-00000010 Subtrahend 2
-00000101 Subtrahend 3
-00001011 Subtrahend 4
-00010111 Subtrahend 5
-00101111 Subtrahend 6
-01011111 Subtrahend 7
          Übertrag
-------------------------
          Differenz (???)
```

4.3. 6502-Subtraktion

Die 6502-Subtraktion läßt sich als Umkehrung der 6502-Addition begreifen:
– Dem ADC-Additionsbefehl entspricht der Befehl **SBC** in der Subtraktion.
– Vor der Addition muß das Carry-Flag gelöscht (CLC), vor der Subtraktion gesetzt werden (SEC).
– Nach der Addition bedeutet ein ausgeführter BCC, daß kein Additionsüberlauf auftrat, nach der Subtraktion bedeutet ein ausgeführter BCS, daß kein Subtraktionsüberlauf („Unterlauf“) auftrat.
Die 8-minus-8-Bit-Subtraktion sieht gegenüber der entsprechenden Addition mithin so aus:

```
Subtraktion       Addition
-----------------------------
SEC               CLC
LDA MINUEND       LDA SUMMAND1
SBC SUBTRAHEND    ADC SUMMAND2
BCC FEHLER        BCS FEHLER
BCS ENDE          BCC ENDE
```

Eine 16-minus-16-Bit-Subtraktion, bei der ML/MH, SL/SH und DL/DH für die jeweils nieder/höherwertigen Bytes des Minuenden, des Subtrahenden und der Differenz stehen, würde folgendermaßen codiert:

```
SEC
LDA ML      ;Low Byte
SBC SL
STA DL
LDA MH      ;High Byte
SBC SH
STA DH
BCC FEHLER
BCS ENDE
```

Bislang haben wir die Subtraktion einfach als Gegenstück zur Addition mit umgekehrten Vorzeichen betrachtet. Wenn wir jedoch ein konkretes Zahlenbeispiel unter die Lupe nehmen, kommt der „Blackout“:

```
  10000000 00000000 Minuend
- 01111111 11111111 Subtrahend
  11111111 1111111  Übertrag
-----------------------------
  00000000 00000001 Differenz
  High     Low
```

Das obige Beispiel entspricht noch der klassischen süddeutschen Methode. Wir beginnen jedoch die Subtraktion durch SEC mit einem Vor-Übertrag! Danach müßte es „richtig“ heißen:

```
  10000000 00000000 Minuend
- 01111111 11111111 Subtrahend
                  1 Vor-Übertrag
  11111111 1111111  Übertrag
-----------------------------
  00000000 00000000 Differenz (?)
  High     Low
```

Dies kann offenbar nicht mit mit rechten Dingen zugehen, denn 32768 - 32767 ist immer noch 1 und nicht 0! Des Rätsels Lösung ist die sog. Komplement-Addition, denn die SEC-SBC-Befehlssequenz bedeutet eigentlich SEC-ACC („ACC“ = Add Complement with Carry“). Wenn Ihnen dies alles zu verwirrend erscheint, so überspringen Sie am besten den nächsten Abschnitt, da er für die Multiplikation und Division nicht benötigt wird.

4.4. Komplement-Addition

4.4.1. Dezimale Komplement-Addition

Die Subtraktion „nimmt viel Kopf“, wenn mehrere Minuendenstellen hintereinander durch Borgen aufgelöst werden müssen. Deshalb haben findige Mathematiker die Subtraktion auf die sog. komplementierte Addition zurückgeführt. Betrachten wir hierzu die folgenden Ziffernpaare:

0 – 9
1 – 8
2 – 7
3 – 6
4 – 5

Die 9 bezeichnen wir als **Einerkomplement** zur 0, die 8 als Einerkomplement zur 1 usw. Umgekehrt ist auch die 0 das Einerkomplement zur 9, die 1 das Einerkomplement zur 8 usw.
Im Dezimalsystem ist die Basis b = 10. Wenn wir z als *Ziffer* und z' als Komplement-Ziffer definieren, dann gilt:

z' = (b - 1) - z

Beispiel:
b = 10, z = 0, ergo
z' = (10 - 1) - 0
z' = 9 - 0
z' = 9

Das Einerkomplement Z' der *Zahl* Z wird gebildet, indem jede einzelne *Ziffer* z der Zahl Z zu z' komplementiert wird. Beispiele:

12345 Z
87654 Z'

43210 Z
56789 Z'

Das sog. **Zweierkomplement** Z'' zur Zahl Z wird gebildet, indem das Einerkomplement Z' um 1 erhöht wird. Beispiele:

```
 12345 Z
 87654 Z'
+    1
-------
 87655 Z''

 43210 Z
 56789 Z'
+    1
-------
 56790 Z''
```

Man hat nun herausgefunden, daß man die normale Subtraktion zweier x-stelliger Zahlen M (= Minuend) minus S (= Subtrahend), d.h.
M - S,
wenn gilt
M > = S,
auf die Komplement-Addition
M + S''
zurückführen kann, wenn man den sich dabei ergebenden Übertrag Ü in die (x+1)te Stelle – hier in die 5. Stelle – „unter den Tisch fallen läßt". Beispiele:

```
  1234 M      1234 M
- 1234 S    + 8766 S''
       Ü     1111  Ü
-------     -------
  0000        0000

  1234 M      1234 M
- 1122 S    + 8878 S''
       Ü     1111  Ü
-------     -------
  0112        0112
```

Wenn S weniger Stellen als M hat, so muß man S mit führenden Nullen auffüllen:

```
  1234 M      1234 M
- 0022 S    + 9978 S''
       Ü     1111  Ü
-------     -------
  1212        1212
```

Obwohl wir uns in diesem Aufsatz auf die natürlichen, sprich positiven Ganzzahlen beschränken wollen, müssen wir auch den Fall M < S betrachten, da er weiter unten bei der 16-minus-16-Bit-Subtraktion im niederwertigen Byte vorkommen kann. Beispiel:

```
  1234 M      1234 M
- 1235 S    + 8765 S''
  111  Ü           Ü
 1          0     N
-------     -------
  9999        9999
  0001        0001 ''
- 0001       -0001
```

Wenn bei der konventionellen Subtraktion der Nach-Übertrag in die (x+1)te Stelle N = 1 und bei der Komplement-Addition N = 0 ist, so muß von dem Ergebnis das Zweierkomplement gebildet und das Vorzeichen umgekehrt werden.
(Bei der konventionellen Subtraktion könnte man auch b ↑ x, d.h. konkret 10 ↑ 4 = 10000 vom Ergebnis 9999 abziehen, also 9999 - 10000 = -1. In der Praxis schreibt man jedoch nicht 1234 - 1235, sondern 1235 - 1234 = +1, und ändert im nachhinein das Vorzeichen in -1.)

Der Sachverhalt wird deutlicher, wenn wir die Komplement-Addition ziffernweise vornehmen. Betrachten wir hierzu noch einmal den denkbar einfachsten Fall, bei dem bei der konventionellen Subtraktion gar kein Übertrag vorkommt:

```
              321 Stelle
  876 M       876 M
- 123 S     + 877 S''
       Ü      111  Ü
-------     -------
  753         753
```

Die Komplement-Addition läßt sich von rechts nach links, d.h. von der 1. bis zur 3. Stelle, ziffernweise so durchführen:

```
  3     2     1   Stelle
  8     7     6   M-Ziffer
+ 9   + 8   + 7   S''-Ziffer
  1     1     1   Ü
-----  ---   ---
  7     5     3
```

Hier ergab sich für jede Ziffer ein Übertrag von 1. Bei dem nächsten Beispiel erfolgt bei der konventionellen Subtraktion jedoch „zwischendrin" ein Übertrag, der sich gleichwohl nicht als Nach-Übertrag auf die (x+1)te Stelle auswirkt:

```
              123 Stelle
  806 M       806 M
- 123 S     + 877 S''
    1  Ü      1 1  Ü
-------     -------
  683         683
```

Wenn wir die Komplement-Addition wieder ziffernweise von der 1. bis zur 3. Stelle durchführen, so wird der Übertrag in der 2. Stelle gleich 0, und wir dürfen dann in der 3. Stelle nur das *Einer*komplement und nicht mehr das Zweierkomplement bilden:

```
  3     2     1   Stelle
  8     0     6   M-Ziffer
+ 8   + 8   + 7   S''-Ziffer
  1     0     1   Ü
-----  ---   ---
  6     8     3
```

Damit kommen wir endlich dem Wesen der Komplement-Addition auf die Spur:

1. Stelle: Wir *beginnen mit einem Vor-Übertrag von 1*, bilden das *Einer*komplement von der 1. Subtrahendenstelle **3**, d.h. 6, und zählen den Vor-Übertrag 1 hinzu, womit sich das *Zweier*komplement 7 ergibt. Dann addieren wir 6 (1. Minuendenstelle) und 7 (Zweierkomplement der 1. Subtrahendenstelle): 6 + 7 = 3, Übertrag 1. Wir tragen in der 1. Differenzstelle 3 ein und übernehmen den Übertrag 1 als Nach-Übertrag in die 2. Stelle.

2. Stelle: Wir bilden das Einerkomplement der 2. Subtrahendenstelle **2**, d.h. 7, und zählen den Vor-Übertrag 1 hinzu, womit sich das Zweierkomplement 8 ergibt. Dann addieren wir 0 (2. Minuendenstelle) und 8 (Zweierkomplement der 2. Subtrahendenstelle): 0 + 8 = 8, Übertrag 0. Wir tragen in der 2. Differenzstelle 8 ein und übernehmen den Übertrag 0 als Nach-Übertrag in die 3. Stelle.

3. Stelle: Wir bilden das Einerkomplement der 3. Subtrahendenstelle **1**, d.h. 8, und zählen den Vor-Übertrag 0 hinzu, womit sich effektiv das *Einer*komplement 8 ergibt. Dann addieren wir 8 (3. Minuendenstelle) und 8 (*Einer*komplement der 3. Subtrahendenstelle): 8 + 8 = 6, Übertrag 1. Da keine weitere Stelle mehr vorkommt, ist dies der Nach-Übertrag.

Wenn der Nach-Übertrag 1 ist, so können wir ihn unter den Tisch fallen lassen, denn die Differenz ist eine positive Zahl und damit richtig. Wenn der Nach-Übertrag 0 ist, so ist die Differenz eine negative Zahl und damit falsch. Merksatz zur Komplement-Addition:

Mit Vor-Übertrag 1 beginnen und nachher prüfen, ob Nach-Übertrag 1 ist. Dann ist Differenz korrekt.

4.4.2. Binäre Komplement-Addition

Für das Binärsystem gilt ebenfalls die Formel
z' = (b - 1) - z (siehe 4.4.1)
Bei der binären Basis b = 2 ergibt sich für die zwei Ziffern z:

z = 0, ergo
z' = (2 - 1) - 0
z' = 1 - 0
z' = 1
und
z = 1, ergo
z' = (2 - 1) - 1
z' = 1 - 1
z' = 0

Zunächst bringen wir einige Übungsbeispiele zum Zweierkomplement von mehrstelligen Binärzahlen Z:

```
 11110000 Z
 00001111 Z'
+00000001
---------
 00010000 Z''

 01010101 Z
 10101010 Z'
+00000001
---------
 10101011 Z''

 00000000 Z
 11111111 Z'
+00000001
---------
100000000 Z''
```

In diesem letzten Fall führt das Zweierkomplement zu einem Überlauf. Betrachten wir nunmehr das Subtraktionsbeispiel, mit dem wir den Abschnitt 4.3 abgebrochen haben (M = Minuend, S = Subtrahend, Ü = Übertrag):

```
   10000000 00000000 M (32768)
-  01111111 11111111 S (32767)
-------------------------------
   00000000 00000001   (00001)
```

Wir formen den Subtrahenden zum Zweierkomplement um

```
   01111111 11111111 S
   10000000 00000000 S'
   10000000 00000001 S''
```

und führen dann die Zweierkomplement-Addition durch

```
   10000000 00000000 M
+  10000000 00000001 S''
  1                  Ü
-------------------------
   00000000 00000001
```

Wenn wir statt dessen eine Einerkomplement-Addition mit einem Vor-Übertrag von 1 vornehmen, ergibt sich folgende Aufstellung:

```
   10000000 00000000 M
+  10000000 00000000 S'
  1                1 Ü
-------------------------
   00000000 00000001
```

Betrachten wir abschließend den Fall einer 16-minus-16-Bit-Subtraktion, bei der vom niederwertigen zum höherwertigen Byte (a) bei der konventionellen Subtraktion ein Übertrag von 1 und (b) bei der Komplement-Addition ein Übertrag von 0 erfolgt, wobei wir zunächst die konventionelle Subtraktion voranstellen (V = Vor-Übertrag, N = Nach-Übertrag):

```
   11111111 00000000 M ( 65280)
-  00001111 11111111 S ( 04095)
         1        0 V (     0) V=0!
      1111  1111111  Ü (    11 )
  0                  N (0      ) N=0!
-------------------------------------
   11101111 00000001    ( 61185)
```

Als Einerkomplement-Addition mit Vor-Übertrag 1 ergibt sich folgende Aufstellung:

```
   11111111 00000000 M  ( 65280)
+  11110000 00000000 S' ( 95904)
         0        1 V  (      1) V=1!
   111               Ü  ( 11   )
  1                  N  (1     ) N=1!
-------------------------------------
   11101111 00000001     ( 61185)
```

4.4.3. 6502-Komplement-Addition

Wir können nunmehr die 6502-Subtraktion SEC SBC als SEC ACC („Add Complement with Carry") begreifen:

1. Durch SEC wird der Vor-Übertrag auf 1 gesetzt.

2. Wir laden den Akkumulator A mit dem Subtrahenden, „EORen" ihn zum Einerkomplement und zählen die Speicherstelle M (= Memory) hinzu. Da durch SEC das Carry-Flag auf 1 gesetzt wird, entspricht der SBC-Befehl praktisch einer Zweierkomplement-Addition.

3. Wenn nach der SBC- bzw. „ACC"-Operation das Carry-Bit bzw. der Nach-Übertrag 1 ist, wurde die Subtraktion ohne negativen Überlauf ausgeführt.

Formelmäßig lassen sich die drei Schritte so zusammenfassen:
A ← A' + M + 1 oder
A ← A'' + M

Der Befehl
EOR #%11111111
invertiert das Bit-Muster von A und erzeugt damit ein Einerkomplement, z.B.:

10101110 Bit-Muster
11111111 EOR-Maske
01010001 Einerkomplement

Die 6502-Subtraktion
SEC
LDA MINUEND
SBC SUBTRAHEND
BCC FEHLER
BCS ENDE
läßt sich folglich so simulieren:
SEC
LDA SUBTRAHEND
EOR #%11111111
ADC MINUEND
BCC FEHLER
BCS ENDE

Man könnte also auf den Subtraktionsbefehl völlig verzichten. Das nachfolgende Demo, das auf der Peeker-Sammeldisk unter dem Namen **KOMPLEMENT.DEMO** enthalten ist, veranschaulicht dies:

```
10 PRINT CHR$(4)"BLOAD KOMPLE-
   MENT"
20 INPUT "Minuend:";M: POKE 768,M
30 INPUT "Subtrahend: ";S: POKE 769,S
40 CALL 770
50 PRINT M;"-";S;"=";M - S
60 PRINT : GOTO 20
```

Gibt man nach
RUN KOMPLEMENT.DEMO
2 und 1 für Subtrahend und Minuend ein, so wird
01=01
2 - 1 = 1
angezeigt. Gibt man jedoch z.B. 1 und 2 ein, so wird
-FF=-FF
1 - 2 = -1

angezeigt. Dabei steht -FF für die interne, d.h. (noch) nicht komplementierte -1.

```
1           ORG  $0300
2  *
3  * KOMPLEMENT
4  * ==========
5  *
6  MINUEND  HEX  60
7  SUBTRAH  HEX  30
8  *
9  COUT     EQU  $FDED
10 CROUT    EQU  $FD8E
11 PRBYTE   EQU  $FDDA
12 * Normale Subtraktion
13 NORMAL   SEC
14          LDA  MINUEND
15          SBC  SUBTRAH
16          PHA
17          BCS  NORMAL1
18          LDA  #"-"
19          JSR  COUT
20 NORMAL1  PLA
21          JSR  PRBYTE
22          LDA  #"="
23          JSR  COUT
24 * Zweierkomplement-Addition
25 KOMPLEM  CLC
26          LDA  SUBTRAH
27          EOR  #%11111111
28          ADC  #1
29          ADC  MINUEND
30          PHA
31          BCS  KOMPLEM1
32          LDA  #"-"
33          JSR  COUT
34 KOMPLEM1 PLA
35          JSR  PRBYTE
36          JMP  CROUT
```

Der Vollständigkeit halber sei abschließend erwähnt, daß die 6502-Subtraktion in verschiedenen Büchern durch die Formel

A ← A - M - (1 - C)

erklärt und in diesem Zusammenhang C als komplementiertes Borgen interpretiert wird. Wenn C = 1 ist, ist (1 - 1) = 0, und C wird nicht zum Subtrahenden hinzugezählt. Wenn C = 0 ist, ist (1 - 0) = 1, und C wird zum Subtrahenden hinzugezählt. Dies hat indes mit der Papier- und Bleistift-Methode wenig gemein.

Literatur

Klaus D. Sänger: Rechnen aufgefrischt, Buch-und-Zeit-Verlag, Köln 1982. (Behandelt in leichtverständlicher Form die dezimalen Papier- und Bleistift-Rechenverfahren.)

E.Pracht/K.Heidenreich: Elementare Zahlentheorie (UTB). Schöningh-Verlag, Paderborn 1978. (Enthält eine recht detaillierte Darstellung der Rechenverfahren für g-adische Stellenwertsysteme, z.B. Komplement-Addition auf S. 127f. Infolge des permanenten Wechsels zwischen exotischen Zahlensystemen mit den Basen 3, 7, 2, -2 (!) usw. für Nicht-Mathematiker schwer verständlich.)

R.Männel/M.Kohler: Algebra für Wirtschaftsschulen. 15. Aufl., Gehlen-Verlag, Bad Homburg 1980. (Eines der zahlreichen Schulbücher, das sich auch mit dem dualen Rechnen befaßt, hier S. 80ff. Lösungheft mit anfordern!)

A.Osborne/D.Bunnell: An Introduction to Microcomputers. Band 0. 3. Aufl., Osborne/McGraw-Hill-Verlag, Berkeley 1982. (Enthält auf S. 89-132 eine gute Einführung in das binäre Rechnen, allerdings beschränkt auf Addition und Subtraktion.)

Lance A.Leventhal/W.Saville: 6502 Assembly Language Subroutines. Osborne/McGraw-Hill-Verlag, Berkeley 1982. (Enthält auf S. 230-305 alle wichtigen 6502-Algorithmen für binäre Addition, Subtraktion, Multiplikation und Division, allerdings ohne nähere Erläuterungen.)

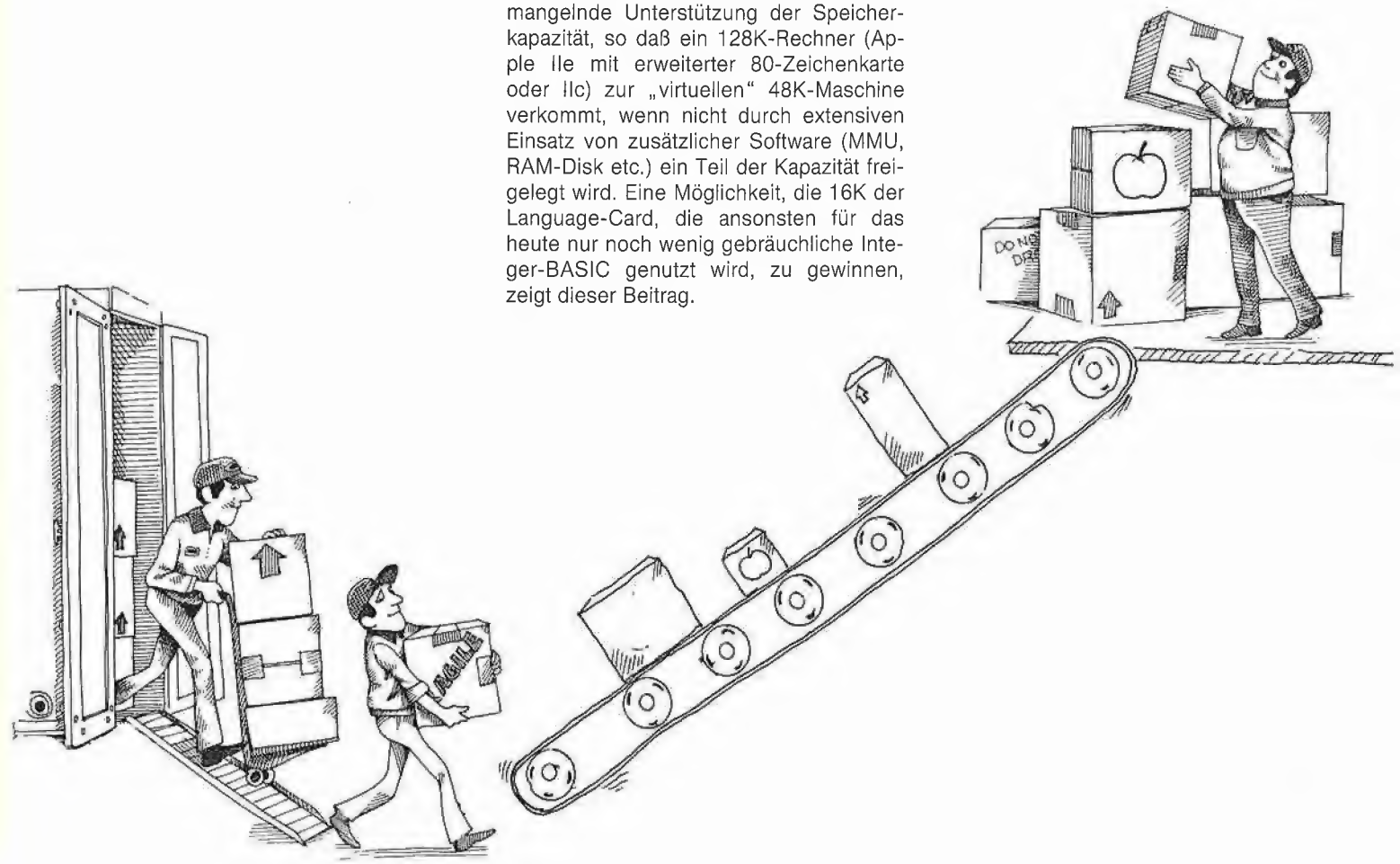
Hinweis: Der zweite Teil, der im Peeker 3/86 erscheinen wird, befaßt sich mit der Multiplikation und Division.

DOS-Mover

Erhöhung der Speicherkapazität um über 30%

von Harald Grumser

Das meistbenutzte Betriebssystem des Apple II, DOS 3.3, dürfte den Höhepunkt seines Erfolgs bereits weit hinter sich gelassen haben. Dennoch wird es trotz der massiven Einführung von ProDOS durch die Firma Apple in absehbarer Zeit nicht zum blassen Eintrag in den Annalen der Computer-Geschichte verkommen.
Einer der gravierendsten Nachteile in der Verbindung Applesoft und DOS 3.3 ist die mangelnde Unterstützung der Speicherkapazität, so daß ein 128K-Rechner (Apple IIe mit erweiterter 80-Zeichenkarte oder IIc) zur „virtuellen" 48K-Maschine verkommt, wenn nicht durch extensiven Einsatz von zusätzlicher Software (MMU, RAM-Disk etc.) ein Teil der Kapazität freigelegt wird. Eine Möglichkeit, die 16K der Language-Card, die ansonsten für das heute nur noch wenig gebräuchliche Integer-BASIC genutzt wird, zu gewinnen, zeigt dieser Beitrag.



1. Die Theorie des Movens

Das Betriebssystem wird beim Booten in den oberen Bereich des verfügbaren Speicherplatzes geladen und schmälert den für den Anwender freien Speicher um über 10K. Bei der Erhöhung der DOS-Puffer (MAXFILES) wird der nutzbare Adreßbereich weiter vermindert. Da auf der anderen Seite die LC in den meisten Fällen ungenutzt bleibt, liegt nahe, das Betriebssystem einschließlich der Puffer in diese 16K zu verlagern. Die Unterteilung der unteren 4K der LC in 2 Bänke bereitet in diesem Zusammenhang keine Schwierigkeiten, da 12K ohnehin für das Betriebssystem und weitere 5 Ein/Ausgabepuffer ausreichen und somit eine Bank für weitere Anwendungen freibleibt.
Diese Übertragung kann sich jedoch nicht auf eine reine Verschiebung des Betriebssystems in einen anderen Speicherbereich beschränken.

1.1. Der Relokator

Fast alle größeren Maschinenprogramme sind nicht an jeder Speicherstelle lauffähig (relokativ), da zum einen Unterprogrammsprünge auf feste Adressen verweisen und zum anderen Datenbereiche und interne Variablen absolut adressiert werden. Wird ein solches Programm einfach verschoben (z.B. durch BRUN FID, A$3000), so verzweigt der Prozessor beim ersten Unterprogrammaufruf nach einer Stelle, die zufällige Werte enthält.
Nun ist es möglich, durch ein Programm alle absoluten Adressen an die neue Lage des Programms anzupassen. Wird also das Programm um $1000 Bytes nach oben geschoben, muß zu allen absoluten Adressen, die eine Adresse innerhalb des Programms ansprechen, $1000 addiert werden (aus JSR $1234 wird JSR $2234). Hierzu muß das gesamte Programm quasi disassembliert werden, was eine Monitor-Routine erledigt, und alle in Frage kommenden Befehle (beim 6502 nur 3-Byte-Befehle) korrigiert werden.
Bei dieser Umsetzung tritt das Problem auf, daß Datenbereiche als Programm interpretiert würden und völlig unerwünschte Umwandlungen zustande kämen. Aus diesem Grund muß dem Relokator eine Tabelle der Datenbereiche zur Verfügung stehen, damit diese Abschnitte unangetastet bleiben.
Ein weiteres Problem bei der Übertragung stellen Adreßtabellen innerhalb des Programms dar. Häufig wird über einen Index aus einer Tabelle die Adresse ermittelt, an der das Programm fortfahren soll (entspricht etwa ON I GOTO... in Applesoft). Auch diese Tabellen müssen dem Relokator bekannt sein, um sie entsprechend anzupassen.
Das bisher Gesagte trifft uneingeschränkt auf jeden „ausgewachsenen" Relokator zu. Der hier eingebaute Relokator berücksichtigt jedoch auch noch einige andere Besonderheiten (s.u.).

1.2. Global-Page, Page 3 und Parameterübergabe

Da DOS häufigen Gebrauch von ROM-Routinen macht und andererseits die Monitor-Routinen DOS aufrufen, muß jeweils vorher eine Umschaltung zwischen RAM (= LC) und ROM stattfinden. Diese Umschaltung kann (abgesehen von exotischen Konstruktionen) nur im ungebankten Bereich erfolgen. Zu diesem Zweck bedient sich das gemovte DOS einer sog. Global-Page (wie auch ProDOS), über die alle Umschaltungen abgewickelt werden.
Aus der Sicht des Betriebssystems bereitet dies keine Schwierigkeiten, da alle Sprünge zu ROM-Routinen von DOS aus vom Relokator auf die Global-Page umgeleitet werden können, wo zunächst das ROM aktiviert wird, die entsprechende Routine ausgeführt und vor dem Rücksprung wieder das RAM selektiert wird.
Um den umgekehrten Aufruf der DOS-Routinen zu erläutern, muß etwas weiter ausgeholt werden: DOS schaltet sich nur bei der Ein- oder Ausgabe in das Geschehen ein. So überwacht das Betriebssystem jedes eingegebene Zeichen und überprüft bei jedem Return, ob die Eingabezeile einen DOS-Befehl enthält, um ihn gegebenenfalls auszuführen und eine leere Eingabezeile an das aufrufende Programm zurückzugeben. Bei der Ausgabe werden alle Zeichen im Eingabepuffer gesammelt und ebenfalls nach einem Return überprüft, ob die ausgegebene Zeile einem DOS-Befehl (angeführt von Ctrl-D) entspricht. Da diese Ein/Ausgabefunktionen über einen Vektor aufgerufen werden, richtet DOS beim Kaltstart diese Programmzeiger auf seine eigenen Ein/Ausgaberoutinen. Der DOS-Mover lenkt diese Vektoren nun auf die Global-Page um, wo wie oben die Umschaltungen stattfinden (in umgekehrter Reihenfolge).

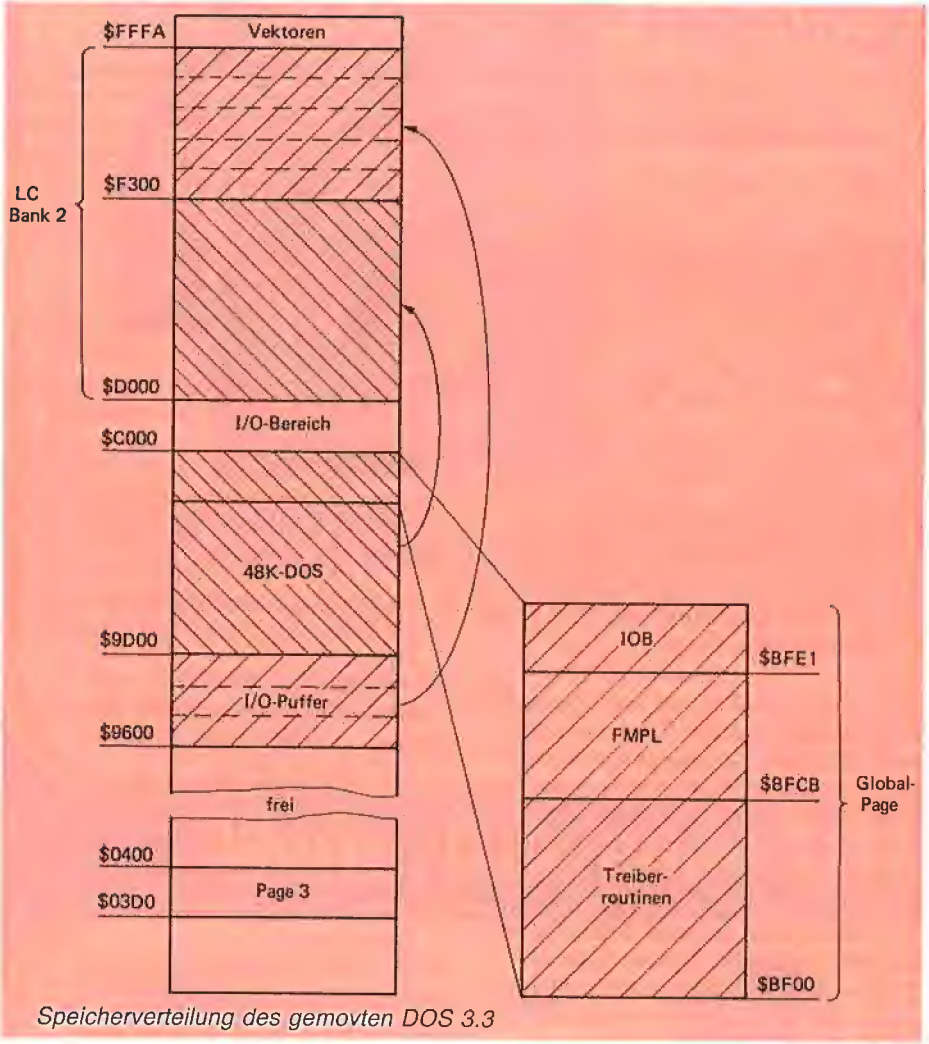
Es gibt jedoch auch noch einen anderen Bereich, auf den das DOS zugreift: die Page 3. Diese Speicherstellen dienen als Schnittstelle zwischen dem DOS und einem Anwenderprogramm.
Bei der Erstellung des Betriebssystems hielt sich die Firma Apple die Möglichkeit offen, eine neue, völlig geänderte Version auf den Markt zu bringen und veröffentlichte daher, im Gegensatz zum Monitor-ROM, keine internen Einsprungstellen. Jedes „seriöse" Programm durfte daher nur über die Page 3 Zugriff auf DOS nehmen. Dieser Umstand ist die wichtigste Voraussetzung für die Kompatibilität des DOS-Movers, da ausschließlich die Integrität dieser Page 3 sichergestellt werden muß. (Läuft ein Anwenderprogramm nicht mehr unter gemovtem DOS, ist es nicht „seriös".)
Der DOS-Mover stellt nun eine neue Verbindung zwischen Page 3 und DOS her und bedient sich dabei der Global-Page zur RAM/ROM-Umschaltung (siehe hierzu **Page 3 unter DOS 3.3**).

In Verbindung mit der Page 3 stellt sich ein anderes Problem. Beim Aufruf der RWTS oder des File-Managers müssen Parameter (Input/Output-Block oder File-Manager-Parameter-Liste) übergeben werden. Die Adresse dieser zwei Parameter-Blöcke liegt normalerweise innerhalb des DOS und kann über eine der Page-3-Routinen abgefragt werden. Da ein Anwenderprogramm jedoch keine Softswitches betätigt, um diese Blöcke, die jetzt in der LC liegen würden, zu lesen bzw. schreiben, müssen diese ebenfalls in einen ungebankten Bereich verlegt werden. Es liegt nahe, auch diese Bereiche in die Global-Page aufzunehmen und durch den Relokator alle Verweise auf diese Parameter durch die entsprechenden Adressen der Global-Page zu ersetzten.

1.3. Gepatchtes DOS

Nachdem die bisherigen Änderungen eher globaler Natur waren und mehr oder weniger „automatisch" ablaufen können, muß auch noch im Detail Hand angelegt werden, um das gemovte DOS an seinen neuen Standpunkt anzupassen. Diese einzelnen Patches werden vor dem Übertragen in die LC anhand einer Patch-Tabelle durchgeführt. Dabei mußten im einzelnen folgende Besonderheiten berücksichtigt werden:

FP/INT – Die Befehle zur Umschaltung der Sprache verlieren bei in die LC gemovtem DOS ihre Bedeutung. Aus diesem Grund wird Integer-BASIC nicht mehr unterstützt. (Dieser DOS-Mover ist somit für Geräte mit Integer-BASIC im ROM nicht geeignet.) DOS erkennt die aktuelle Sprache an einem Erkennung-Byte bei $E000. Integer-BASIC enthält dort $20, während Applesoft-BASIC durch $4C erkannt wird.
Da während des Prüfens dieser Speicherstelle im gemovten DOS die LC aktiv ist, muß jeder Zugriff auf $E000 durch eine Adresse ersetzt werden, die $4C enthält. Diese Änderung wird vom Relokator vor-



Speicherverteilung des gemovten DOS 3.3

genommen, der $E000 mit einer Global-Page-Adresse überschreibt.
Beim Test der aktuellen Sprache schaltet DOS zwischen RAM und ROM um. Dieser Teil muß lahmgelegt werden, da ansonsten das Bankswitching-Prinzip zum erliegen käme. Der Versuch, auf Integer-BASIC umzuschalten, endet somit in jedem Fall mit der Meldung „LANGUAGE NOT AVAILABLE".

MAXFILES – Die DOS-Puffer residieren nun auch in der LC, wobei aus Platzgründen jedoch nur noch 5 Puffer bereitgestellt werden können (siehe **Speicherverteilung unter gemovtem DOS**).
Zunächst muß die Anfangsadresse des ersten Puffers in die LC verlegt werden, das heißt in diesem Fall, daß nicht mehr die Anfangsadresse des DOS (normalerweise $9D00) zum Aufbau der sog. DOS-Puffer-Kette benutzt werden darf, sondern eine feste Adresse im LC-Bereich.

Der MAXFILES-Befehl muß dahingehend geändert werden, daß ab 6 Puffern die Meldung „OUT OF RANGE" erscheint (normalerweise 16 Puffer). Ansonsten würde das DOS durch die Puffer überschrieben werden (die Puffer liegen jetzt *überhalb* des DOS).
Wenn alle Puffer von oben nach unten initialisiert wurden, entspricht der Anfang des letzten (untersten) Puffers dem HIMEM-Wert. Hier würde HIMEM nun auf eine Adresse innerhalb des LC-Bereichs deuten. Aus diesem Grund wird stets der Anfang der Global-Page ($BF00) als System-Obergrenze eingesetzt.

INIT – Der normale INIT-Befehl besteht im wesentlichen aus vier Teilen: Zunächst wird via RWTS die Diskette formatiert. Danach muß die Catalog-Spur angelegt werden, die auch die VTOC enthält. Im Anschluß daran wird das im Speicher befindliche DOS auf die ersten drei Diskettenspuren übertragen. (Tatsächlich wird das DOS in zwei Blöcken in verdrehter Reihenfolge abgelegt, da die Boot-Routine im ersten Sektor zu liegen kommen muß, jedoch im Speicher im hinteren Teil unter der RWTS steht.) Als letztes wird das im Speicher befindliche Programm unter dem angegebenen Namen durch einen simulierten SAVE-Befehl auf die Diskette geschrieben.
Aus verständlichen Gründen ist das gemovte DOS nicht mehr zum Booten geeignet. Daher endet die gepatchte INIT-Routine nach dem zweiten Schritt. Da die Diskette nun aber kein DOS enthält, wird sie als Datendiskette angelegt, d.h. die Spuren 1 und 2 werden ebenfalls freigegeben, wodurch man weitere 8K an Kapazität gewinnt.
Die Angabe eines Namens beim INIT-Befehl ist im gemovten DOS hinfällig und verursacht eine Fehlermeldung.

Reset/NMI/IRQ – Da im Moment eines NMI- oder IRQ-Signals die LC aktiv sein kann, müssen die entsprechenden Vektoren, die sich hardware-mäßig nur am Ende des gesamten Adreßbereichs befinden können ($FFFA-$FFFF), in der LC gesetzt sein, d.h. in diesem Fall auf eine Routine in der Global-Page weisen, die die LC abschaltet und die normalen Interrupt-Routinen aufruft. Obwohl die LC nach einem Reset-Signal normalerweise deaktiviert ist, wurde auch der Reset-Vektor in diesem Sinne belegt, da Karten existieren, die nicht durch Reset abzustellen sind.
Wenn man nun z.B. die verschlungenen Pfade einer Reset-Behandlung verfolgt, wird klar, wieviele Komponenten einen Rechner zum Abstürzen bringen können. Nach einem Reset-Signal durchläuft der Prozessor folgende Schritte:
– Die Adresse der Reset-Routine wird über den Vektor ab $FFFC ermittelt.
– Die Reset-Routine ab $FA62 initialisiert den Monitor und setzt die Ein/Ausgabe-Vektoren für Tastatur und 40-Z/Z-Bildschirm.
– Am Ende der Reset-Routine wird nach Prüfung des Power-Up-Bytes (gegebenenfalls Kaltstart) über den Vektor ab $03F2 eine weitere Routine aufgerufen.
– $03F2 weist auf eine Global-Page-Routine, die die LC aktiviert und zum eigentlichen DOS-Warmstart verzweigt.
– Die Warmstart-Routine setzt u.a. die Ein/Ausgabe-Vektoren auf die entsprechenden DOS-Routinen und springt zurück in die Global-Page.
– Die aufgerufene Global-Page-Routine schaltet das ROM an und führt einen Applesoft-Warmstart aus.
Die darauffolgende Eingabe (und gleichzeitige Ausgabe) eines Befehls läuft nach einem nicht minder undurchsichtigen System ab.

1.4. Übergabe der Kontrolle

Ein springender Punkt bei der Installierung ist der Übergang vom normalen DOS zum gemovten DOS. Nachdem alle Anpassungen im normalen DOS durchgeführt wurden, wird es in die LC geschoben. Außer-

dem müssen die Interrupt-Vektoren initialisiert werden. Danach erfolgt die Übertragung der Global-Page und der neuen Page 3. Da ein Anwenderprogramm Werte aus dem RWTS-Input/Output-Block lesen kann, muß der aktuelle IOB ebenfalls in die Global-Page übertragen werden.
Danach wäre am einfachsten, einen DOS-Kaltstart durchzuführen. Diese Vorgehensweise würde jedoch dazu führen, daß der DOS-Mover nicht aus einem Applesoft-Programm gestartet werden kann, da ein DOS-Kaltstart im Anschluß einen Applesoft-Kaltstart durchführt. Daher werden die zur System-Initialisierung benötigten Routinen, die sich in diesem Stadium auf das Setzen der Ein/Ausgabe-Vektoren und den Neuaufbau der Puffer-Kette beschränken, einzeln aufgerufen bzw. selbst durchgeführt.
Zuletzt muß nur noch die Eingabezeile gelöscht werden (d.h. Zeilenlänge Null und Return als erstes Zeichen), da der DOS-Befehl „BRUN DOSMOVER" sonst vom Anwenderprogramm nochmals interpretiert würde, was im Applesoft-Direktmodus die Meldung „SYNTAX ERROR" zur Folge hätte.

2. Arbeiten mit gemovtem DOS

Bei aller Theorie soll noch einmal die Handhabung des DOS-Movers von der praktischen Seite beleuchtet werden, da vielleicht nicht jeder die oben beschriebenen Punkte in vollem Umfang verstanden hat. Die Änderungen gegenüber dem 48K-DOS lassen sich durch die folgenden Punkte beschreiben:

- Der hier vorgestellte DOS-Mover arbeitet nur auf Rechnern mit 48K plus LC, mit FP-BASIC im ROM.
- Der DOS-Mover muß stets durch BRUN DOSMOVER gestartet werden, nachdem *standardmäßiges DOS 3.3* gebootet wurde.
- Treffen diese Voraussetzungen nicht zu oder ist das DOS bereits gemovt, ertönt ein Klingelzeichen und das DOS bleibt unberührt. (Der DOS-Mover kann also auch im gemovten DOS ohne Konsequenzen nochmals aufgerufen werden, um sicherzustellen, daß in jedem Fall 64K-DOS vorliegt.)
- Der komplette Move-Vorgang dauert etwa eine Sekunde und darf *auf keinen Fall* durch Reset abgebrochen werden, da ansonsten das teilmodifizierte Betriebssystem den Rechner abstürzen ließe.
- Der Aufruf des DOS-Movers sollte im Direktmodus oder am Anfang des Programms vor der ersten Zuweisung von String-Variablen erfolgen, da HIMEM geändert wird. Siehe hierzu **DOSMOVER.START**.
- Der Aufruf des DOS-Movers kann auch durch einen EXEC-File erfolgen, der jedoch keine weiteren DOS-Befehle enthalten sollte.
- Wird der DOS-Mover im laufenden Programm gestartet, sollten keine Textdateien geöffnet sein, da deren Inhalt vor dem Sichern auf Diskette durch String-Variablen überschrieben werden könnte.
- Der MAXFILES-Befehl läßt nur noch maximal 5 DOS-Puffer zu und meldet ansonsten „OUT OF RANGE".
- Der INIT-Befehl erlaubt keine Angabe eines Dateinamens mehr und formatiert die Diskette als Datendiskette ohne DOS.
- Der INT-Befehl gibt in jedem Fall die Meldung „LANGUAGE NOT AVAILABLE" aus. Auch darf keineswegs versucht werden, Integer-BASIC durch BLOAD INTBASIC zu laden, da dadurch das DOS in der LC überschrieben würde.

Obwohl der DOS-Mover alle Regeln der Kompatibilität befolgt, läuft weder FID noch RENUMBER unter gemovtem DOS. In diesem Sinne sollte nur dann von dieser Möglichkeit der „Speichererweiterung" Gebrauch gemacht werden, wenn dies tatsächlich benötigt wird oder Vorteile bringt, wie z.B. bei COPYA, das das erweiterte Speicherangebot ausnutzt und somit eine Diskette in weniger Schüben kopiert.
COPYA muß jedoch folgendermaßen geändert werden, da der normale INIT-Befehl benutzt wird:

```
250 PRINT CHR$(4)"INIT,S";SS;
    ",D";SD;",V"; PEEK (714):FT = 1
```

Es spricht grundsätzlich nichts dagegen, den DOS-Mover auch für geänderte DOS-Versionen zu nutzen, die nur gepatcht und nicht neu assembliert wurden. Eine Überprüfung der Lauffähigkeit sei dem Leser überlassen, mit der Bitte, die entsprechenden Erfolge und Fehlschläge mitzuteilen.

Ein letztes Wort für die Leser, die vielleicht enttäuscht sind über die etwas kurzen Erläuterungen zum Programmablauf: Das genaue Verständnis aller Programmteile, insbesondere der einzelnen Patches, erfordert detaillierte Kenntnisse der DOS-Interna, die in einem einzigen Artikel nicht vermittelt werden können. Daher bleibt nur der tröstliche Hinweis auf die reichlichen Kommentare im Listing.

Kurzhinweise

1. Zweck:
Verschieben von DOS 3.3 in die LC zur Erweiterung des verfügbaren Speicherplatzes.
2. Konfiguration:
Apple II+ (mit LC und FP-BASIC im ROM), IIe oder IIc;
DOS 3.3.
3. Test:
RUN DOSMOVER.START
4. Sammeldisk:
DOSMOVER.START
(Applesoft-HELLO-Programm)
T.DOSMOVER
(Big-Mac-Quelltext)
DOSMOVER
(Maschinenprogramm)

Speicheraufteilung unter gemovtem DOS (LC, Bank 2)

```
normal    gemovtes DOS

$96ØØ -> $BFØØ       : HIMEM (ergibt lØ488 Bytes zusätzlich)
         $BFØØ-$BFCA: Global-Page-Vektoren
$B5BB -> $BFCB-$BFEØ: File-Manager-Parameter-Liste
$B7E8 -> $BFEl-$BFFl: RWTS-Input/Output-Block
         $BFF2-$BFFF: frei

$9DØØ -> $DØØØ
$9EØØ -> $DlØØ
$9FØØ -> $D2ØØ
$AØØØ -> $D3ØØ
   ...             : gemovtes DOS

$BDØØ -> $FØØØ
$BEØØ -> $FlØØ
$BFØØ -> $F2ØØ

         $F3ØØ-$F552: 5. Puffer
         $F553-$F7A5: 4. Puffer
$96ØØ -> $F7A6-$F9F8: 3. Puffer
$9853 -> $F9F9-$FC4B: 2. Puffer
$9AA6 -> $FC4C-$FE9E: 1. Puffer
         $FE9F-$FFF9: frei
$FFFA -> $FFFA-$FFFB: NMI-Vektor
$FFFC -> $FFFC-$FFFD: Reset-Vektor
$FFFE -> $FFFE-$FFFF: IRQ-Vektor
```

DAS APPLE II HANDBUCH

Die rasche Orientierung für APPLE IIplus, IIe, IIc

Schnelle Antwort auf Alltagsfragen am APPLE II – leicht nachzuschlagen, praxisbezogen, für IIplus, IIe, IIc in nur 1 Buch!
Unterschiede IIplus/IIe, DOS 3.3/ProDOS, E/A-Interfacekarten/Ports, 40/80 Zeichendarstellung, US/DTS-Tastatur, 48K/128K Systeme etc.
Grafik/Soundmöglichkeiten, eine der APPLE-Stärken, in stark erweiterter Beschreibung.
Kurzführer „Steckkartenerweiterungen" mit Fotos; „Sofortbetrieb von Disketten/Cassettengeräten"; „Druckerbetrieb"; „Direktbefehle"; „Tastaturbedienung" etc.

Backgroundwissen: BASIC für Beginner/Professionelle; MC/BASIC-Kombination; MC-Entwicklung mit MONITOR/MINIASSEMBLER; APPLE-PASCAL-BS; Disketten/Plattenspeicherung; Dateiformate etc.
Ausführlicher Anhang zu Editor, Speicherbelegung, Codes des APPLESOFT-Interpreters etc.

DAS APPLE II HANDBUCH für IIplus, IIe, IIc, 472 Seiten, Softcover, DM 66,–

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40
te-wi

Weiterführende Literatur...

NEU

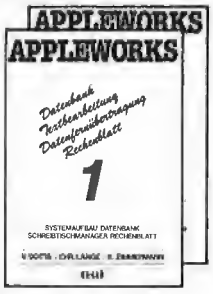
Reparaturanleitung Computer: Apple II, IIplus
Einzigartige Serviceunterläge für Reparaturen und Entwicklungsarbeiten am Apple II. Enthält Schaltpläne, Bauteile- und Vergleichstypenliste; Prüfpunkte mit Oszillogrammen der Signalformen, Logiktabellen, Spannungsangaben; schnelle Servicetests; Anleitung zur systematischen Fehlersuche. In A4-Mappe, DM 29,80

LOGO – Jeder kann programmieren
(Daniel Watt)
Buch des Jahres in den U SA. Für die Computer APPLE II, C-64, IBM PC, ATARI bis 520 ST., TI-99 und Schneider CPCs.
Hochwertiges Textbuch für Logo-Kurse für zu Hause und im Lehrbereich.
384 Seiten, A4, DM 59,–

NEU

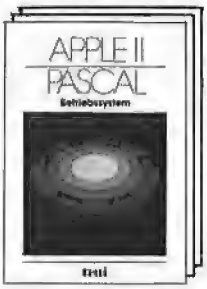
APPLE II – Bewegte 3D-Graphik
(Phil Cohen)
Selbstentworfene Graphiken und Diagramme – animiert oder als Standbilder – eben oder räumlich: alle erforderlichen BASIC-Programme mit Erklärung finden Sie in diesem Buch.
200 Seiten, Softcover, DM 49,–

Apple Maschinensprache
Für BASIC-Programmierer der einfachste Zugang zur Muttersprache des Apple. Wesentlich schnellere Maschinenprogramme, direkte Manipulation des Mikroprozessors 6502 im Apple – als Brücke dorthin benötigt dieses Buch nur die drei BASIC-Befehle, POKE, CALL, PEEK. D. Inman/K. Inman. DM 49,–



NEU

APPLE WORKS + APPLE II, IIe, IIc = Elektronischer Schreibtischmanager
Dieses Programmsystem vereinigt die Funktion Texterstellung, Datenarchivierung, Formblattkalkulation, Datenfernübertragung. Das System mit den höchsten Verkaufsziffern. Sämtliche System/Anwendungsfragen in 2 Bänden. Beispiele aus der Wirtschaft u. v. m.
je 264 Seiten, je DM 49,–



Erstes deutsches Referenzwerk sämtlicher Befehle und Systemroutinen von Apple II, IIplus, IIe
APPLE II PASCAL NEU
Betriebssystem, 272 S., DM 49,–
Sprache, 216 S., DM 39,–
Pascal 1.2 Addendum, 112 S., DM 36,–

Grundlagenbuch, Bestseller
APPLE II PASCAL,
Eine praktische Anleitung,
544 S., DM 59,–

tm 4439

Noch im Programm:
6502 – Programmieren in Assembler DM 59,–
VisiCalc, 50 Programme auf Diskette, DM 79,–
Computer für Kinder, APPLE II, DM 29,80

In Vorbereitung:
Macintosh Programmier-Handbuch mit Microsoft BASIC 2.0 DM 59,–

Page 3 unter DOS 3.3

(nur die von DOS belegten Adressen)

normales DOS

```
Ø3DØ- 4C BF 9D   JMP $9DBF   DOS-Warmstart
Ø3D3- 4C 84 9D   JMP $9D84   DOS-Kaltstart
Ø3D6- 4C FD AA   JMP $AAFD   File-Manager-Aufruf
Ø3D9- 4C B5 B7   JMP $B7B5   RWTS-Aufruf
Ø3DC- AD ØF 9D   LDA $9DØF   Bestimmung der
Ø3DF- AC ØE 9D   LDY $9DØE    Adresse des
Ø3E2- 6Ø         RTS          Input/Output-Blocks
Ø3E3- AD C2 AA   LDA $AAC2   Bestimmung der
Ø3E6- AC C1 AA   LDY $AAC1    Adresse der
Ø3E9- 6Ø         RTS          File-Manager-Parameter-Liste
Ø3EA- 4C 51 A8   JMP $A851   DOS-Vektoren anhängen
```

gemovtes DOS

```
Ø3DØ- 4C 34 BF   JMP $BF34   DOS-Warmstart
Ø3D3- 4C 2E BF   JMP $BF2E   DOS-Kaltstart
Ø3D6- 4C 12 BF   JMP $BF12   File-Manager-Aufruf
Ø3D9- 4C 1B BF   JMP $BF1B   RWTS-Aufruf
Ø3DC- A9 BF      LDA #$BF    Bestimmung der
Ø3DE- AØ CB      LDY #$CB     Adresse des
Ø3EØ- 6Ø         RTS          Input/Output-Blocks
Ø3E1- EA         NOP
Ø3E2- EA         NOP
Ø3E3- A9 BF      LDA #$BF    Bestimmung der
Ø3E5- AØ E1      LDY #$E1     Adresse der
Ø3E7- 6Ø         RTS          File-Manager-Parameter-Liste
Ø3E8- EA         NOP
Ø3E9- EA         NOP
Ø3EA- 4C 24 BF   JMP $BF24   DOS-Vektoren anhängen
```

DOSMOVER.START

```
1Ø HOME : PRINT TAB( 9)"DOS-Mover": PRINT
2Ø PRINT "48K-DOS: " FRE (Ø) + 16 ↑ 4" Bytes frei"
3Ø PRINT : PRINT "DOS verschieben? (J/N) ";
4Ø GET A$: PRINT A$: IF A$ < > "J" THEN END
5Ø PRINT CHR$ (4)"BRUN DOSMOVER": PRINT
6Ø PRINT "64K-DOS: " FRE (Ø) + 16 ↑ 4" Bytes frei"
```

BSAVE DOSMOVER, A$8ØØØ, L$Ø35Ø

DOSMOVER

```
1   *-------------------------------------------------*
2   *                                                 *
3   *                    DOS-Mover                    *
4   *                                                 *
5   *                Harald Grumser, 1985             *
6   *-------------------------------------------------*
7
8            ORG  $8ØØØ
9
1Ø  LENGTH   EQU  $2F        ;OP-Code-Länge - 1
11  CSW      EQU  $36        ;Ausgabeadresse
12  KSW      EQU  $38        ;Eingabeadresse
13  PC       EQU  $3A        ;Programmzähler
14  IND      EQU  $3C        ;Hilfsregister
15  OPRND    EQU  IND        ;Operand
16  FRETOP   EQU  $6F        ;BASIC-String-Ende
17  MEMSIZE  EQU  $73        ;= HIMEM
18
19  IN       EQU  $2ØØ       ;Eingabepuffer
2Ø  DOSWARM  EQU  $3DØ       ;DOS-Warmstart
21  NMI      EQU  $3FB       ;Sprung zur NMI-Routine
22
23  ROMSEL   EQU  $CØ81      ;ROM lesen (Bank 2)
24  RAMSEL   EQU  $CØ83      ;RAM lesen (Bank 2)
25
26  INSDS2   EQU  $F88C      ;OP-Code auswerten
27  PCADJ    EQU  $F953      ;PC um LENGHT eröhen
28  IRQ      EQU  $FA4Ø      ;Monitor-IRQ
29  RESET    EQU  $FA62      ;Monitor-Reset
3Ø  BELL1    EQU  $FBDD      ;Klingelzeichen
31
32  * ROM-Referenzen des DOS (Auszug)
33
34  RESTART  EQU  $D43C      ;BASIC-Warmstart
35  LINKSET  EQU  $D4F2      ;Linkbytes neu setzen
36  SETPTRS  EQU  $D665      ;CLEAR durchführen
37  NEWSTT   EQU  $D7D2      ;nächsten Befehl ausführen
38  ERRENT   EQU  $D865      ;Einsprung für Fehlermeldung
39  BASCOLD  EQU  $EØØØ      ;BASIC-Kaltstart
4Ø  RDKEY    EQU  $FDØC      ;Zeichen lesen
41  COUT     EQU  $FDED      ;Zeichen ausgeben
42  INPORT   EQU  $FE8B      ;Eingabe-Vektor setzen
43  OUTPORT  EQU  $FE95      ;Ausgabe-Vektor setzen
44
45  *-------------------------------------------------*
46  *                     Start                       *
47
48           LDA  $AAB6      ;BASIC-Flag
49           CMP  #$4Ø       ;FP-BASIC?
5Ø           BNE  ERROR      ;nein, dann DOS belassen
51           PLA             ;DOS-Mover muß durch BRUN
52           CMP  #<$A17C    ; gestartet werden und er-
53           BNE  ERROR      ; kennt daran, ob es sich
54           PLA             ; um 48K-DOS handelt
55           CMP  #>$A17C    ;ansonsten erfolgt ein
56           BEQ  RELOC      ; Warmstart ohne Move
57  ERROR    JSR  BELL1      ;zur Warnung!
58           JMP  DOSWARM
59
6Ø  *-------------------------------------------------*
61  *                    Relokator                    *
62
63  RELOC    LDA  #<$9D84    ;Startadresse
64           LDY  #>$9D84    ;($9DØØ-$9D83 sind Daten)
65           STA  PC         ; in Programm-Counter
66           STY  PC+1       ; eintragen
67
68  * Prüfen, ob PC innerhalb eines Datenbereichs
69
7Ø  DATA     LDX  #DATTBLE1-DATTBLE
71  DATA1    DEX             ;Tabellenende?
72           BMI  NODATA     ;ja, dann kein Datenbereich
73           LDA  DATTBLE,X
74           DEX
75           CMP  PC+1       ;High-Byte vergleichen
76           BNE  DATA1
77           LDA  DATTBLE,X
78           CMP  PC         ;Low-Byte vergleichen
79           BNE  DATA1
8Ø           LDA  DATTBLE1,X ;neue PC-Adresse
81           STA  PC         ; übernehmen
82           LDA  DATTBLE1+1,X
83           STA  PC+1
84
85  NODATA   LDX  #$ØØ
86           JSR  INSDS2     ;Länge & Format bestimmen
87           LDY  #$Ø2
88           LDA  (PC),Y     ;Operand, High-Byte
89           STA  OPRND+1
9Ø           DEY
91           LDA  (PC),Y     ;Operand, Low-Byte
92           STA  OPRND
93           DEY
94           LDA  (PC),Y     ;OP-Code
95           CMP  #$2Ø       ;JSR?
96           BEQ  REFS
97           CMP  #$4C       ;JMP?
98           BNE  NOJMP
99
1ØØ * Prüfen, ob Sprung ins ROM
1Ø1
1Ø2 REFS     LDX  #REFTBLE1-REFTBLE
1Ø3 REFS1    DEX             ;Tabellenende?
1Ø4          BMI  INTLADR    ;ja, dann interner Sprung
1Ø5          LDA  REFTBLE,X
1Ø6          DEX
1Ø7          CMP  OPRND+1    ;High-Byte vergleichen
1Ø8          BNE  REFS1
1Ø9          LDA  REFTBLE,X
11Ø          CMP  OPRND      ;Low-Byte vergleichen
111          BNE  REFS1
112          LDA  REFTBLE1+1,X
113          PHA             ;neue Adresse
```

```
114           LDA  REFTBLE1,X ;High-Byte -> Akku
115           TAX               ; Low-Byte -> X-Reg.
116           PLA
117           JMP  INTLADR1
118
119 NOJMP      LDA  LENGTH      ;Adressierungsart bestimmen
120           CMP  #$02         ;(ABS)/ ABS/ ABS,X/ ABS,Y
121           BNE  NXTOPC       ;alle anderen ignorieren
122 INTLADR   LDA  OPRND+1      ;liegt Adresse innerhalb
123           CMP  #>$9D00      ; des DOS-Bereichs?
124           BCC  NXTOPC       ;nein, dann belassen
125           CMP  #>$C000
126           BEQ  NXTOPC
127           LDX  OPRND
128           BNE  NOE000       ;$E000 muß durch
129           CMP  #>$E000      ; FPID ersetz werden
130           BNE  NOE000
131           LDX  #<FPID
132           LDA  #>FPID
133           BNE  INTLADR1     ;unbedingt
134
135  * Referenzen auf IOB oder FMPL nach Global-Page legen
136
137 NOE000    CMP  #>$B7E8      ;Referenz auf IOB
138           BNE  NOIOB
139           CPX  #<$B7E8      ;Low-Byte überprüfen
140           BCC  NOIOB        ;(darunter)
141           CPX  #<$B7FB
142           BCS  NOIOB        ;(darüber)
143           TXA               ;Low-Byte
144           ADC  #NEWIOB-$B7E8
145           TAX               ; anpassen
146           LDA  #>NEWIOB     ;neues High-Byte
147           BNE  INTLADR1     ; und eintragen
148
149 NOIOB     CMP  #>$B5A9      ;Referenz auf FMPL
150           BNE  NOFMPL       ; (erniedrigt wegen Offset)
151           CPX  #<$B5A9      ;Low-Byte überprüfen
152           BCC  NOFMPL       ;(darunter)
153           CPX  #<$B5CD
154           BCS  NOFMPL       ;(darüber)
155           TXA               ;Low-Byte
156           ADC  #NEWFMPL-$B5BB
157           TAX               ; anpassen
158           LDA  #>NEWFMPL    ;neues High-Byte
159           BNE  INTLADR1     ; und eintragen
160
161  * Interne Referenzen anpassen und eintragen
162
163 NOFMPL    CLC
164           ADC  #$D0-$9D     ;Verschiebe-Offset
165 INTLADR1  LDY  #$02
166           STA  (PC),Y       ;neuer Operand, High-Byte
167           TXA
168           DEY
169           STA  (PC),Y       ;neuer Operand, Low-Byte
170 NXTOPC    JSR  PCADJ        ;PC =
171           STA  PC           ; PC + LENGTH
172           STY  PC+1
173           CPY  #>$C000      ;DOS-Ende erreicht?
174           BEQ  SETADRS      ;ja, dann weiter
175           JMP  DATA         ;ansonsten weiter in Schleife
176
177 *-------------------------------------------------------*
178 *                Adreßtabellen anpassen                 *
179
180 SETADRS   CLC
181           LDX  #35*2-1      ;35 Adressen
182 CMDIADRS  LDA  $9D10,X      ; für den
183           ADC  #$D0-$9D     ; Command-Interpreter
184           STA  $9D10,X      ; berichtigen
185           DEX               ;(jeweils nur das
186           DEX               ; High-Byte)
187           BPL  CMDIADRS
188           LDX  #26*2-1      ;26 Adressen
189 FMADRS    LDA  $AAC9,X      ; für den
190           ADC  #$D0-$9D     ; File-Manager
191           STA  $AAC9,X      ; berichtigen
192           DEX
193           DEX
194           BPL  FMADRS
195
196 *-------------------------------------------------------*
197 *                  Patches durchführen                  *
198
199           LDA  #<PTCHTBLE+3 ;Adresse des
200           STA  IND          ;  ersten Patch-Bytes
201           LDA  #>PTCHTBLE+3 ; eintragen
202           STA  IND+1
203           LDA  PTCHTBLE     ;Adresse der
204           STA  PC           ; ersten Patch-Adresse
205           LDA  PTCHTBLE+1   ; eintragen
206           STA  PC+1
207           LDX  PTCHTBLE+2   ;Anzahl der zu
208 PATCH     TXA               ; überschreibenden
209           TAY               ; Bytes
210           DEY
211 PATCH1    LDA  (IND),Y      ;Patch
212           STA  (PC),Y       ; durchführen
213           DEY
214           BPL  PATCH1
215           TXA               ;Patch-Länge =
216           TAY               ; Index auf nächste
217           LDA  (IND),Y      ; Patch-Adresse
218           STA  PC
219           INY               ;neue
220           LDA  (IND),Y      ; Patch-Adresse
221           BEQ  MOVER        ;(Tabellenende erreicht)
222           STA  PC+1
223           INY               ;Index auf Länge
224           LDA  (IND),Y      ;Patch-Länge des
225           TAX               ; nächsten Bereichs
226           SEC               ; (statt INY)
227           TYA               ; zu aktuellem Bereich
228           ADC  IND          ; addiert ergibt
229           STA  IND          ; neuen Adresse des
230           BCC  PATCH        ; ersten Patch-Bytes
231           INC  IND+1
232           BNE  PATCH        ;unbedingt
233
234 *-------------------------------------------------------*
235 * DOS verschieben, Systemvektoren setzen, DOS-Neustart  *
236
237 MOVER     BIT  ROMSEL       ;ROM lesen
238           BIT  ROMSEL       ;RAM schreiben
239           LDA  #>$9D00
240           STA  IND+1        ;DOS von $9D00
241           LDA  #>$D000      ; nach $D000
242           STA  PC+1         ; verschieben
243           LDY  #0
244           STY  IND          ;Die Speicheraufteilung
245           STY  PC           ; sieht dann wie folgt aus:
246 MOVE      LDA  (IND),Y
247           STA  (PC),Y       ; $BF00 - Global-Page
248           INY               ; $C000 - I/O-Bereich
249           BNE  MOVE         ; $D000 - gemovtes DOS
250           INC  IND+1        ; $F300 - 5 Puffer
251           INC  PC+1
252           LDA  PC+1
253           CMP  #>$F300
254           BNE  MOVE
255
256           LDY  #$06
257 VECMOVE   LDA  VEKTBLE-1,Y ;neuen NMI-, Reset-
258           STA  $FFFA-1,Y    ; und IRQ-Vektor
259           DEY               ; in LC ablegen
260           BNE  VECMOVE
261
262 GLBMOVE   LDA  GLBIMGE,Y   ;Gobal-Page-Image
263           STA  GLOBAL,Y     ; in letzte RAM-
264           INY               ; Seite übertragen
265           BNE  GLBMOVE
266
267           LDY  #$10
268 COPYIOB   LDA  $B7E8,Y      ;alten IOB
269           STA  NEWIOB,Y     ; in neuen IOB
270           DEY               ; übertragen
271           BPL  COPYIOB
272
273           LDA  #<NEWOUT     ;neue
274           STA  CSW          ; Ein/Ausgabe-
275           LDA  #>NEWOUT     ; Vektoren
276           STA  CSW+1        ; eintragen
277           LDA  #<NEWIN
278           STA  KSW
279           LDA  #>NEWIN
280           STA  KSW+1
281           BIT  RAMSEL       ;RAM lesen/schreiben
282           LDA  #$60         ;RTS nach Page-3-Move
283           STA  $D141        ; eintragen und Page-3-
284           JSR  $D125        ; Image kopieren
285           JSR  $D106        ;Puffer und HIMEM initialisieren
286           LDA  #$8D         ;leere Eingabezeile
287           STA  IN           ; vortäuschen
288           LDX  #0           ;(Zeilenlänge von GETLN)
289           JMP  ENBLEROM     ;E N D E
290
291 GLBIMGE   EQU  *
292 *-------------------------------------------------------*
293 *                     Global-Page                       *
294 *-------------------------------------------------------*
295           ORG  $BF00
296 GLOBAL
297 NEWIN     JSR  ENBLERAM     ;neue Eingabe-Intercept-
```

```
298             JSR   $D181       ; Routine
299             JMP   ENBLEROM
300 NEWOUT      JSR   ENBLERAM    ;neue Ausgabe-Intercept-
301             JSR   $D1BD       ; Routine
302             JMP   ENBLEROM
303 NEWFM       JSR   ENBLERAM
304             JSR   $DDFD       ;neuer File-Manager-Aufruf
305             JMP   ENBLEROM
306 NEWRWTS     JSR   ENBLERAM
307             JSR   $EAB5       ;neuer RWTS-Aufruf
308             JMP   ENBLEROM
309 NEWCNCT     JSR   ENBLERAM    ;neuer Aufruf
310             JSR   $DB51       ; um DOS anzuhängen
311 ENBLEROM     BIT   ROMSEL
312             RTS
313
314 NEWCOLD     JSR   ENBLERAM
315             JMP   $D084       ;neue Kaltstart-Adresse
316 NEWWARM     JSR   ENBLERAM
317 FPID        JMP   $D0BF       ;neue Warmstart-Adresse
318
319 * FPID (FP-Erkennung-Byte) muß $4C enthalten
320
321 EXTWARM     BIT   ROMSEL
322             JMP   RESTART     ;BASIC-Warmstart
323 EXTLNKST    BIT   ROMSEL
324             JMP   LINKSET     ;Programm neu linken
325 EXTCOLD     BIT   ROMSEL
326             JMP   BASCOLD     ;BASIC-Kaltstart
327 EXTNSTT     BIT   ROMSEL
328             JMP   NEWSTT      ;neuen BASIC-Befehl ausführen
329 EXTERR      BIT   ROMSEL
330             JMP   ERRENT      ;BASIC-Fehlerausgabe
331 EXTNMI      BIT   ROMSEL
332             JMP   NMI         ;NMI-Routine
333 EXTREST     BIT   ROMSEL
334             JMP   RESET       ;Reset-Routine
335 EXTIRQ      BIT   ROMSEL
336             JMP   IRQ         ;IRQ-Routine
337
338 EXTBRUN     LDA   $DD72       ;BRUN-Ladeadresse
339             STA   BRUNADR+1   ; in Dummy-Adreßfeld
340             LDA   $DD73       ; übertragen
341             STA   BRUNADR+2
342             BIT   ROMSEL      ; und Routine
343 BRUNADR     JSR   $FFFF       ; aufrufen
344             JMP   ENBLERAM
345 EXTSTPT     BIT   ROMSEL
346             JSR   SETPTRS     ;CLEAR durchführen
347             JSR   ENBLERAM    ;(kein RTS, da Stack neu
348             JMP   $D7FF       ; initialisiert wurde)
349 EXTRDKEY    BIT   ROMSEL
350             JSR   RDKEY       ;Zeichen einlesen
351             JMP   ENBLERAM
352 EXTCOUT     BIT   ROMSEL
353             JSR   COUT        ;Zeichen ausgeben
354             JMP   ENBLERAM
355 EXTOUTPT    BIT   ROMSEL
356             JSR   OUTPORT     ;Ausgabe-Slot festlegen
357             JMP   ENBLERAM
358 EXTINPRT    BIT   ROMSEL
359             JSR   INPORT      ;Eingabe-Slot festlegen
360             JMP   ENBLERAM
361 EXTCSW      BIT   ROMSEL
362             JSR   CSWIND      ;Ausgaberoutine
363             JMP   ENBLERAM
364 EXTKSW      BIT   ROMSEL      ;ROM lesen
365             JSR   KSWIND      ;Eingaberoutine
366 ENBLERAM    BIT   RAMSEL      ;RAM lesen
367             BIT   RAMSEL      ;RAM schreiben
368             RTS
369
370 CSWIND      JMP   (CSW)
371 KSWIND      JMP   (KSW)
372
373 NEWFMPL     EQU   *           ;neue FMPL
374 NEWIOB      EQU   *+22        ;neuer IOB
375
376 *-------------------------------------------------------*
377 *                  Tabellen und Patches                 *
378 *-------------------------------------------------------*
379             ORG   *-GLOBAL+GLBIMGE
380
381 * Tabelle der Datenbereiche
382
383 DATTBLE     DA    $9E51       ;Page-3-Image
384             DA    $A884       ;Strings und Variablen
385             DA    $B397       ;FM-Variablen und -tabellen
386             DA    $B6B3       ;FMPL und Boot-Image
387             DA    $B7DF       ;u.a. IOB
388             DA    $BA11       ;RWTS-Tabellen (Nibble)
389             DA    $BFA8       ;Sektor- und Skewing-Tabelle
```

```
390 DATTBLE1    DA    $9E81       ;(Endadressen
391             DA    $AAFD       ; der o.g.
392             DA    $B65E       ; Datenbereiche)
393             DA    $B7B5
394             DA    $B800
395             DA    $BC56
396             DA    $BFDC
397
398 * Tabelle der externen DOS-Referenzen
399
400 REFTBLE     DA    SETPTRS     ;Original-Adresse
401             DA    NEWSTT
402             DA    RDKEY
403             DA    COUT
404             DA    INPORT
405             DA    OUTPORT
406 REFTBLE1    DA    EXTSTPT     ;neue Adresse in
407             DA    EXTNSTT     ; der Global-Page
408             DA    EXTRDKEY
409             DA    EXTCOUT
410             DA    EXTINPRT
411             DA    EXTOUTPT
412
413 * NMI-, Reset-, und IRQ-Vektortabelle
414
415 VEKTBLE     DA    EXTNMI
416             DA    EXTREST
417             DA    EXTIRQ
418
419 * Patch-Tabelle
420
421 PTCHTBLE    DA    $9E51       ;Page-3-Image
422             DFB   29
423             JMP   NEWWARM     ;DOS-Warmstart
424             JMP   NEWCOLD     ;DOS-Kaltstart
425             JMP   NEWFM       ;File-Manager-Aufruf
426             JMP   NEWRWTS     ;RWTS-Aufruf
427             LDA   #>NEWFMPL   ;Position der
428             LDY   #<NEWFMPL   ; neuen FMPL
429             RTS               ; bestimmen
430             NOP
431             NOP
432             LDA   #>NEWIOB    ;Position des
433             LDY   #<NEWIOB    ; neuen IOB
434             RTS               ; bestimmen
435             NOP
436             NOP
437             JMP   NEWCNCT     ;DOS anhängen
438
439             DA    $9D00       ;Relokator-Adreßtabelle
440             DFB   16          ; (für Masterdisk)
441             DA    $FE79       ;Adresse des 1. Puffers
442             DA    NEWIN       ;Eingaberoutine (KBD)
443             DA    NEWOUT      ;Ausgaberoutine (VID)
444             DA    $DD75       ;Adresse des 1. Namenspuffer
445             DA    $DD93       ;Adresse des 2. Namenspuffer
446             DA    $DD60       ;Adresse der Dateilänge
447             DA    $DD00       ;DOS-Adresse (ehem. für INIT)
448             DA    NEWFMPL     ;Adresse der FMPL
449
450             DA    $9DB7
451             DFB   1
452             DFB   <$9D6B      ;BASIC-Link nicht überschreiben
453
454             DA    $9D56       ;FP-BASIC-Linktabelle
455             DFB   12
456             DA    $D7FC       ;RUN-Kommando
457             DA    $D7FC       ;CHAIN-Kommando
458             DA    EXTERR      ;Fehlermeldung ausgeben
459             DA    EXTCOLD     ;BASIC-Kaltstart
460             DA    EXTWARM     ;BASIC-Warmstart
461             DA    EXTLNKST    ;Programm neu linken
462
463             DA    $A25D       ;MAXFILES-Patch
464             DFB   3
465             JMP   $D6BC       ;(Integer-BASIC-Bereich
466             DA    $A3BC       ; kann überschrieben werden)
467             DFB   13
468             CMP   #6          ;maximal 5 Puffer
469             BCC   MXFLSOK     ;weniger, dann weiter
470             JMP   $D3C9       ;"RANGE ERROR"
471 MXFLSOK     STA   $DD57       ;MAXFILES
472             JMP   $DAD4       ;Pufferkette neu bilden
473             DA    $AAB1       ;(Default-MAXFILES)
474             DFB   1
475             DFB   5           ;5 Puffer als Default
476
477             DA    $A5B7
478             DFB   1
479             RTS               ;kein Intger-BASIC anwählen
480
```

```
481           DA   $A83B       ;HIMEM-Patch
482           DFB  15
483           PLA              ;(ehm. HIMEM)
484           PLA
485           LDA  #<GLOBAL    ;neuer
486           STA  MEMSIZE     ; HIMEM-Wert
487           STA  FRETOP      ; liegt unterhalb
488           LDA  #>GLOBAL    ; der Global-Page
489           STA  MEMSIZE+1
490           STA  FRETOP+1
491           RTS
492
493           DA   $AAC1       ;FMPL-Pufferadressen
494           DFB  6
495           DA   NEWIOB      ;neuer IOB
496           DA   $E6BB       ;interner VTOC-Puffer
497           DA   $E7BB       ;interner Directory-Puffer
498
499           DA   $A56B       ;INIT-Patch
500           DFB  1
501           RTS              ;kein HELLO-Programm speichern
502           DA   $A909
503           DFB  1
504           DFB  $41         ;INIT jetzt ohne Namensangabe
505           DA   $AEB3
```

Hinweis zur Patch-Tabelle

Die Bearbeitung der Patch-Tabelle wurde im Hinblick auf eine Erweiterung programmiert und erlaubt das Hinzufügen weiterer Patch-Bereiche in beliebiger Reihenfolge (z.B. CATALOG-Ctrl-C-Patch). Jeder Patch-Bereich muß folgendermaßen aufgebaut sein:

- Patch-Adresse (2 Bytes)
- Länge des zu patchenden Bereichs (1 Byte)
- eigentlicher Patch (max. 256 Bytes)

Das Ende einer Tabelle muß durch zwei Nullen abgeschlossen werden.

```
506           DFB  1
507           DFB  $04         ;VTOC ab Spur 1 freigeben
508           DA   $AEFF
509           DFB  3
510           JMP  $E67F       ;kein DOS-Image eintragen
511
512           DA   $B7E4       ;Adreß-Pointer zum IOB
513           DFB  2
514           DA   NEWIOB      ;IOB nun in der Global-Page
515
516           DA   $B7E8+6     ;DCT-Adresse im IOB
517           DFB  2
518           DA   $EAFB       ;DCT bleibt in der LC
519
520  * Indirekte Sprünge werden nicht vom Relokator korrigiert
521
522           DA   $9EBA       ;indirekter Sprung zur
523           DFB  3           ; aktuellen Eingabe
524           JMP  EXTKSW
525           DA   $9FC5       ;indirekter Sprung zur
526           DFB  3           ; aktuellen Ausgabe
527           JMP  EXTCSW
528           DA   $A394       ;indirekter Sprung zur
529           DFB  3           ; BRUN-Adresse
530           JMP  EXTBRUN
531           DA   0           ;Endmarker
```

Hexdump DOSMOVER

BSAVE DOSMOVER, A$8000, L$0350

```
8000: AD B6 AA C9 40 D0 0A 68
8008: C9 7C D0 05 68 C9 A1 F0
8010: 06 20 DD FB 4C D0 03 A9
8018: 84 A0 9D 85 3A 84 3B A2
8020: 0E CA 30 19 BD 67 82 CA
8028: C5 3B D0 F5 BD 67 82 C5
8030: 3A D0 EE BD 75 82 85 3A
8038: BD 76 82 85 3B A2 00 20
8040: 8C F8 A0 02 B1 3A 85 3D
8048: 88 B1 3A 85 3C 88 B1 3A
8050: C9 20 F0 04 C9 4C D0 20
8058: A2 0C CA 30 21 BD 83 82
8060: CA C5 3D D0 F5 BD 83 82
8068: C5 3C D0 EE BD 90 82 48
8070: BD 8F 82 AA 68 4C C1 80
8078: A5 2F C9 02 D0 4B A5 3D
8080: C9 9D 90 45 C9 C0 F0 41
8088: A6 3C D0 0A C9 E0 D0 06
8090: A2 37 A9 BF D0 2B C9 B7
8098: D0 10 E0 E8 90 0C E0 FB
80A0: B0 08 8A 69 F9 AA A9 BF
80A8: D0 17 C9 B5 D0 10 E0 A9
80B0: 90 0C E0 CD B0 08 8A 69
80B8: 10 AA A9 BF D0 03 18 69
80C0: 33 A0 02 91 3A 8A 88 91
80C8: 3A 20 53 F9 85 3A 84 3B
80D0: C0 C0 F0 03 4C 1F 80 18
80D8: A2 45 BD 10 9D 69 33 9D
80E0: 10 9D CA CA 10 F4 A2 33
80E8: BD C9 AA 69 33 9D C9 AA
80F0: CA CA 10 F4 A9 A4 85 3C
80F8: A9 82 85 3D AD A1 82 85
8100: 3A AD A2 82 85 3B AE A3
8108: 82 8A A8 88 B1 3C 91 3A
8110: 88 10 F9 8A A8 B1 3C 85
8118: 3A C8 B1 3C F0 12 85 3B
8120: C8 B1 3C AA 38 98 65 3C
8128: 85 3C 90 DD E6 3D D0 D9
8130: 2C 81 C0 2C 81 C0 A9 9D
8138: 85 3D A9 D0 85 3B A0 00
8140: 84 3C 84 3A B1 3C 91 3A
8148: C8 D0 F9 E6 3D E6 3B A5
8150: 3B C9 F3 D0 EF A0 06 B9
8158: 9A 82 99 F9 FF 88 D0 F7
8160: B9 9C 81 99 00 BF C8 D0
8168: F7 A0 10 B9 E8 B7 99 E1
8170: BF 88 10 F7 A9 09 85 36
8178: A9 BF 85 37 A9 00 85 38
8180: A9 BF 85 39 2C 83 C0 A9
8188: 60 8D 41 D1 20 25 D1 20
8190: 06 D1 A9 8D 8D 00 02 A2
8198: 00 4C 2A BF 20 BE BF 20
81A0: 81 D1 4C 2A BF 20 BE BF
81A8: 20 BD D1 4C 2A BF 20 BE
81B0: BF 20 FD DD 4C 2A BF 20
81B8: BE BF 20 B5 EA 4C 2A BF
81C0: 20 BE BF 20 51 DB 2C 81
81C8: C0 60 20 BE BF 4C 84 D0
81D0: 20 BE BF 4C BF D0 2C 81
81D8: C0 4C 3C D4 2C 81 C0 4C
81E0: F2 D4 2C 81 C0 4C 00 E0
81E8: 2C 81 C0 4C D2 D7 2C 81
81F0: C0 4C 65 D8 2C 81 C0 4C
81F8: FB 03 2C 81 C0 4C 62 FA
8200: 2C 81 C0 4C 40 FA AD 72
8208: DD 8D 7A BF AD 73 DD 8D
8210: 7B BF 2C 81 C0 20 FF FF
8218: 4C BE BF 2C 81 C0 20 65
8220: D6 20 BE BF 4C FF D7 2C
8228: 81 C0 20 0C FD 4C BE BF
8230: 2C 81 C0 20 ED FD 4C BE
8238: BF 2C 81 C0 20 95 FE 4C
8240: BE BF 2C 81 C0 20 8B FE
8248: 4C BE BF 2C 81 C0 20 C5
8250: BF 4C BE BF 2C 81 C0 20
8258: C8 BF 2C 83 C0 2C 83 C0
8260: 60 6C 36 00 6C 38 00 51
8268: 9E 84 A8 97 B3 B3 B6 DF
8270: B7 11 BA A8 BF 81 9E FD
8278: AA 5E B6 B5 B7 00 B8 56
8280: BC DC BF 65 D6 D2 D7 0C
8288: FD ED FD 8B FE 95 FE 7F
8290: BF 4C BF 8B BF 94 BF A6
8298: BF 9D BF 58 BF 5E BF 64
82A0: BF 51 9E 1D 4C 34 BF 4C
82A8: 2E BF 4C 12 BF 4C 1B BF
82B0: A9 BF A0 CB 60 EA EA A9
82B8: BF A0 E1 60 EA EA 4C 24
82C0: BF 00 9D 10 79 FE 00 BF
82C8: 09 BF 75 DD 93 DD 60 DD
82D0: 00 DD CB BF B7 9D 01 6B
82D8: 56 9D 0C FC D7 FC D7 52
82E0: BF 46 BF 3A BF 40 BF 5D
82E8: A2 03 4C BC D6 BC A3 0D
82F0: C9 06 90 03 4C C9 D3 8D
82F8: 57 DD 4C D4 DA B1 AA 01
8300: 05 B7 A5 01 60 3B A8 0F
8308: 68 68 A9 00 85 73 85 6F
8310: A9 BF 85 74 85 70 60 C1
8318: AA 06 E1 BF BB E6 BB E7
8320: 6B A5 01 60 09 A9 01 41
8328: B3 AE 01 04 FF AE 03 4C
8330: 7F E6 E4 B7 02 E1 BF EE
8338: B7 02 FB EA BA 9E 03 4C
8340: B8 BF C5 9F 03 4C AF BF
8348: 94 A3 03 4C 6A BF 00 00
```

(die zwei letzten Nullen sind von Bedeutung)

PEEKER Börse

Verkauf Hardware

Fernschreiberinterface am Gameport m. Programm DM 79,– P. Benner, Hubertusstr. 131, 4150 Krefeld

Apple-SUPERANGEBOT

Apple II C Systemeinheit, Monitor und Monitorstand DM 3000,–*

Apple II C Systemeinheit, Mouse, 2. Diskettenlaufwerk, Monitor und Monitorstand DM 3700,–*

Apple Macintosh 128 KB, Monitor, Tastatur, Mouse, Macwrite, Macpaint DM 4500,–*

Apple Macintosh 512 KB Monitor, Tastatur, Mouse, Macwrite, Macpaint DM 5500,–*

Mac Ten 10 MB Platte DM 5000,–*

GEBRAUCHT

Apple Macintosh 128 KB Monitor, Tastatur, Mouse, Macwrite, Macpaint DM 3700,–*

Apple Macintosh 512 KB Monitor, Tastatur, Mouse, Macwrite, Macpaint DM 4500,–*

* Preise incl. MwSt.

Täglich von 9.00 bis 16.00 Uhr Tel. 02 11 / 57 10 01-02

Bonnsystem GmbH
Lueplatz 6, 4000 Düsseldorf 11

Verkaufe Komplettsystem Apple II+ mit: 64K, 80Z, Z80, Centronics-Interface, Monitor 22 MHz, externe RAFI-Halltastatur. 1. Diskettenlaufwerk 35/40 Track, 100% kompatibel, 2. LW TEAC mit 640 KB, zus. mit Controller + angepaßtes DOS, Pascal, CP/M. Programme CP/M + WordStar, Mail-Merge; UCSD-Pascal 1.1; DOS 3.3 mit VisiCalc, VisiDex; Pro-DOS. Gerät 1 Jahr alt, generalüberholt, 6 Mon. Garantie. Preis DM 4900,– A. Schäpers, Tel: 089 / 6 01 41 56.

Apple III + Profile VB DM 4.500,– Sharp MZ 80 B kompl., Floppy, Drucker, viel Software, Schreibsystem (Wordstar), VB DM 3500,– Apple Grafik Tablett + Orig. Software VB DM 800,– TA-Kugelkopfdrucker DIN A 3 seriell + paral. 20Z/sec. VB DM 600,– Tel: 0 26 41 / 14 41

IIe + Imagewriter Monitor III Z80 2 Disk (40/80 Track) 128 KB Joystick Mouse 2SSC orig. AWork + AWriter 100 Disk Softw. 6000,– 3/4 J. alt alles Original Tel: 0 56 73 / 26 17

Imagewriter + Super-S-Card + Hdb. Tel.: 02 41 / 8 63 58 ab 18h

Apple /// u. Apple // 1 Apple /// System 256 KB DM 2330,–; 1 Silentype Drucker DM 235,–; 2 8" Laufwerke mit Controller f. A//e DM 1256,00; 1 Typenraddrucker C-Itho F10/55 parallel DM 2000,–; 1 Handkopierer 3 M DM 50,–; 2 Visicalc /// à DM 160,–; 2 Mail List Manager // /à DM 99,–; 1 Pro Fit Datenbank A// DM 300,–; 2 Visidex Apple //à DM 50,–; 2 Visicalc Apple // à DM 50,–; 1 Pro Fit Datenbank A// DM 400,–; 1 Senior Analyst A// DM 50,–; 1 Profile /// Kit DM 80,–; Täglich von 9.00 bis 18.00 Uhr, Tel.: 08 21 / 51 00 29 CHS Datentechnik GmbH

Apple IIe 128KB, Disk II-+Contr. TAXAN VIS. III, EPSON FX 80+, Joyst (org.), IF: Drucker, RGB; 80Z, Lit.: Handb., Peeker u.a. SW.; Grafik, Spiele, Peeker u.a. Nur kompl. abzug. gegen Höchstgebot, Tel. 06 41 / 7 51 39

Sonderpreis APPLE //e, 2 Disk org. IxCentr., 1xGrappler +, 80Z+64K, Z80 Mon., Softw. u.a. Appleworks, etc., Tel: 0 81 61 / 8 51 71

Microsoft Softcard II original. neuw. Z80B. 6 MHz 64K Speicher. CP/M-80. Handbücher. NP DM 1680,– für DM 1100,– zu verkaufen. Tel. 0 26 33/9 61 47 (ab 17.00 Uhr).

Apple IIe, 128KB, 80Z, Z80, SSC, org. Monitor, org. Laufw. externe Preh Tastatur VB DM 2900. ERPHI Doppel Floppy (1,2MB) Gehäuse, Netzteil u. Controller VB DM 1600 Tel: 0 97 21 / 2 49 34

Apple /// 256 KB mit 2. Laufwerk und Monitor kpl. VB DM 3700,– T: 06 41 / 7 09 53 97 ab 17.00 h: 06 41 / 4 58 51

Apple IIe komp., 128 K RAM 40/80 Zei. 16K Eprom, 16K Lang. Kart, IBM-LOOK, abges. Tasta. *Z80-Karte f. CPM 2 Laufw. je 96 TPI-2×80-Track, VB 2500,– DM, Tel: 0 22 41 / 2 79 73 *Gross/Kleinschreibg.-+Monitor

BASIS 108, 128 KB, Pseudo-floppy, Z80 Seriell + Parallel, 2-Drives, Software: 3 Betriebssysteme, 9 Sprachen, jede Menge Spiele + Utilities + Bücher + Monitor + Tastatur für nur DM 3000,– (NP ca. DM 8000), Tel: 0 62 02/7 23 74

Verkaufe neuwertigen Apple IIe + Apple Monitor II + Apple Disk II mit Controller + 80 Zeichen-64KB Erweiterungskarte (doppelte Auflösung in HGR möglich) für DM 2700,– VHS Tel.: 0 62 34 / 74 58

Apple IIc,+Monitor IIc+Maus, +umfangr. Softw. z.B. Applewriter, Double-hires-tools, Datenbank, Tasc (Basic-Compiler) + umfangr. Apple-Literatur, + alle Peeker mit Sammeldisks. Verk. DM 3000,– Tel.: Bütow (0 42 89) 543

512 Kb Erweiterung für den MAC incl. Einbau und Übernahmegaran. für nur DM 480,– 1Mb auf Anfr. Disketten 31/2 Zoll nur DM 3,90 24 Stunden Service Anfragen Tel: 02 21 / 7 90 29 01

Apple IIc, 2. Laufw. Scribe-Drucker div. Softw. VB DM 3300,– T: 05 61 / 8 04 48 38 od 49 83 43 Hoffmann

Mac Profi Speichererweiterung 128=512KB DM 498,– Baus. 298,– Fa. Schlösser ab 17.00 Uhr Tel: 089 / 98 58 89

IBS AP-22 Z-80H 8 Mhz 64 KB RAM 450,–, Tel. 02 08/75 14 66 ab 18 Uhr

Apple IIe, IIc Mac und Apple-Zubehör – Bei uns sind Sie an der richtigen Adresse. Kostenlos Preisliste anfordern! Welz Electronic, Tel. 04 192 / 46 28

Epson Interfaceumbau f. AWorks DM 20. Mahr, Waldakker 71, 73 Esslingen

Z80B+CPM 3.0u.2.2Ø; Accelerator II + Disk je DM 450,– Tel.: 02 28 / 67 74 23

Verkauf Software

*****STOCKMASTER II*****
Das Apple II-Programm für echte Börsengewinne. Diskette nur DM/SFr. 485,–. Beschreibung 'pe02' anfordern bei: Töngi Computer-Praxis, Aspeltstr. 4, 6500 Mainz. für die Schweiz: Denton Consultants AG, Auwisstr. 17, CH-8127 Forch/Zürich.

Wator-Simulation einer Räuber-Beute-Bez. Erzeugt Tabelle u. Grafik (Monitor); druckt Unter Mat (Grafik), erprobt, komfortabel, Domentation, (Pascal 1.2-Vers. DM 30) Schluckebier, 5805 Breckerfeld, Berliner Str. 21

Software Uhr für Apple II+, e, c, Zeitschaltmöglichkeit Diskette + Anleitung DM 25,– Oecking Tel: Do. 02 31 / 39 19 20

Wg. System-Wechsel günstig abzug. Mac-Software, z.B. Multiplan DM 400,– u.a. Original, neu Tel: 0 61 02 / 1 72 42

APPLE/IIe weg. Systemw. Peeker 84/85 kompl. + Disk 1-10 ohne 3,9 DM 150,– Pro-DOS-Edit. 1.0 +MMU 2.0 + FSS1, 2, 10 DM 100,– Turbo-Pasc. 3.0 DM 140,– alles org. mit Anl. 0 77 65 / 12 87

Pirate Defence 2.0 Kopierschutz. Bombensicher und preiswert zugleich. Für ProDOS, DOS, DIVERSI u.a. Info (50 Pf.) bei C. Bregler, Tulpenstr.2, 7519 Eppingen. Händleranfragen erwünscht!

Original Sharp CPM + Pascal/MT+, NP DM 2000,– VB DM 600,–. Tel: 0 26 41 / 14 41

Apple: Public Domain: Volume DM 15 Games, Schach, Mathe, Graphic etc. Derw. Lehrerpro., „Mini-Logo". Gratisinfo: Fa. Waltraud Muhle, Waldwinkel 3, 2105 Seevetal 3

GOÄ-Privatliquidation Praxiserprobt, Preiswert und Schnell mit Mahnungsprg. und Ausdruck eines Bank-Zahlungsbeleges G. Gross-,Säckinger Str. 16, 7887 Laufenburg,Tel: 0 77 63 / 55 27

PRINT-SHOP Grafiken aus oder in HGR kopieren? Disk DM 50,– B. Rüter, Rahdener Str. 65, 4955 Hille.

Imagewriter-Zeichensatzgenerator DM 30,– Info unter: 0 23 07 / 7 40 75

Verschiedenes

APPLE REPARATUREN
(auch compatible M-boards, z.B. **Atlas, Arca, CES, Datastar, Dipa, Lasar, Mewa, PC-48 + 64, Plato, Radix, o. ae.) sowie Zusatzkarten und Disk-Drives** führt unser Spezialistenteam mit mehr als 5-jähriger Kunden- und Reparatur-Dienst-Erfahrung, garantiert zuverlässig und besonders kostengünstig aus.
Bitte genaue Fehlerangabe sowie Tel. Nr. für evtl. Rückfragen nicht vergessen.
Auf Wunsch Kostenvoranschlag.
aaa-electronic gmbh
Habsburgerstr. 134, 7800 Freiburg, Tel. 0761/276864, Tx. 772642aaad

Auth. Roulette-Permanenzen div. Casinos auf DOS-Format. Ihre Softwareideen werden schnell und günstig vom Insider realisiert. sogleich anrufen! Tel: 069 / 44 46 90

Für Ihre Unterlagen

Abonnement bestellt

am: _____

Vertrauensgarantie:
Ich habe davon Kenntnis genommen, daß ich die Bestellung schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1 innerhalb von 7 Tagen widerrufen kann. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels).

Peeker
Leserservice
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Abo-Karte

Ja, ich möchte **Peeker** abonnieren.
Liefern Sie mir **Peeker** ab Ausgabe ..... zum Jahresbezugspreis von DM 72,– (Inland) inkl. MwSt. Die Lieferung erfolgt frei Haus. Porto, Verpakkung und Zustellgebühren übernimmt der Verlag. Der Jahresbezugspreis für das Ausland beträgt DM 72,– inkl. MwSt., zzgl. DM 18,– Versandspesen.

Ich wünsche jährliche Berechnung durch:

☐ Verlagsrechnung ☐ Abbuchung von meinem Bank- bzw. Postscheckkonto

Bank / PschA _____

Bankleitzahl _____ Kto.-Nr. _____

Datum _____ Unterschrift _____

Für Ihre Unterlagen

Folgende Bücher bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Versandbuchhandlung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Buch-Karte

Bitte senden Sie mir gegen Rechnung folgende Bücher:

Menge	Autor, Titel	à DM	gesamt DM

Datum _____ Unterschrift _____

Für Ihre Unterlagen

Folgende Disketten
und Programme bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Softwareabteilung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Software-Karte

Bitte senden Sie mir gegen Rechnung folgende Disketten:

☐ Peeker-Sammeldiskette, einzeln
Disk# _____, Disk# _____
Disk# _____, Disk# _____
Preis je Disk DM 28,– (einzeln)

☐ Peeker-Sammeldiskette,
im Fortsetzungsbezug
ab Disk # _____
(Mindestbezug 6 Disketten)
Preis je Disk DM 20,–
Neben DOS-Disketten auch liefern
☐ CP/M ja ☐ CP/M nein
☐ Pascal ja ☐ Pascal nein

☐ Apple DOS 3.3, Begleitdiskette, DM 28,–
☐ ProDOS, Band 1, Begleitdiskette, DM 28,–
☐ ProDOS, Band 2, Begleitdiskette, DM 28,–
☐ Apple Assembler, Begleitdiskette, DM 28,–
☐ ProDOS-Editor 1.0, Programm, DM 98,–
☐ MMU 2.0, Programm, DM 98,–
☐ INPUT 2.0, Programm, DM 98,–
☐ Softbreaker 1.0, Programm, DM 48,–
☐ DB-Meister, Programm, DM 290,–
☐ Superplot, Programm, DM 48,–
☐ Superquick, Programm, DM 48,–
☐ Turtle Graphics, Programm, DM 98,–

Datum _____ Unterschrift _____

Abo-Karte

Name

Firma

Abteilung

Straße

PLZ/Ort

Vertrauensgarantie:
Ich habe davon Kenntnis genommen, daß ich die Bestellung schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag, Postfach 10 28 69, 6900 Heidelberg 1 innerhalb von 14 Tagen widerrufen kann. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels).

Datum

Unterschrift

Verlagshinweis:
Das Abonnement verlängert sich zu den jeweils gültigen Bedingungen um ein Jahr, wenn es nicht 2 Monate vor Jahresende schriftlich gekündigt wird.

Buch-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

Software-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

Peeker
Leserservice
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Peeker
Buchabteilung
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Peeker
Softwareabteilung
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

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

„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.
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.).
Gerätevoraussetzung: Apple IIe oder IIc; ferner Apple II+ im 40-Zeichenmodus

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

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.

Gerätevoraussetzung: Apple IIe mit 64K-Karte oder IIc

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

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.
Mit Softbreaker unterbrochene Programme werden komplett, d. h. die ganzen 64K einschließlich Language Card, in nur ca. 11 Sekunden auf einer formatierten Diskette gesichert.
Gerätevoraussetzung: Apple IIe mit 64K-Karte

Hüthig Software Service,
Postfach 10 28 69, D-6900 Heidelberg

Matrizenrechnung

in der betriebswirtschaftlichen Praxis

von Dipl.-Betriebswirt Willy Holtkamp

1. Wozu Matrizenrechnung?

Wenn es gilt, betriebliche Abläufe in ihrer Wirkung darzustellen, greift der Unternehmenscontroller gern zu Tabellenprogrammen wie Visicalc oder er wendet die Matrizenrechnung an.
Leider besitzt der Applesoft-Interpreter nicht die bekannten MAT-Befehle. Deshalb muß man hier FOR-NEXT-Schleifen anwenden, die viel Programmierarbeit erfordern und Speicherplatz wegnehmen. Die langsamere Verarbeitung kann zum Teil durch den Einsatz eines Compilers und der Accelerator-Karte wettgemacht werden.
Ein effizientes Arbeiten mit Matrizen gewährleistet das **Ampersoft-Paket** von Microsparc Inc., das zum Preis von ca. 200,- DM bei einschlägigen Importeuren erhältlich ist. Eine preiswerte Alternative (ca. $20.00) wird von **LRS-Systems** angeboten (deutscher Importeur nicht bekannt). Um die Vorzüge der beiden Programme zu demonstrieren, habe ich zwei Beispiele gewählt.

1.1. Beispiel Nr. 1

Das Beispiel Nr. 1 ist der „Einführung in die lineare Planungsrechnug mit Algol- und Fortran-Programmen“ von Gerhard Niemeyer entnommen. Ich habe zu Demonstrationszwecken zwei Applesoft-Versionen erstellt. Das erste BASIC-Programm **MATRIX** behandelt Vektoren – also eindimensionale Felder – wie Matrizen, während das Programm **VEKTOR** Vektoren in der gewohnten Vektorform definiert. Ampersoft verlangt die Definition von mindestens zweidimensionalen Feldern, während das LRS-Programm die übliche Behandlung von Vektoren vorsieht. Außerdem ist zu beachten, daß bei Ampersoft Spalte 0 und Zeile 0 freibleiben müssen. Dagegen erfordert die Anwendung von LRS die Definition von Feldern einschließlich Spalte 0 und Zeile 0.

1.2. Erstellung der Matrizen

Der Controller Kraus hat in seinem Produktionsbetrieb folgendes Planungsmodell aufgestellt:

1. Der Produktionsplan für die Produkte P1 und P2 sieht für den Zeitraum Oktober bis Dezember wie folgt aus:

	P1	P2
Oktober	10	15
November	15	30
Dezember	20	40

A(3,2) [Monat, Stück]

Er bezeichnet den Produktionsplan mit dem Namen Matrix A und stellt fest, daß die Dimension von Mat A 3 Zeilen und 2 Spalten beträgt.

2. Die Produkte P1 und P2 werden aus den Einzelteilen a, b, c und d zusammengesetzt. Die nachfolgende Tabelle gibt Auskunft über die Kombination, die für jeweils 1 Stück notwendig ist:

	a	b	c	d
P1	2	1	3	0
P2	4	6	0	2

B(2,4) [Produkt, Einzelteil]

Die Stückliste für die Endprodukte P1 und P2 wird von Kraus Matrix B genannt und weist 2 Zeilen und 4 Spalten auf.

3. Zur Herstellung der Einzelteile a-d benötigt der Betrieb die Materialarten A, B, C, und zwar in folgenden Mengen:

	A	B	C
Einzelteil a	0,2	0,3	0
Einzelteil b	0	0	0,2
Einzelteil c	0,8	0,2	0,4
Einzelteil d	0,4	0,2	0

C(4,3) [Einzelteil, Materialmenge]

Die Matrix C gibt also den Materialbedarf je Stück der Einzelteile a-d an und besitzt einen Umfang von 4 Zeilen und 3 Spalten.

4. Zur Erstellung der Einzelteile braucht der Betrieb nicht nur Material, sondern auch Arbeitskräfte. Deren Zeitbedarf in Minuten für die einzelnen Arbeitsgänge hat Controller Kraus in der Matrix D festgehalten:

	D	E	F	G	H
Einzelteil a	10	8	0	0	5
Einzelteil b	4	1	3	2	0
Einzelteil c	0	0	4	1	6
Einzelteil d	2	5	2	1	3

D(4,5) [Einzelteil, Zeit]

5. Die Materialkostenmatrix F besitzt 3 Zeilen und 1 Spalte und hat folgendes Aussehen:

Material A	10,00
Material B	8,00
Material C	15.00

F(3,1) [Material, Preis]

6. Die Arbeitskosten je Minute für die Arbeitsgänge D, E, F, G und H hat Kraus in der Matrix G zusammengefaßt. Die Matrix G hat den Umfang von 5 Zeilen und 1 Spalte.

Arbeitsgang D	0,08
Arbeitsgang E	0,10
Arbeitsgang F	0,20
Arbeitsgang G	0,15
Arbeitsgang H	0,10

G(5,1) [Arbeitsgang, Kosten]

1.3. Die Kostenermittlung

Kraus stellt nun folgende Überlegungen an: Er benötigt die unten aufgeführten Daten, um die Gesamtkosten des Zeitraums Oktober-Dezember ermitteln zu können.

a) Bedarf an Einzelteilen
b) Bedarf an den einzelnen Materialarten
c) Bedarf an Zeit für die einzelnen Arbeitsgänge
d) Materialkosten
e) Arbeitskosten
f) Materialkosten je Stück der Einzelteile
g) Arbeitskosten je Stück der Einzelteile

Diese Informationen führen dann zu den Kosten für 3 Monate, wie aus der **Ermittlung der Gesamtkosten** zu erkennen ist. Die endgültige Lösung kann dann entsprechend **Lösung 1** oder **Lösung 2** errechnet werden.
Diese betriebswirtschaftliche Lösung ist noch einmal in der **Tabelle 1** zusammengestellt. Die **Tabelle 2** gibt eine Übersicht für die Berechnungen mit Hilfe des Matrizen- und Vektor-Programms.

2. Ampersoft

Die Entscheidung zum Kauf von Ampersoft löste meine bisherigen Probleme mit einem Schlag! Jetzt stehen mir 46333 Bytes RAM zur Verfügung, so daß auch große Matrizen bewältigt werden können (siehe später folgendes Beispiel Nr. 2).
In Verbindung mit der Accelerator-Karte ist Ampersoft jeder mir bekannten Lösung hinsichtlich Verarbeitungsgeschwindigkeit überlegen.
Das Programm AMPERSOFT1 (die im folgenden besprochenen Programme sind nur auf der Peeker-Sammeldiskette enthalten) macht deutlich, wie einfach die Programmierung mittels Ampersoft zu handhaben ist. Dazu trägt auch die komfortable Print-Using-Routine von Ampersoft bei, die in AMPERSOFT2 zu ersehen ist.
Die Erstellung von Ampersoft-Dateien und das Einlesen dieser Dateien verdeutlichen die Programme AMPERSOFT3 und AMPERSOFT4.

2.1. Beispiel Nr. 2

Das erste Beispiel vernachlässigte bewußt die Dateibehandlung. Nun will der Controller Kraus sechs Matrizen als Ampersoft-Files auf die Diskette schreiben und anschließend diese Matrizen wieder einlesen und gemäß dem folgenden Schema miteinander verknüpfen. Dabei wählt er bewußt eine etwas umständliche Regelung, um möglichst viele Diskettenzugriffe und Memory-Clear-Vorgänge abwickeln zu können.
Schema:

1. Matrix A mal Matrix B ergibt den File „G1815“
2. Matrix C plus Matrix D ergibt den File „R1256“
3. Matrix E mal Matrix F ergibt den File „S3434“
4. Matrix H mal Matrix K ergibt den File „T3434“
5. Matrix P mal Matrix Q ergibt den File „W4040“
6. Matrix S plus Matrix T ergibt den File „X3434“

Aus dem Programm AMPERSOFT5 ist zu ersehen, wie mächtig Ampersoft ist. Es bereitet keinerlei Probleme, die sechs Files auf einmal abzuspeichern. Das geht bei Applesoft sonst nur, wenn man einen DOS-Mover zur Verfügung hat, wie er z.B. beim Kauf einer Saturn-128K-Karte mitgeliefert wird.
AMPERSOFT6 stellt das Ampersoft-Programm dar, das die benötigten Files einliest und dann gemäß dem erwähnten Schema verknüpft. Ohne die Benutzung der Accelerator-Karte braucht der Apple 6 Minuten und 21 Sekunden; mit der Accelerator-Karte kann die Verarbeitungszeit auf 3 Minuten und 21 Sekunden gesenkt werden.
Die zweite Version (AMPERSOFT7) nutzt den großen Arbeitsspeicher aus, indem statt 3 Files maximal 6 Files im Speicher des Apple gehalten werden. Auf die Verarbeitungsgeschwindigkeit hat das praktisch keinen Einfluß. Version 3 (AMPERSOFT8) addiert die Matrizen S und T zur Matrix X und löscht dann den Speicher. Anschließend werden dann die Matrizen P und Q zur neuen Matrix „W4040“ verknüpft und abgelegt. Der optimierte Programmablauf weist die gleiche Verarbeitungsdauer auf wie die Versionen 1 und 2.

Ermittlung der Gesamtkosten

Zu a) Mat A(3,2) · Mat B(2,4) = Mat Ergebnis (3,4)

$$\begin{matrix}\text{Okt.}\\ \text{Nov.}\\ \text{Dez.}\end{matrix}\begin{pmatrix}10 & 15\\ 15 & 30\\ 20 & 40\end{pmatrix} \cdot \begin{matrix}\\ \text{P1}\\ \text{P2}\end{matrix}\begin{pmatrix}2 & 1 & 3 & 0\\ 4 & 6 & 0 & 2\end{pmatrix} = \begin{matrix}\text{a} & \text{b} & \text{c} & \text{d}\end{matrix}\begin{pmatrix}80 & 100 & 30 & 30\\ 150 & 195 & 45 & 60\\ 200 & 260 & 60 & 80\end{pmatrix}$$

Zu b) Mat Ergebnis (3,4) · Mat C(4,3) = Mat Materialbedarf (3,3)

$$\begin{matrix}\text{Okt.}\\ \text{Nov.}\\ \text{Dez.}\end{matrix}\begin{pmatrix}80 & 100 & 30 & 30\\ 150 & 195 & 45 & 60\\ 200 & 260 & 60 & 80\end{pmatrix} \cdot \begin{matrix}\text{Einz.}\ \text{a}\\ \text{Einz.}\ \text{b}\\ \text{Einz.}\ \text{c}\\ \text{Einz.}\ \text{d}\end{matrix}\begin{pmatrix}0{,}2 & 0{,}3 & 0\\ 0 & 0 & 0{,}2\\ 0{,}8 & 0{,}2 & 0{,}4\\ 0{,}4 & 0{,}2 & 0\end{pmatrix} = \begin{matrix}\text{A} & \text{B} & \text{C}\end{matrix}\begin{pmatrix}52 & 36 & 32\\ 90 & 66 & 57\\ 120 & 88 & 76\end{pmatrix}$$

Zu c) Mat Ergebnis (3,4) · Mat D(4,5) = Mat Arbeitszeitbedarf (3,5)

$$\begin{matrix}\text{Okt.}\\ \text{Nov.}\\ \text{Dez.}\end{matrix}\begin{pmatrix}80 & 100 & 30 & 30\\ 150 & 195 & 45 & 60\\ 200 & 260 & 60 & 80\end{pmatrix} \cdot \begin{matrix}\text{Einz.}\ \text{a}\\ \text{Einz.}\ \text{b}\\ \text{Einz.}\ \text{c}\\ \text{Einz.}\ \text{d}\end{matrix}\begin{pmatrix}10 & 8 & 0 & 0 & 5\\ 4 & 1 & 3 & 2 & 0\\ 0 & 0 & 4 & 1 & 6\\ 2 & 5 & 2 & 1 & 3\end{pmatrix} =$$

3. LRS-Systems

In einer Ausgabe der Zeitschrift „Nibble" fand ich eine Kleinanzeige der Firma LRS-Systems. Ich schickte $20.00 in einem Einschreibebrief und erhielt nach kurzer Zeit eine kopierbare Diskette und ein vorbildlich gestaltetes Anwender-Handbuch. Nach dem Studium dieses Handbuches war es für mich kein Problem, die nachfolgenden Programme zu schreiben. Das Programm LRS1 kann mit AMPERSOFT6 verglichen werden. Ohne Anwendung der Accelerator-Karte benötigt LRS 17 Minuten und 9 Sekunden (Ampersoft: 6 Min., 21 Sek.). Steht eine Accelerator-Karte zur Verfügung, so kann die Verarbeitsungszeit auf ca. 7 Minuten und 39 Sekunden gesenkt werden (Ampersoft: 3 Min., 21 Sek.). LRS2 zeigt die direkte Lösung und entspricht in etwa der Ampersoft-Lösung in AMPERSOFT8. Das LRS-Programm braucht 6 Minuten und 38 Sekunden, wenn die Accelerator-Karte benutzt wird.

Setzt man nun noch Diversi-DOS ein, so beträgt die Dauer der Bearbeitung nur noch 5 Minuten und 38 Sekunden. Das sind immerhin noch 2 Minuten und 17 Sekunden mehr als bei Benutzung von Ampersoft.

4. Vergleich Ampersoft mit LRS-Systems

4.1. Gemeinsamkeiten

- Matrixoperationen (IDN, INV, TRN usw.)
- Löschen von Feldern
- Ermittlung der Determinante

4.2. Vorteile von Ampersoft gegenüber LRS

- DOS-Mover bringt Arbeitsspeicher von 46K netto
- Komfortables Print-Using
- Sortieren von Real-, Integer- und String-Feldern
- Schnelle Garbage Collection
- Schnelles Speichern und Laden von Files
- File-Behandlung ist mit einem einzigen Befehl zu programmieren
- Allgemein sehr schnelle Verarbeitung

4.3. Vorteile von LRS gegenüber Ampersoft

- Geringer Preis
- Programm kann entweder ab $8AA6 oder ab $0800 geladen werden. (Das Laden ab $0800 gestattet ein optimales Ausnutzen des Arbeitsspeichers, wenn z.B. Hires-Grafik vorgesehen ist.)
- Ausgezeichnetes Handbuch

	D	E	F	G	H
Okt.	1260	890	480	260	670
Nov.	2400	1695	885	495	1200
Dez.	3200	2260	1180	660	1600

Zu d) Mat Materialbedarf (3,3) · Mat F (3,1) = Mat Materialkosten (3,1)

$$\begin{pmatrix} 52 & 36 & 32 \\ 90 & 66 & 57 \\ 120 & 88 & 76 \end{pmatrix} \cdot \begin{matrix} \text{Material A} \\ \text{Material B} \\ \text{Material C} \end{matrix} \begin{pmatrix} 10 \\ 8 \\ 15 \end{pmatrix} = \begin{pmatrix} 1288 \\ 2283 \\ 3044 \end{pmatrix}$$

Zu e) Mat Arbeitszeitbedarf (3,5) · Mat G (5,1) = Mat Arbeitskosten (3,1)

$$\begin{matrix} & D & E & F & G & H \\ \text{Okt.} \\ \text{Nov.} \\ \text{Dez.} \end{matrix} \begin{pmatrix} 1260 & 890 & 480 & 260 & 670 \\ 2400 & 1695 & 885 & 495 & 1200 \\ 3200 & 2260 & 1180 & 660 & 1600 \end{pmatrix} \cdot \begin{matrix} \text{Arbeitsgang D} \\ \text{Arbeitsgang E} \\ \text{Arbeitsgang F} \\ \text{Arbeitsgang G} \\ \text{Arbeitsgang H} \end{matrix} \begin{pmatrix} 0{,}08 \\ 0{,}10 \\ 0{,}20 \\ 0{,}15 \\ 0{,}10 \end{pmatrix} = \begin{pmatrix} 391{,}80 \\ 732{,}75 \\ 977{,}00 \end{pmatrix}$$

Zu f) Mat C (4,3) · Mat F (3,1) = Mat Einzteilkosten/M (4,1)

$$\begin{matrix} \text{Einz. a} \\ \text{Einz. b} \\ \text{Einz. c} \\ \text{Einz. d} \end{matrix} \begin{pmatrix} 0{,}2 & 0{,}3 & 0 \\ 0 & 0 & 0{,}2 \\ 0{,}8 & 0{,}2 & 0{,}4 \\ 0{,}4 & 0{,}2 & 0 \end{pmatrix} \cdot \begin{matrix} \text{Material A} \\ \text{Material B} \\ \text{Material C} \end{matrix} \begin{pmatrix} 10 \\ 8 \\ 15 \end{pmatrix} = \begin{pmatrix} 4{,}40 \\ 3{,}00 \\ 15{,}60 \\ 5{,}60 \end{pmatrix}$$

Zu g) Mat D (4,5) · Mat G (5,1) = Mat Einzteilkosten/A (4,1)

$$\begin{matrix} \text{Einz. a} \\ \text{Einz. b} \\ \text{Einz. c} \\ \text{Einz. d} \end{matrix} \begin{pmatrix} 10 & 8 & 0 & 0 & 5 \\ 4 & 1 & 3 & 2 & 0 \\ 0 & 0 & 4 & 1 & 6 \\ 2 & 5 & 2 & 1 & 3 \end{pmatrix} \cdot \begin{matrix} \text{Arbeitsgang D} \\ \text{Arbeitsgang E} \\ \text{Arbeitsgang F} \\ \text{Arbeitsgang G} \\ \text{Arbeitsgang H} \end{matrix} \begin{pmatrix} 0{,}08 \\ 0{,}10 \\ 0{,}20 \\ 0{,}15 \\ 0{,}10 \end{pmatrix} = \begin{pmatrix} 2{,}10 \\ 1{,}32 \\ 1{,}55 \\ 1{,}51 \end{pmatrix}$$

Lösung 1:
Mat Materialkosten (3,1) + Mat Arbeitskosten (3,1)
= Mat Gesamtkosten (3,1)

$$\begin{matrix} \text{Okt.} \\ \text{Nov.} \\ \text{Dez.} \end{matrix} \begin{pmatrix} 1288{,}00\ \text{DM} \\ 2283{,}00\ \text{DM} \\ 3044{,}00\ \text{DM} \end{pmatrix} + \begin{pmatrix} 391{,}80\ \text{DM} \\ 732{,}75\ \text{DM} \\ 977{,}00\ \text{DM} \end{pmatrix} = \begin{pmatrix} 1679{,}80\ \text{DM} \\ 3015{,}75\ \text{DM} \\ 4021{,}00\ \text{DM} \end{pmatrix}$$

Lösung 2:
Mat Ergebnis (3,4) · {Mat Einzteilkosten/M + Mat Einzteilkosten/A}
= Mat Gesamtkosten

$$\begin{matrix} \text{Okt.} \\ \text{Nov.} \\ \text{Dez.} \end{matrix} \begin{pmatrix} 80 & 100 & 30 & 30 \\ 150 & 195 & 45 & 60 \\ 200 & 260 & 60 & 80 \end{pmatrix} \cdot \left[\begin{matrix} \text{Einz. a} \\ \text{Einz. b} \\ \text{Einz. c} \\ \text{Einz. d} \end{matrix} \begin{pmatrix} 4{,}40 \\ 3{,}00 \\ 15{,}60 \\ 5{,}60 \end{pmatrix} + \begin{matrix} \text{Einz. a} \\ \text{Einz. b} \\ \text{Einz. c} \\ \text{Einz. d} \end{matrix} \begin{pmatrix} 2{,}10 \\ 1{,}32 \\ 1{,}55 \\ 1{,}51 \end{pmatrix} \right] =$$

$$\begin{matrix} \text{Okt.} \\ \text{Nov.} \\ \text{Dez.} \end{matrix} \begin{pmatrix} 80 & 100 & 30 & 30 \\ 150 & 195 & 45 & 60 \\ 200 & 260 & 60 & 80 \end{pmatrix} \cdot \begin{matrix} \text{Einz. a} \\ \text{Einz. b} \\ \text{Einz. c} \\ \text{Einz. d} \end{matrix} \begin{pmatrix} 6{,}50 \\ 4{,}32 \\ 17{,}15 \\ 7{,}11 \end{pmatrix} = \begin{pmatrix} 1679{,}80\ \text{DM} \\ 3015{,}75\ \text{DM} \\ 4021{,}00\ \text{DM} \end{pmatrix}$$

Tabelle 1

Kostensituation im 4. Quartal

Monat	Materialkosten	Arbeitskosten	Gesamtkosten
Oktober:	1288,00	391,80	1679,80
November:	2283,00	732,75	3015,75
Dezember:	3044,00	977,00	4021,00

5. Fazit

Mit dem Erscheinen der beschriebenen Hilfsprogramme wurde eine Lücke bei der Anwendungsmöglichkeit des Apple II geschlossen. Jetzt kann der Unternehmenscontroller auch umfangreiche und schwierige Simulationen in relativ kurzer Zeit durchführen.

Kurzhinweise

1. Zweck: Demonstrierung der Matrizenrechnung in der betriebswirtschaftlichen Praxis mit Hilfe der Programm-Pakete Ampersoft und LSR
2. Konfiguration:
Apple II+, IIe oder IIc;
DOS 3.3; Ampersoft oder LSR müssen natürlich vorher von Ihrer eigenen Originaldiskette geladen werden!
3. Test:
RUN MATRIX
RUN VEKTOR
(es erfolgt keine Ausgabe der Ergebnisse)
4. Sammeldisk:
MATRIX
(Applesoft-Programm zur Berechnung von Vektoren als Matrix)
VEKTOR
(Applesoft-Programm zur Vektor-Berechnung)
AMPERSOFT1
(Ampersoft als Alternative zu MATRIX und VEKTOR)
AMPERSOFT2
(Ampersoft-Lösung: Ausdruck der Ergebnisse)
AMPERSOFT3
(Beispiel: Erstellen von Ampersoft-Files)
AMPERSOFT4
(Beispiel: Einlesen von Ampersoft-Files)
AMPERSOFT5
(Beispiel Nr. 2: Erstellen von 6 großen Ampersoft-Files)
AMPERSOFT6
(Beispiel Nr. 2: Ampersoft-Lösung, Version 1)
AMPERSOFT7
(Beispiel Nr. 2: Ampersoft-Lösung, Version 2)
AMPERSOFT8
(Beispiel Nr. 2: Ampersoft-Lösung, Version 3, direkt)
LRS1
(Beispiel Nr. 2: LRS-System, ähnlich Version 1)
LSR2
(Beispiel Nr. 2: LSR-System, ähnlich Version 3)

Weitere Kosteninformationen
Der Bedarf an Einzelteilen beträgt:

	A	B	C	D
Oktober:	80	100	30	30
November:	150	195	45	60
Dezember:	200	260	60	80

Der Bedarf an den Materialarten A, B, C je Monat beträgt:

	A	B	C
Oktober:	52	36	32
November:	90	66	57
Dezember:	120	88	76

Der Arbeitszeitbedarf für die Arbeitsgänge D-H beträgt:

	D	E	F	G	H
Oktober:	1260	890	480	260	670
November:	2400	1695	885	495	1200
Dezember:	3200	2260	1180	660	1600

Die Einzelteil-Stückkosten hinsichtlich Material- u. Arbeitskosten betragen:

	Stückkosten (Material)	Arbeitskosten/Stück
Einzelteil a	4,40	2,10
Einzelteil b	3,00	1,32
Einzelteil c	15,60	1,55
Einzelteil d	5,60	1,51

Die gesamten Einzelteil-Stückkosten betragen:

Einzelteil a	=	6,50
Einzelteil b	=	4,32
Einzelteil c	=	17,15
Einzelteil d	=	7,11

Der Bedarf an Einzelteilen (E(I,J) · Summe Einzelteil-Stückkosten (T(I)) ergibt die nachstehend aufgeführten Gesamtkosten/Monat:

Oktober	=	1679,80
November	=	3015,75
Dezember	=	4021,00

Tabelle 2

Berechnungsübersicht für Matrix-Beispiel

Feldbezeichnung:	Beschreibung der Matrix:
Mat A(3,2)	Produktionsplan für das 4. Quartal
Mat B(2,4)	Einzelteile a–d
Mat C(4,3)	Materialbedarf für die Einzelteile a–d
Mat D(4,5)	Zeitbedarf für die Einzelteile a–d
Mat F(3,1)	Materialkosten für die Materialien A–C
Mat G(5,1)	Arbeitskosten für die Arbeitsgänge D–H
Mat E(3,4)	Mat A · Mat B = Mat Ergebnis
Mat H(3,3)	Mat Ergebnis · Mat C = Mat Materialbedarf
Mat M(3,5)	Mat Ergebnis · Mat D = Mat Arbeitszeitbedarf
Mat N(3,1)	Mat Materialbedarf · Mat F = Mat Materialkosten
Mat P(3,1)	Mat Arbeitszeitbedarf · Mat G = Mat Arbeitskosten
Mat Q(4,1)	Mat C · Mat F = Mat Einzteilkosten/M
Mat R(4,1)	Mat D · Mat G = Mat Einzteilkosten/A
Mat S(3,1)	Mat Materialkosten + Mat Arbeitskosten = Mat Gesamtkosten
Mat T(4,1)	Mat Einzteilkosten/M + Mat Einzteilkosten/A = Mat Gesamteinzteilkosten
Mat V(3,1)	Mat Ergebnis · Mat Gesamteinzteilkosten = Mat Gesamtkosten

MATRIX

```
1000 REM Matrizenbeispiel aus Niemeyer
1010 REM Programm MATRIX
1020 REM von W. Holtkamp
1030 DIM A(3,2),B(2,4),C(4,3),D(4,5),F(3,1),
     G(5,1)
1040 DIM E(3,4),H(3,3),M(3,5),N(3,1),P(3,1),
     Q(4,1),R(4,1),S(3,1),T(4,1),V(3,1)
1050 REM Eingabedaten
1060 REM A(I,J)
1070 DATA 10,15,15,30,20,40
1080 REM B(I,J)
1090 DATA 2,1,3,0,4,6,0,2
1100 REM C(I,J)
1110 DATA .2,.3,0,0,0,.2,.8,.2,.4,.4,.2,0
1120 REM D(I,J)
1130 DATA 10,8,0,0,5,4,1,3,2,0,0,0,4,1,6,2,
     5,2,1,3
1140 REM F(I,J)
1150 DATA 10,8,15
1160 REM G(I,J)
1170 DATA .08,.1,.2,.15,.1
1180 REM Matrizen einlesen
1190 FOR I = 1 TO 3: FOR J = 1 TO 2
1200 READ A(I,J)
1210 NEXT J,I
1220 FOR I = 1 TO 2: FOR J = 1 TO 4
1230 READ B(I,J)
1240 NEXT J,I
1250 FOR I = 1 TO 4: FOR J = 1 TO 3
1260 READ C(I,J)
1270 NEXT J,I
1280 FOR I = 1 TO 4: FOR J = 1 TO 5
1290 READ D(I,J)
1300 NEXT J,I
1310 FOR I = 1 TO 3: FOR J = 1 TO 1
1320 READ F(I,J)
1330 NEXT J,I
1340 FOR I = 1 TO 5: FOR J = 1 TO 1
1350 READ G(I,J)
1360 NEXT J,I
1370 REM Ergebnisse berechnen
1380 FOR I = 1 TO 3
1390 FOR J = 1 TO 4
1400 EI = 0
1410 FOR K = 1 TO 2
1420 EI = EI + A(I,K) * B(K,J)
1430 E(I,J) = EI
1440 NEXT K,J,I
1450 FOR I = 1 TO 3
1460 FOR J = 1 TO 3
1470 HI = 0
1480 FOR K = 1 TO 4
1490 HI = HI + E(I,K) * C(K,J)
1500 H(I,J) = HI
1510 NEXT K,J,I
1520 FOR I = 1 TO 3
1530 FOR J = 1 TO 5
1540 MI = 0
1550 FOR K = 1 TO 4
1560 MI = MI + E(I,K) * D(K,J)
1570 M(I,J) = MI
1580 NEXT K,J,I
1590 FOR I = 1 TO 3
1600 FOR J = 1 TO 1
1610 NI = 0
1620 FOR K = 1 TO 3
1630 NI = NI + H(I,K) * F(K,J)
1640 N(I,J) = NI
1650 NEXT K,J,I
1660 FOR I = 1 TO 3
1670 FOR J = 1 TO 1
1680 PJ = 0
1690 FOR K = 1 TO 5
1700 PJ = PJ + M(I,K) * G(K,J)
1710 P(I,J) = PJ
1720 NEXT K,J,I
1730 FOR I = 1 TO 4
1740 FOR J = 1 TO 1
1750 QI = 0
1760 FOR K = 1 TO 3
1770 QI = QI + C(I,K) * F(K,J)
1780 Q(I,J) = QI
1790 NEXT K,J,I
1800 FOR I = 1 TO 4
1810 FOR J = 1 TO 1
1820 RI = 0
1830 FOR K = 1 TO 5
1840 RI = RI + D(I,K) * G(K,J)
1850 R(I,J) = RI
1860 NEXT K,J,I
1870 FOR I = 1 TO 3
1880 FOR J = 1 TO 1
1890 S(I,J) = N(I,J) + P(I,J)
1900 NEXT J,I
1910 FOR I = 1 TO 4
1920 FOR J = 1 TO 1
1930 T(I,J) = Q(I,J) + R(I,J)
1940 NEXT J,I
1950 FOR I = 1 TO 3
1960 FOR J = 1 TO 1
1970 VJ = 0
1980 FOR K = 1 TO 4
1990 VJ = VJ + E(I,K) * T(K,J)
2000 V(I,J) = VJ
2010 NEXT K,J,I
2020 REM Ergebnis-Ausdruck ab Zeile 2030
```

VEKTOR

```
1000 REM Matrizenbeispiel aus Niemeyer
1010 REM Programm VEKTOR
1020 REM von W. Holtkamp
1030 DIM A(3,2),B(2,4),C(4,3),D(4,5),F(3),
     G(5)
1040 DIM E(3,4),H(3,3),M(3,5),N(3),P(3),
     Q(4),R(4),S(3),T(4),V(3)
1050 REM Eingabedaten
1060 REM A(I,J)
1070 DATA 10,15,15,30,20,40
1080 REM B(I,J)
1090 DATA 2,1,3,0,4,6,0,2
1100 REM C(I,J)
1110 DATA .2,.3,0,0,0,.2,.8,.2,.4,.4,.2,0
1120 REM D(I,J)
1130 DATA 10,8,0,0,5,4,1,3,2,0,0,0,4,1,6,
     2,5,2,1,3
1140 REM F(J)
1150 DATA 10,8,15
1160 REM G(J)
1170 DATA .08,.1,.2,.15,.1
1180 REM Matrizen/Vektoren einlesen
1190 FOR I = 1 TO 3: FOR J = 1 TO 2
1200 READ A(I,J)
1210 NEXT J,I
1220 FOR I = 1 TO 2: FOR J = 1 TO 4
1230 READ B(I,J)
1240 NEXT J,I
1250 FOR I = 1 TO 4: FOR J = 1 TO 3
1260 READ C(I,J)
1270 NEXT J,I
1280 FOR I = 1 TO 4: FOR J = 1 TO 5
1290 READ D(I,J)
1300 NEXT J,I
1310 FOR J = 1 TO 3
1320 READ F(J)
1330 NEXT J
1340 FOR J = 1 TO 5
1350 READ G(J)
1360 NEXT J
1370 REM Ergebnisse ermitteln
1380 FOR I = 1 TO 3
1390 FOR J = 1 TO 4
1400 EI = 0
1410 FOR K = 1 TO 2
1420 EI = EI + A(I,K) * B(K,J)
1430 E(I,J) = EI
1440 NEXT K,J,I
1450 FOR I = 1 TO 3
1460 FOR J = 1 TO 3
1470 HI = 0
1480 FOR K = 1 TO 4
1490 HI = HI + E(I,K) * C(K,J)
1500 H(I,J) = HI
1510 NEXT K,J,I
1520 FOR I = 1 TO 3
1530 FOR J = 1 TO 5
1540 MI = 0
1550 FOR K = 1 TO 4
1560 MI = MI + E(I,K) * D(K,J)
1570 M(I,J) = MI
1580 NEXT K,J,I
1590 FOR I = 1 TO 3
1600 NI = 0
1610 FOR J = 1 TO 3
1620 NI = NI + H(I,J) * F(J)
1630 N(I) = NI
1640 NEXT J,I
1650 FOR I = 1 TO 3
1660 PI = 0
1670 FOR J = 1 TO 5
1680 PI = PI + M(I,J) * G(J)
1690 P(I) = PI
1700 NEXT J,I
1710 FOR I = 1 TO 4
1720 QI = 0
1730 FOR J = 1 TO 3
1740 QI = QI + C(I,J) * F(J)
1750 Q(I) = QI
1760 NEXT J,I
1770 FOR I = 1 TO 4
1780 RI = 0
1790 FOR J = 1 TO 5
1800 RI = RI + D(I,J) * G(J)
1810 R(I) = RI
1820 NEXT J,I
1830 FOR I = 1 TO 3
1840 S(I) = N(I) + P(I)
1850 NEXT I
1860 FOR I = 1 TO 4
1870 T(I) = Q(I) + R(I)
1880 NEXT I
1890 FOR I = 1 TO 3
1900 VI = 0
1910 FOR J = 1 TO 4
1920 VI = VI + E(I,J) * T(J)
1930 V(I) = VI
1940 NEXT J,I
1950 REM Ergebnis-Ausdruck ab Zeile
     1960
```

Konfiguration der seriellen Schnittstellen beim Apple IIc

von Enno Klatt

Das in Applesoft-BASIC geschriebene Programm **INIT.SERIELL** ermöglicht die Konfiguration der seriellen Schnittstellen unter den Betriebssystemen ProDOS und DOS. Diese Konfiguration ist bis zum nächsten Einschalten des Rechners permanent vorhanden. Das Programm ist eine Alternative zur zeitraubenden Konfiguration mit Hilfe der „UTILITY-Diskette".

Was passiert nach PR#1?

Im „Apple IIc Reference Manual" wird eine Konfiguration z.B. für Schnittstelle 1 mit Kontroll-Sequenzen nach dem Aktivieren mit „PR#1", beschrieben. Es fehlt hier nur der wichtige Hinweis, daß alle so vorgenommenen Änderungen der Konfiguration nur bis zum nächsten „PR#1" gelten, das heißt sie sind nicht permanent.
Ein „PR#1" führt stets eine Initialisierung der Schnittstellen durch. Dabei wird die Schnittstelle mit den im sekundären Speicherbereich (d.h. im „oberen" Bildschirmspeicher) abgelegten Konfigurationswerten eingestellt. Diese Speicherstellen werden aber durch obige Kontroll-Sequenzen nicht verändert.
Das Konfigurationsprogramm INIT.SERIELL verändert nun die obigen Speicherstellen. Wird z.B. ein Bildschirm-Echo eingerichtet, so ist dies während jeder Drukkerausgabe, also nach jedem „PR#1" vorhanden (sofern der Rechner nicht ausgeschaltet wurde).
Das Programm kann Teil des STARTUP-Programms unter ProDOS sein oder von diesem aufgerufen werden. Entsprechendes gilt für DOS. Mit einem Programmlauf kann entweder „Port 1" (Druckeranschluß) oder „Port 2" (Kommunikationsanschluß) konfiguriert werden.

Das Konfigurationsprogramm ist speziell für den Apple IIc gedacht, da nur dieser zwei serielle Schnittstellen besitzt, die vom Rechner software-mäßig gesteuert werden.

Anpassung des Programms

Das Programm arbeitet nicht interaktiv, da es die Konfiguration eines Ports ohne Zeitverlust vornehmen soll.
Bevor das Programm gestartet werden kann, sind in den Programmzeilen 1060 und 1070 (siehe Listing) die DATA-Anweisungen mit dem gewünschten Port und der Konfiguration einzutragen, womit die Beispiel-Eintragungen überschrieben werden.

Zeile 1060:
Hier ist die Nummer des zu konfigurierenden Ports in numerischer Form einzugeben:
– Druckeranschluß: 1 für Port 1,
– Kommunikationsanschluß: 2 für Port.

Zeile 1070:
Apple hat mit dem IIc die sogenannten Produkt-Identifikationsnummern, kurz „PIN" genannt, definiert. Diese werden im Apple-IIc-Handbuch, „System-Dienstprogramme" in bezug auf die UTILITY-Diskette erklärt. Eine Konfiguration mit PINs hat die Form „xxx/xxxx", wobei „x" für bestimmte Ziffern steht. Diese Konfiguration ist in der DATA-Zeile 1070 als Zeichenkette einzugeben.
Die Konfiguration kann aus der **PIN-Tabelle** ermittelt werden. Darüber hinaus ist sie noch einmal in den Zeilen 1090 bis 1150 kurz zusammengefaßt.

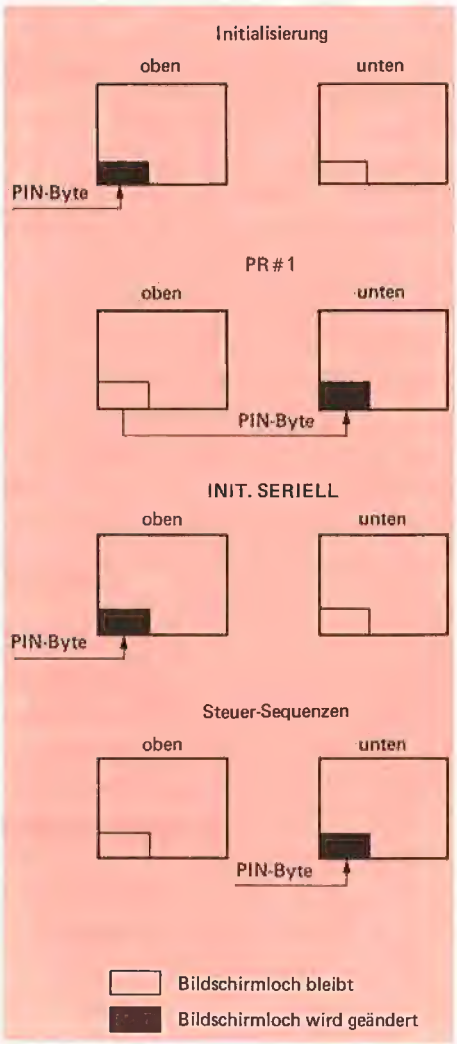
Das Konfigurationsprogramm wird nun auf übliche Weise gestartet. Es kann damit jederzeit neu konfiguriert werden, z.B. um beim Wechsel von einem Drucker zu einem Plotter geeignete Anpassungen der Schnittstellen vornehmen zu können.
Sollen beide Ports konfiguriert werden, so muß das Programm zweimal – abgespeichert unter sinnvollen Namen und versehen mit den besprochenen Eintragungen – ausgeführt werden.

Fehlermeldungen

–Angabe des Ports: Wurde in der DATA-Zeile 1060 nicht der numerische Wert 1 oder 2 eingeben, so wird nach der Fehlermeldung nicht konfiguriert.
–Angabe der Konfiguration in der Form „xxx/xxxx": Alle PINs werden auf die erlaubten maximalen bzw. minimalen Wert geprüft. Ist ein PIN oder sind mehrere nicht korrekt, so werden diese aufgelistet und es wird nicht konfiguriert.

Programmbeschreibung

Applesoft-BASIC besitzt im Gegensatz zu Rechnern mit einer neueren BASIC-Generation, wie z.B. dem HX-20 von Epson, keine direkte BASIC-Anweisung zur Schnittstellen-Konfiguration. Für eine „permanente" Konfiguration bleibt nur der Weg über PEEKs und POKEs.
Wie eingangs erwähnt, führt ein „PR#1" stets eine Initialisierung der Schnittstellen durch, wobei das Initialisierungsprogramm die Konfigurationswerte aus den sogenannten „Screenholes" (Bildschirmlöchern) des „Auxiliary Memory" (sekundären oder oberen Speicherbereichs) entnimmt. Es gibt drei Möglichkeiten, diese Speicherstellen zu beeinflussen:



Initialisierung der Screenholes (schematisch)

1. Rechner aus- und einschalten: Beim Einschalten werden aus dem ROM Konfiguration-Ausgangswerte in die Screenholes geschrieben.
2. Utility-Programm: Konfigurationen können definiert werden. Sie gelten bis zum nächsten Einschalten.
3. PEEKs und POKEs: Mit dem hier vorgestellten Programm oder Teilen daraus kann die Konfiguration eingestellt werden.

Es ist wichtig zu wissen, daß obige Konfigurationen erst nach der nächsten Initialisierung, z.B. mit „PR#1", an der Schnittstelle wirksam werden.

Es sind jeweils vier Speicherstellen für Port 1 und Port 2 vorhanden, die die Konfiguration enthalten (siehe Listing Zeile 1420 bis 1580). So enthält z.B. die Speicherstelle $0478 (Sekundärer Speicherbereich) die durch PIN 2 und PIN 3 definierten Werte. Da eine Speicherstelle aus einem Byte mit acht Bits besteht, repräsentiert z.B. der in DATA-Zeile 1180 angegebene dezimale Wert „14" den binären Wert „0000 1110". Sinngemäß definieren alle PINs Werte, die die entsprechenden Bits in den o.g. Speicherstellen auf 1 oder 0 setzen.

Das Konfigurationsprogramm ist in vier Abschnitte gegliedert:
– Konfigurationswerte definieren,
– Eingabe auslesen,
– Konfigurationswerte berechnen,
– Konfiguration durchführen.
Die Ausführungszeit des Programms kann enorm reduziert werden, wenn nur der Programmabschnitt „Konfiguration durchführen" ausgeführt wird. Zu diesem Zweck läßt man sich nach Zeile 1730 die Variablen PO, W1, W2, W3 und W4 anzeigen. Werden diese Variablen dann durch einfache Wertzuweisung vorangestellt, kann sich das Programm auf den letzten Teil reduzieren.

Kurzhinweise

1. Zweck:
Programm zur Initialisierung von jeweils einer der beiden seriellen Schnittstellen beim Apple IIc.
2. Konfiguration:
Apple IIc;
DOS 3.3 oder ProDOS
3. Test:
RUN INIT.SERIELL
(vorher eventuell gewünschte PIN-Werte in DATA-Zeilen eintragen)
4. Sammeldisk:
INIT.SERIELL
(Applesoft-Programm)

INIT.SERIELL

```
1000 REM Serielle Schnittstellen definieren
1010 REM von Enno Klatt, Hannover
1020 HOME
1030 PRINT TAB( 10);"Serielle Schnittstelle konfigurieren"
1040 REM ------------------------------------------------
1050 REM Programmabschnitt: Konfigurationswerte definieren
1060 DATA 1           : REM "PORT 1" oder "PORT 2" numerisch
1070 DATA "166/1124" : REM Konfiguration in "ASCII": "xxx/xxxx"
1080 REM PIN
1090 REM 1: Drucker, Kommunikation
1100 REM 2: 6/1, 6/2, 7/1, 7/2, 8/1, 8/2 Byte/Stopp-Bits
1110 REM 3: 110, 300, 1200, 2400, 4800, 9600, 19200 Baud
1120 REM 4: keine, gerade, ungerade, Mark-, Space-Parität
1130 REM 5: Echo aus, Echo ein
1140 REM 6: ohne LF, mit LF
1150 REM 7: kein CR, CR nach 40, 72, 80, 132 Zeichen
1160 DATA 0,1                    : REM PIN 1
1170 DATA 64,192,32,160,0,128 : REM PIN 2
1180 DATA 3,6,8,10,12,14,15    : REM PIN 3
1190 DATA 0,96,32,160,224      : REM PIN 4
1200 DATA 0,128                  : REM PIN 5
1210 DATA 0,64                   : REM PIN 6
1220 DATA 0,40,72,80,132       : REM PIN 7
1230 REM ------------------------------------------------
1240 REM Programmabschnitt: Eingabe auslesen
1250 REM "PORT" bestimmen
1260 READ P0
1270 REM Konfiguration lesen (PINs bestimmen)
1280 READ K0$
1290 K1 = VAL ( MID$ (K0$,1,1)):K2 = VAL ( MID$ (K0$,2,1))
1300 K3 = VAL ( MID$ (K0$,3,1)):K4 = VAL ( MID$ (K0$,5,1))
1310 K5 = VAL ( MID$ (K0$,6,1)):K6 = VAL ( MID$ (K0$,7,1))
1320 K7 = VAL ( MID$ (K0$,8,1))
1330 REM Konfiguration prüfen
1340 IF K1 < 1 OR K1 > 2 THEN PRINT "PIN 1":FL = 1
1350 IF K2 < 1 OR K2 > 6 THEN PRINT "PIN 2":FL = 1
1360 IF K3 < 1 OR K3 > 7 THEN PRINT "PIN 3":FL = 1
1370 IF K4 < 1 OR K4 > 5 THEN PRINT "PIN 4":FL = 1
1380 IF K5 < 1 OR K5 > 2 THEN PRINT "PIN 5":FL = 1
1390 IF K6 < 1 OR K6 > 2 THEN PRINT "PIN 6":FL = 1
1400 IF K7 < 1 OR K7 > 5 THEN PRINT "PIN 7":FL = 1
1410 IF FL = 1 THEN PRINT " war(en) falsch": END
1420 REM
1430 REM ------------------------------------------------
1440 REM Programmabschnitt: Konfigurationswerte berechnen
1450 REM Adressen im "AUXILIARY MEMORY" welche die
1460 REM - permanente Konfiguration speichern -
1470 REM "PORT 1"-Adressen: Drucker
1480 REM $0478 (1144) definiert durch PIN 2 und 3   --> Wert: W1
1490 REM $0479 (1145) definiert durch PIN 4         --> Wert: W2
1500 REM $047A (1146) definiert durch PIN 1,5 und 6 --> Wert: W3
1510 REM $047B (1147) definiert durch PIN 7         --> Wert: W3
1520 REM
1530 REM "PORT 2"-Adressen: Kommunikation
1540 REM $047C (1148) definiert durch PIN 2 und 3   --> Wert: W1
1550 REM $047D (1149) definiert durch PIN 4         --> Wert: W2
1560 REM $047E (1150) definiert durch PIN 1,5 und 6 --> Wert: W3
1570 REM $047F (1151) definiert durch PIN 7         --> Wert: W4
1580 REM
1590 REM "PIN 1" auswerten
1600 FOR I = 1 TO 2: READ X(I): NEXT :W3 = X(K1)
1610 REM "PIN 2" auswerten
1620 FOR I = 1 TO 6: READ X(I): NEXT :W1 = X(K2)
1630 REM "PIN 3" auswerten
1640 FOR I = 1 TO 7: READ X(I): NEXT :W1 = W1 + X(K3)
1650 REM "PIN 4" auswerten
1660 FOR I = 1 TO 5: READ X(I): NEXT :W2 = X(K4)
1670 REM "PIN 5" auswerten
1680 FOR I = 1 TO 2: READ X(I): NEXT :W3 = W3 + X(K5)
1690 REM "PIN 6" auswerten
1700 FOR I = 1 TO 2: READ X(I): NEXT :W3 = W3 + X(K6)
1710 REM "PIN 7" auswerten
1720 FOR I = 1 TO 5: READ X(I): NEXT :W4 = X(K7)
1730 REM ------------------------------------------------
1740 REM Programmabschnitt: Konfiguration durchführen
1750 REM Sekundären Speicherbereich zum Schreiben aktivieren
1760 REM "80STORE" an
1770 POKE 49153,0: REM $C001
1780 REM "AUXILIARY MEMORY" an
1790 S = PEEK (49237): REM $C055
1800 REM "PORT" auswählen
1810 IF P0 = 1 THEN 1850
1820 IF P0 = 2 THEN 1910
1830 PRINT "PORT-Angabe falsch!": END
1840 REM "PORT 1"
1850 POKE 1144,W1 + 16: REM BIT 5       stets an -->    16 = 16
1860 POKE 1145,W2 + 11: REM BIT 1, 2, 4 stets an --> 1+2+8 = 11
1870 POKE 1146,W3
1880 POKE 1147,W4
1890 GOTO 1980
1900 REM "PORT 2"
1910 POKE 1148,W1 + 16
1920 POKE 1149,W2 + 11
1930 POKE 1150,W3
1940 POKE 1151,W4
1950 REM ------------------------------------------------
1960 REM Sekundären Speicherbereich deaktivieren
1970 REM "PRIMARY MEMORY" an
1980 S = PEEK (49236): REM $C054
1990 REM "80STORE" aus
2000 POKE 49152,0: REM $C000
2010 END
```

PIN-Tabelle

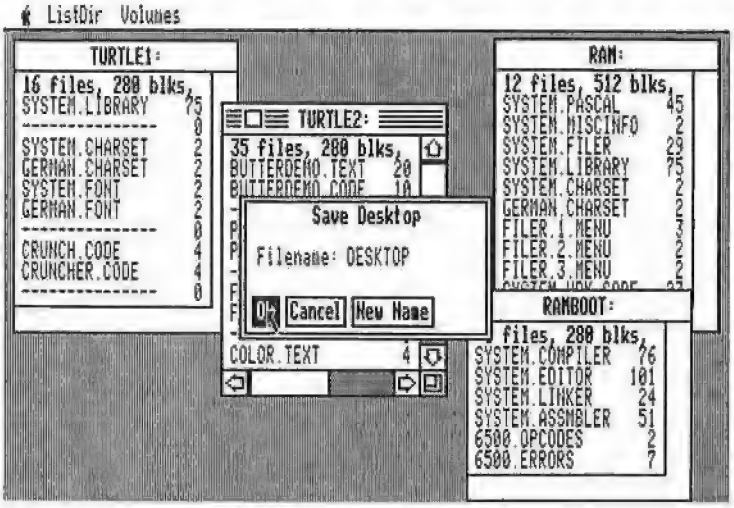
Die gesamte PIN setzt sich zusammen aus 7 Ziffern im Format abc/defg. Die Ziffern a-g entsprechen den PINs 1-7 und haben folgende Bedeutungen:

```
Ziffer a: 1 = Drucker
          2 = Kommunikation
Ziffer b: 1 = 6 Daten-Bits/ 1 Stopp-Bit
          2 = 6 Daten-Bits/ 2 Stopp-Bits
          3 = 7 Daten-Bits/ 1 Stopp-Bit
          4 = 7 Daten-Bits/ 2 Stopp-Bits
          5 = 8 Daten-Bits/ 1 Stopp-Bit
          6 = 8 Daten-Bits/ 2 Stopp-Bits
Ziffer c: 1 =   110 Baud
          2 =   300 Baud
          3 =  1200 Baud
          4 =  2400 Baud
          5 =  4800 Baud
          6 =  9600 Baud
          7 = 19200 Baud
Ziffer d: 1 = keine Parität
          2 = gerade Parität
          3 = ungerade Parität
          4 = Mark-Parität
          5 = Space-Parität
Ziffer e: 1 = Bildschirm-Echo aus
          2 = Bildschirm-Echo an
Ziffer f: 1 = Ohne LF nach Cr
          2 = LF nach CR
Ziffer g: 1 = kein Cr
          2 = CR nach  40 Zeichen
          3 = CR nach  72 Zeichen
          4 = CR nach  80 Zeichen
          5 = Cr nach 132 Zeichen
```

TurtleGraphics-Library-Paket

von Dieter Geiß

Turtle-Utilities für Fenstertechnik und Apple-Maus in einfacher und doppelter Hires-Grafik für Pascal 1.1/1.2 auf Apple II+/e/c mit Maus oder Joystick.
2 Disketten mit umfangreichem Manual, DM 98,–
Erscheinungstermin Anfang 1986



Das Utility-Paket besteht aus vier Modulen, die von Programmierern benutzt werden können, um professionelle grafische Anwendungsprogramme in Pascal zu schreiben.

Benötigt wird ein Apple Pascal Betriebssystem, entweder die Version 1.1 oder die neue Version 1.2. Bestehende Programme laufen ohne Einschränkung mit der neuen „TurtleGraphics", wenn diese nicht zu viel Speicherplatz verbraucht haben, da die neue „TurtleGraphics" umfangreicher als die alte ist.

Im einzelnen bietet das Paket folgende Möglichkeiten:

- volle Kompatibilität mit der alten „TurtleGraphics"
- lauffähig auf Pascal 1.1 und 1.2
- funktionsfähig mit angeschlossener UltraTerm-Karte
- alle zeitkritischen Funktionen in reinem Assembler programmiert
- Benutzung der zweiten Hires-Seite ($4000–$5FFF) möglich
- für Apple IIc und Apple IIe mit erweiterter 80-Zeichen-Karte Benutzung der doppelten Hires-Grafik mit 560 × 192 Punkten bzw. 16 neuen Farben möglich
- schnelle Prozeduren zum Zeichnen eines Punktes oder einer Linie
- Linearisierung von Teilen des Hires-Schirms
- Benutzung mehrerer Zeichensätze gleichzeitig
- Laden und Speichern von Hires-Bildern mit Ausdruck über Pascal-SUPERDUMP
- Scrolling des Hires-Schirms oder eines Teils in vier Richtungen
- drei verschiedene Schriftarten: Fett-, Breit- und Proportionalschrift, beliebig mischbar (acht Möglichkeiten)
- spezielle schnelle Ausgabe von Text
- Cursor bei Eingabe frei programmierbar
- Ein-/Ausgabe von INTEGER-, CHAR-, STRING- und REAL-Werten im Grafikmodus
- Menüzeile wie beim Macintosh (Die nachfolgenden Module benötigen Maus/Joystick)
- Pull-down-Menüs
- Laden und Speichern von Fenstern (Windows)
- Öffnen von Fenstern
- Aktivieren und Deaktivieren von Fenstern
- Verschieben und Vergrößern/Verkleinern von Fenstern
- Scrolling von Fensterinhalten in allen vier Richtungen
- Umfangreiche Demos als Quelltexte.

Hüthig Software Service · Postfach 10 28 69 · 6900 Heidelberg

Eine ganze „Bibliothek“ von Applesoft-Erweiterungen können Sie in Ihren BASIC-Programmen verwenden, wenn Sie Assembler-Routinen nicht nur mit einem einfachen CALL aufrufen, sondern für deren Aufruf neue Befehlsworte schaffen und zusätzlich Parameter übergeben. Im folgenden wird beschrieben, wie Sie das machen können.

Befehlserw

CHRGET-M

von Dr. Jürgen B. Kehrel

Im Peeker, Heft 2/84, habe ich Ihnen bereits ausführlich dargelegt, wie Sie mit dem Ampersand-Befehl „&“ einen Übergang zwischen Applesoft und Assembler schaffen können. Dort haben wir gesehen, daß Applesoft fast alle Informationen mittels der CHRGET/CHRGOT-Routine liest. Da diese im RAM-Bereich liegt ($00B1-$00C8), kann sie natürlich von uns verändert werden. Das gibt uns neben dem Ampersand eine zweite komfortable Möglichkeit, eigene Maschinenprogramme in Applesoft einzubinden. Wir müssen nur CHRGET so manipulieren, daß unser Programm erst überprüfen kann, ob es „gemeint“ ist, bevor die Kontrolle wieder an Applesoft zurückgegeben wird. Zu den reservierten Wörtern von Applesoft fügen wir eigene hinzu oder verleihen den existierenden Befehlen neue Bedeutungen. Es sollte davon abgesehen werden, Namen wie
SORTIERE
zu verwenden, da Applesoft darin OR entdeckt, diesen Teil als Token abspeichert (s. Applesoft BASIC Programmmieranleitung, S. 126/127) und hinterher als
S OR TIERE
listet. Also Vorsicht bei der Namensgebung!

Wenn wir mehrere verschiedene Maschinenprogramme aufrufen oder in einem Programm an unterschiedlichen Stellen einspringen wollen, müssen wir eine Entscheidungs- und Sprungtabelle aufbauen. Damit nicht bei jedem Zeichen, das mit CHRGET gelesen wird, die ganze Tabelle durchsucht wird und dann erst der Befehl an Applesoft zurückgeht, was einen merklich langsameren Programmablauf zur Folge hätte, werden wir alle neuen Befehle mit demselben Zeichen beginnen lassen. Dieses sollte im übrigen Text nicht auftauchen. Bewährt hat sich das Paragraphzeichen „§“ (auf amerikanischen Tastaturen der Klammeraffe „@“). Wenn ein Befehl nicht damit anfängt, gehen wir sofort nach Applesoft weiter.
Noch eine Anmerkung, bevor wir uns das Listing genauer ansehen:
Applesoft holt nicht alle Zeichen über CHRGET. REMs und DATAs werden z.B. mittels LDA TXTPTR,Y übersprungen. Das gleiche gilt (andernfalls wäre diese Manipulation nicht möglich) bei dem Überspringen von Zeilenenden nach unzutreffenden IF-Anweisungen. Auch Strings werden nicht über CHRGET eingelesen: Nach Auswertung des Stringdeskriptors (String-Länge und -Position) wird der Textpointer (TXTPTR, $00B8) mit der String-Länge addiert. Vorsicht ist jedoch bei Variablenna-

men geboten, da diese mit CHRGET eingelesen werden. Solange alle eigenen Befehle mit „§“ beginnen, bereitet dies jedoch keine Schwierigkeiten.
Unser Programm **CHRGET.SPRUNG** setzt zunächst einmal HIMEM unter sich selbst, wenn es gestartet wird. Wie schon beim INSTRING-Programm aus Peeker, Heft 2/84 sollte das gleich in der ersten Zeile des Applesoft-Programms mit
PRINT CHR$(4);“BRUN CHRGET .SPRUNG“
geschehen. Danach wird in die Speicherstellen $00BA-$00BC ein Sprung zur zweiten Einstiegsstelle geschrieben und aus der CHRGET/CHRGOT-Routine immer zu dieser Adresse verzweigt. Die Initialisierung ist damit abgeschlossen.
Jedes gelesene Zeichen (auch im Direktmodus) wird danach mit „§“ verglichen, und im Falle der Übereinstimmung springt das Programm in die Auswertung. Andernfalls werden die in $00BA-$00BC überschriebenen Befehle nachgeholt, und es wird evtl. in die restliche CHRGET-Routine nach $00BE zurückgesprungen. Applesoft „merkt“ gar nicht, daß zwischendurch noch etwas geschehen ist.
Nehmen wir also an, ein „§“ sei gefunden worden. Jetzt werden so lange die nachfolgenden Zeichen gelesen, bis entweder eine offene Klammer, ein Doppelpunkt oder eine Null folgt, allerdings maximal 7 Zeichen. Unser Befehlswort wird temporär nach TEMP zwischengespeichert, um für die Suche in der Sprungtabelle zur Verfügung zu stehen. Die offene Klammer gibt uns die Möglichkeit, noch Parameter an das eigentliche Maschinenprogramm zu übergeben, so daß ein Befehl z.B. die Form „§SUCH (A$,B$,F%,I%)“ haben kann, womit wir die o.g. INSTRING-Suche auch starten könnten.
Im Anschluß daran wird nun das Befehlswort in der Tabelle gesucht. Wird es nicht gefunden, erfolgt die Meldung „SYNTAX ERROR“. Wird es in voller Länge gefunden, so wird die in den folgenden Bytes stehende Anfangsadresse in den Sprungbefehl kopiert und danach ausgeführt.
Ihrer Phantasie sind bei der Aufstellung der Sprungtabelle keine Grenzen gesetzt, solange diese nicht länger als 256 Zeichen wird, da sie sonst nicht mehr indiziert gelesen werden kann. Je länger Ihre Befehlsworte sind, um so weniger Befehle gehen natürlich hinein, so daß Sie eventuell einen Kompromiß schließen müssen zwischen der Anschaulichkeit der Namen und der Zahl der möglichen Befehle. Die im Listing angegebenen Befehle sind natürlich nur Beispiele. Sie sollten bei einer Erweiterung oder Umbenennung aber folgende Struktur beibehalten:

Name1(ohne §), $00, Adresse 1, Name 2, $00, Adresse 2, ..., Name n, $00, Adresse n, $FF.

$00 markiert das Namensende, $FF das Tabellenende.
Das „Programm“ BELL ist ein einfacher Test, ob die Routine auch läuft. Nachdem Sie das Maschinenprogramm assembliert oder direkt eingegeben und mit „BSAVE CHRGET.SPRUNG, A$8400, L$00A5“ abgespeichert haben, starten Sie es mit BRUN. Wenn Sie nun direkt auf der Tastatur §BELL eingeben, sollte beim Drücken der Return-Taste Ihr Apple einen kurzen „Beep“ abgeben und danach sofort der normale Cursor auftauchen. Erscheint dagegen „?SYNTAX ERROR“, haben Sie etwas falsch eingetippt.
Als weiteres Beispiel wurde der erweiterte GOTO-Befehl aufgenommen, der als Argument auch mathematische Ausdrücke zuläßt. Hier zeigt sich, daß auch einzelne Befehlsworte in die Tabelle geschrieben werden können. Zu beachten ist jedoch, daß bei der Benutzung des neuen Befehls eines der Trennzeichen („:“ oder „(“) nachgestellt werden muß. Also:
§ GOTO : 2 * 5.

Beim Eintippen einer Programmzeile, in der ein §-Befehl vorkommt, kann Wundersames geschehen. Wenn das erste Zeichen nach der Zeilennummer ein „§“ ist, wird der folgende Befehl sofort ausgeführt, ohne die Zeile in den Programmtext zu übernehmen. Dies liegt daran, daß die Zeilennummer von der Routine LINGET mittels CHRGET eingelesen wird, bis keine Ziffer mehr folgt. Da, um dies festzustellen, das „§“ mitgelesen wird, verzweigt das geänderte CHRGET sofort zur Befehlsausführung. Um dies zu vermeiden, sollte entweder nur mit ausgeschalteter CHRGET-Manipulation programmiert (z.B. durch „FP“ vor dem Eingeben eines neuen Programms) oder dem §-Befehl im o.g. Fall ein „:“ vorangestellt werden. CHRGET.SPRUNG wird durch CALL 33792 eingeschaltet, wenn es sich bereits im Speicher befindet.

Sie haben sich jetzt ein Gerüst geschaffen, in das Sie nach Herzenslust eigene Routinen schreiben können. Im Peeker werden im Verlauf der nächsten Ausgaben sicherlich interessante Maschinenprogramme erscheinen, die Sie mit CHRGET-.SPRUNG ganz einfach zu Ihrer privaten Programmbibliothek zusammenstellen können.

eiterung durch anipulation

Kurzhinweise

1. Zweck:
Hilfsprogramm zur Applesoft-Befehlserweiterung.
2. Konfiguration:
Apple II+, IIe oder IIc;
DOS 3.3 (nicht ProDOS wegen HIMEM-Änderung!).
3. Test:
BRUN CHRGET.SPRUNG
§ BELL
4. Sammeldisk:
CHRGET.SPRUNG
(Maschinenprogramm)
T.CHRGET.SPRUNG
(Big-Mac-Quelltext).

CHRGET.SPRUNG

```
BSAVE CHRGET.SPRUNG, A$84ØØ, L$ØØA5

                    1   * CHRGET.SPRUNG
                    2   *
                    3   HIMEM    EQU  $ØØ73
                    4   CHRGET   EQU  $ØØB1
                    5   TXTPTR   EQU  $ØØB8
                    6   BACK     EQU  $ØØBE
                    7   CHRSPG   EQU  $ØØBA
                    8   NEWSTT   EQU  $D7D2
                    9   GOTOENT  EQU  $D941
                    1Ø  FRMNUM   EQU  $DD67
                    11  SYNERR   EQU  $DEC9
                    12  GETADR   EQU  $E752
                    13  *
                    14          ORG  $84ØØ
                    15  ****************************************
                    16  *                                      *
                    17  * Syntax: §NAME(1.PARAM,2.PARAM, ...) *
                    18  *                                      *
                    19  ****************************************
                    2Ø  * Dr. Jürgen B. Kehrel, Juli 1984
                    21  *
                    22  * Setze HIMEM auf START
                    23  *
8400: A9 ØØ         24  START    LDA  #<START
8402: 85 73         25          STA  HIMEM
8404: A9 84         26          LDA  #>START
8406: 85 74         27          STA  HIMEM+1
                    28  *
                    29  * Setze Einsprung auf EINSPG
                    3Ø  *
8408: A9 4C         31          LDA  #$4C       ;JMP
840A: 85 BA         32          STA  CHRSPG
840C: A9 15         33          LDA  #<EINSPG
840E: 85 BB         34          STA  CHRSPG+1
8410: A9 84         35          LDA  #>EINSPG
8412: 85 BC         36          STA  CHRSPG+2
8414: 6Ø            37  RETURN   RTS            ;zurück z. Aufrufer
                    38  *
                    39  * Zweiter Einstiegspunkt
                    4Ø  *
8415: C9 4Ø         41  EINSPG   CMP  #$4Ø       ;ist es "§"?
8417: FØ Ø7         42          BEQ  AUSWRT     ;weitere Auswertung
8419: C9 3A         43          CMP  #$3A       ;ist es ":"?
841B: BØ F7         44          BCS  RETURN
841D: 4C BE ØØ      45          JMP  BACK       ;zurück zu CHRGET
8420: AØ ØØ         46  AUSWRT   LDY  #Ø         ;lade gesamt. Namen
8422: A2 ØØ         47          LDX  #Ø
8424: E6 B8         48  NAMEN    INC  TXTPTR
8426: DØ Ø2         49          BNE  NAMEN1
8428: E6 B9         5Ø          INC  TXTPTR+1
842A: B1 B8         51  NAMEN1   LDA  (TXTPTR),Y ;nächstes Zeichen
842C: C9 3A         52          CMP  #$3A       ;ist es ":"
842E: FØ 12         53          BEQ  SUCHEN
8430: C9 28         54          CMP  #$28       ;ist es "("
8432: FØ ØE         55          BEQ  SUCHEN
8434: C9 ØØ         56          CMP  #ØØ        ;ist es Zeilenende
8436: FØ ØA         57          BEQ  SUCHEN
8438: 9D 82 84      58          STA  TEMP,X
843B: E8            59          INX
843C: E0 Ø8         6Ø          CPX  #8         ;max. Länge 7
843E: FØ 3F         61          BEQ  ERR        ;zu lang
8440: DØ E2         62          BNE  NAMEN
8442: 8E 89 84      63  SUCHEN   STX  LNAME     ;Namenslänge
8445: A2 ØØ         64          LDX  #Ø
8447: AØ ØØ         65  SUCH1    LDY  #Ø
8449: BD 8A 84      66  SUCH2    LDA  TAB,X
844C: C9 FF         67          CMP  #$FF
844E: FØ 2F         68          BEQ  ERR        ;nicht gefunden
8450: D9 82 84      69          CMP  TEMP,Y
8453: DØ 1D         7Ø          BNE  WEITER     ;nächster Name
8455: E8            71          INX
8456: C8            72          INY
8457: CC 89 84      73          CPY  LNAME
845A: DØ ED         74          BNE  SUCH2
845C: BD 8A 84      75          LDA  TAB,X      ;Name ganz gelesen?
845F: DØ 1E         76          BNE  ERR
8461: E8            77          INX
8462: BD 8A 84      78          LDA  TAB,X      ;speichere Adresse
8465: 8D 7Ø 84      79          STA  AUFRUF+1
8468: E8            8Ø          INX
8469: BD 8A 84      81          LDA  TAB,X
846C: 8D 71 84      82          STA  AUFRUF+2
846F: 4C FF FF      83  AUFRUF   JMP  $FFFF      ;Pseudowert
8472: BD 8A 84      84  WEITER   LDA  TAB,X
8475: FØ Ø3         85          BEQ  WEITER1    ;Namensende gefund.
8477: E8            86          INX
8478: DØ F8         87          BNE  WEITER
847A: E8            88  WEITER1  INX            ;überspringe Adr.
847B: E8            89          INX
847C: E8            9Ø          INX
847D: DØ C8         91          BNE  SUCH1
847F: 4C C9 DE      92  ERR      JMP  SYNERR     ;SYNTAX ERROR
                    93  TEMP     DS   7          ;temp. Speicher
                    94  LNAME    DS   1          ;Namenslänge
                    95  *
                    96  * Tabelle: Namen + Adressen
                    97  *
8482: AB            98  TAB      DFB  171        ;GOTO-Token
8483: ØØ            99          HEX  ØØ
8484: 96 84         1ØØ         DA   GOTO
8486: 42 45 4C      1Ø1         ASC  'BELL'
8489: 4C
848A: ØØ            1Ø2         HEX  ØØ
848B: DD FB         1Ø3         DA   $FBDD
848D: FF            1Ø4         HEX  FF
                    1Ø5  *
                    1Ø6  * Neuer GOTO-Befehl
                    1Ø7  *
848E: 2Ø B1 ØØ      1Ø8  GOTO     JSR  CHRGET     ;":" übergehen
8491: 2Ø 67 DD      1Ø9          JSR  FRMNUM     ;Ausdruck
8494: 2Ø 52 E7      11Ø          JSR  GETADR     ;auswerten
8497: 2Ø 41 D9      111          JSR  GOTOENT    ;GOTO ausführen
849A: 4C D2 D7      112          JMP  NEWSTT     ;neuer Befehl

165 Bytes
```

Bildschirmmasken in der Language Card

von Carl Frieder Mahr

Oft stellt sich bei Programmen das Problem, daß man nach dem Ausdruck einer Meldung nicht mehr weiß, wie der Bildschirm vorher aussah. Man hat es in diesem Fall schwer, den ursprünglichen Bildschirminhalt zu rekonstruieren, speziell wenn man mit Bildschirmmasken arbeitet. Hier schafft das folgende Programm **SCREEN.MOVER** Abhilfe, das es ermöglicht, vor der Ausgabe einer Meldung den Bildschirminhalt abzuspeichern und nachher wieder herzustellen.
Eine andere Anwendungsmöglichkeit für das Programm besteht darin, eine Bildfolge abzuspeichern, die danach sehr schnell aufgerufen werden kann. Es lassen sich dann in Verbindung mit der Lores- oder der Double-Lores-Grafik einfache Animationen realisieren.

Die Befehle

SCREEN.MOVER kann nach seinem Aufruf durch „BRUN SCREEN.MOVER" bis zu acht Bildschirminhalte verwalten. Mit „&STORE 0" bis „&STORE 7" läßt sich der aktuelle Bildschirminhalt in einen der Speicherplätze 0-7 abspeichern. Der Befehl zum Laden eines Bildschirminhalts lautet entsprechend „&LOAD 0" bis „&LOAD 7". Die Festlegung der Puffernummer kann auch als Variable oder als Ausdruck eingegeben werden, also z.B. „&LOAD I/2".
Der SCREEN.MOVER stellt automatisch fest, ob die 80-Zeichenkarte aktiv ist, so daß sich sowohl 80- wie auch 40-Z/Z-Bildschirmseiten abspeichern lassen. Man sollte jedoch davon absehen, im 80-Z/Z-Modus zu speichern und im 40-Z/Z-Modus den Puffer wieder einzulesen.
Eine Beispiel zur Benutzung der beiden Befehle zeigt das Demoprogramm SC.MOVER.DEMO.

Speicherbelegung

Das Bildschirm-Speicherprogramm liegt normalerweise ab Adresse $0300 (768), läßt sich aber mit einem kleinen Patch in der INIT-Routine auch in andere Speicherbereiche laden.
Die Bildschirminhalte belegen die Language Card ab $D000 (53248), sowohl Bank 1 als auch Bank 2. Man kann auf Diskette abgespeicherte Bildschirme (z.B. fertige Menü- oder Hilfsseiten) direkt dorthin laden und mit „&LOAD 0..7" abrufen. Durch „POKE 785,2" läßt sich die Pufferzahl auf 2 begrenzen, um unter ProDOS nur die Bank 2 der Language Card zu benutzen.
Es sei abschließend darauf hingewiesen, daß SCREEN.MOVER die sog. Screenholes („Bildschirmlöcher") ausspart, in denen diverse Parameter für die Peripheriegeräte abgelegt werden. Durch den DOS-3.3-Test

```
LOAD PROGRAMM1
CALL -151
1000:0
400<1000.13FFM
LOAD PROGRAMM2
```

(PROGRAMM1 und PROGRAMM2 sind beliebige Applesoft-Programme) kann man sich davon überzeugen, daß das unbedachte Löschen der Screenholes unweigerlich zu einer Rekalibrierung des Laufwerks führt.

Kurzhinweise

1. Zweck:
Pufferung von bis zu 8 Bildschirmseiten in der LC.
2. Konfiguration:
Apple II+ (nur 40 Z/Z), IIe oder IIc; nicht Videx-Karte!
DOS 3.3 oder ProDOS
(unter ProDOS durch „POKE 785,2" die Pufferzahl begrenzen)
3. Test:
RUN SC.MOVER.DEMO
4. Sammeldisk:
SC.MOVER.DEMO
(Applesoft-Demoprogramm)
T.SCREEN.MOVER
(Big-Mac-Quelltext)
SCREEN.MOVER
(Maschinenprogramm)

SC.MOVER.DEMO

```
10 HOME
20 PRINT CHR$ (4)"BRUN SCREEN.MOVER"
30 LIST : PRINT : PRINT "Taste drücken": GET A$
40 & STORE 0: REM Bildschirm speichern
50 HOME : PRINT "Taste drücken": GET A$
60 & LOAD 0: REM Bildschirm wieder laden
70 VTAB 10: REM Cursor unter Listing setzen
```

SCREEN.MOVER

```
1   ********************************
2   *                              *
3   *         SCREEN.MOVER         *
4   *         ------------         *
5   *                              *
6   *    von Carl Frieder Mahr     *
7   *       Esslingen, 1985        *
8   *                              *
9   ********************************
10
11  * Gerätevoraussetzung:
12  *
13  * im 40-Z/Z-Modus: Apple II+,IIe,IIc
14  * im 80-Z/Z-Modus: Apple IIe,IIc
15
16  * Befehle:
17  *
18  * &LOAD  n : Lädt Bildschirm aus
19  *            Puffernummer n.
20  * &STORE n : Speichert Bildschirm
21  *            in Puffernummer n.
22  * Anmerkung: 8 Puffer stehen zur
23  *            Verfügung mit n = (0..7).
24
```

```
                        * Token-Definitionen
                        
                        TOKLOAD  =    $B6        ;"LOAD"
                        TOKSTORE =    $A8        ;"STORE"
                        
                        * Zeropage-Definitionen
                        
                        SCRN     EQU  $19        ;Bildschirm
                        LC1      EQU  $1B        ;ungerade Spalten
                        LC2      EQU  $1D        ;gerade Spalten
                        
                        * Applesoft-Routinen
                        
                        SYNERRP  EQU  $DEC9      ;"SYNTAX ERROR"
                        ILLQTYP  EQU  $E199      ;"ILL QUANTITY"
                        GTBYTC   EQU  $E6F5      ;Byte holen
                        
                        * Softswitches
                        
                        RD80VID  EQU  $C01F      ;80-Z/Z-Flag
                        PAGE1    EQU  $C054      ;Main RAM
                        PAGE2    EQU  $C055      ;Aux RAM
                        RWBANK1  EQU  $C08B      ;LC-Switch
                        RWBANK2  EQU  $C083      ;LC-Switch
                        RDROM    EQU  $C081      ;ROM-Switch
                        
                                 ORG  $0300
                        
                        * Ampersand-Vektor initialisieren
                        
0300: A9 0B     INIT     LDA  #<START
0302: 8D F6 03          STA  $03F6
0305: A9 03             LDA  #>START
0307: 8D F7 03          STA  $03F7
030A: 60                RTS
                        
                        * Befehl LOAD/STORE sichern
                        
030B: 48        START    PHA
                        
                        * Puffernummer holen und
                        * auf Gültigkeit überprüfen
                        
030C: 20 F5 E6          JSR  GTBYTC
030F: 8A                TXA
0310: C9 08             CMP  #$08
0312: 90 03             BCC  OK
                        
                        * Puffernummer zu groß =>
                        * Illegal Quantity Error drucken
                        
0314: 4C 99 E1          JMP  ILLQTYP
                        
                        * LC-Softswitches setzen und
                        * Basisadressen initialisieren
                        *
                        * Puffer 0: $D000..$D7FF Bank 2
                        * Puffer 1: $D800..$DFFF Bank 2
                        *
                        * Die Puffer 2 bis 7 sind nur unter
                        * DOS 3.3 benutzbar!
                        *
                        * Puffer 2: $E000..$E7FF Bank 1/2
                        * Puffer 3: $E800..$EFFF Bank 1/2
                        * Puffer 4: $F000..$F7FF Bank 1/2
                        * Puffer 5: $F800..$FFFF Bank 1/2
                        * Puffer 6: $D000..$D7FF Bank 1
                        * Puffer 7: $D800..$DFFF Bank 1
                        
0317: 2C 83 C0  OK       BIT  RWBANK2    ;Standard
031A: 2C 83 C0           BIT  RWBANK2    ;ist Bank 2
031D: 0A                 ASL             ;Puffernummer
031E: 0A                 ASL             ;mal 8 ergibt
031F: 0A                 ASL             ;Blockadresse
0320: 69 D0              ADC  #$D0
0322: 90 08              BCC  INBANK2
                        
                        * LOAD/STORE Puffernummer 6,7 =>
                        * LOAD/STORE ab $D000 in Bank 1
                        
0324: E9 30              SBC  #$30
0326: 2C 8B C0           BIT  RWBANK1
0329: 2C 8B C0           BIT  RWBANK1
                        
                        * Basisadressen festlegen
                        
032C: 85 1C     INBANK2  STA  LC1+1
032E: 69 04              ADC  #$04
0330: 85 1E              STA  LC2+1
0332: A9 04              LDA  #$04
0334: 85 1A              STA  SCRN+1
0336: A9 00              LDA  #$00
0338: 85 19              STA  SCRN
033A: 85 1B              STA  LC1
033C: 85 1D              STA  LC2
                        
                        * Befehl auswerten
                        
033E: 68                 PLA
033F: C9 B6              CMP  #TOKLOAD
0341: F0 0A              BEQ  LOAD
0343: C9 A8              CMP  #TOKSTORE
0345: F0 36              BEQ  STORE
                        
                        * Falscher Befehl => SYNTAX ERROR
                        
0347: 2C 81 C0           BIT  RDROM
034A: 4C C9 DE           JMP  SYNERRP
                        
                        * Bildschirminhalt laden
                        * ----------------------
                        
034D: A0 00     LOAD     LDY  #$00
034F: 2C 1F C0  LOAD0    BIT  RD80VID    ;80 Z/Z?
0352: 10 0A              BPL  LOAD1      ;=> nein
                        
                        * 80-Z/Z-Darstellung =>
                        * gerade Spalten laden
                        
0354: 2C 55 C0           BIT  PAGE2
0357: B1 1D              LDA  (LC2),Y
0359: 91 19              STA  (SCRN),Y
035B: 2C 54 C0           BIT  PAGE1
035E: B1 1B     LOAD1    LDA  (LC1),Y
0360: 91 19              STA  (SCRN),Y
0362: C8                 INY
                        
                        * I/O-RAM überspringen
                        
0363: C0 78              CPY  #$78
0365: D0 02              BNE  LOAD2
0367: A0 80              LDY  #$80
0369: C0 F8     LOAD2    CPY  #$F8
036B: 90 E2              BCC  LOAD0
                        
                        * MSB erhöhen
                        
036D: E6 1A              INC  SCRN+1
036F: E6 1C              INC  LC1+1
0371: E6 1E              INC  LC2+1
0373: A5 1A              LDA  SCRN+1
0375: C9 08              CMP  #$08       ;fertig?
0377: 90 D4              BCC  LOAD
0379: 2C 81 C0           BIT  RDROM
037C: 60                 RTS
                        
                        * Bildschirminhalt speichern
                        * --------------------------
                        
037D: A0 00     STORE    LDY  #$00
037F: 2C 1F C0  STORE0   BIT  RD80VID    ;80 Z/Z?
0382: 10 0A              BPL  STORE1     ;=> nein
                        
                        * 80-Z/Z-Darstellung =>
                        * gerade Spalten speichern
                        
0384: 2C 55 C0           BIT  PAGE2
0387: B1 19              LDA  (SCRN),Y
0389: 91 1D              STA  (LC2),Y
038B: 2C 54 C0           BIT  PAGE1
038E: B1 19     STORE1   LDA  (SCRN),Y
0390: 91 1B              STA  (LC1),Y
0392: C8                 INY
0393: D0 EA              BNE  STORE0
                        
                        * MSB erhöhen
                        
0395: E6 1A              INC  SCRN+1
0397: E6 1C              INC  LC1+1
0399: E6 1E              INC  LC2+1
039B: A5 1A              LDA  SCRN+1
039D: C9 08              CMP  #$08       ;fertig?
039F: 90 DC              BCC  STORE
03A1: 2C 81 C0           BIT  RDROM
03A4: 60                 RTS

165 Bytes
```

Dreidimensionale Funktionsdarstellung

von Alfred Böhm

Will man das Bild einer Funktion mit 2 unabhängigen Variablen in einer Zeichenebene darstellen, so muß man perspektivisch zeichnen. Dabei werden Raumpunkte mit 3 Koordinaten in Bildpunkte mit 2 Koordinaten übertragen: P(x,y,z) → Q(u,v).

Die Theorie

Um dies besser verstehen zu können, betrachten wir **Abb.2**. Sowohl für die Raum- als auch für die Bildpunkte nehmen wir an, daß sie in einem kartesischen (rechtwinkligen) Koordinatensystem dargestellt werden. Der Ursprung des räumlichen Koordinatensystems fällt dabei in der Zeichenebene mit dem Ursprung des ebenen Koordinatensystems zusammen, die u- und v-Achse des ebenen Koordinatensystems fallen mit der x- und z-Achse des räumlichen zusammen.
Wie stellt man nun einen Raumpunkt mit den Koordinaten P(x,y,z) als Bildpunkt Q(u,v) in der Ebene dar? Betrachten wir dazu folgendes Beispiel:
Um den Punkt P(1,3,3) darzustellen, gehen wir auf der x-Achse 1 Längeneinheit (LE) nach rechts, danach 3 LE in y-Richtung und daraufhin noch 3 LE in z-Richtung. In der Ebene beschreiben wir denselben Punkt durch (u,v). Die Ebenenkoordinaten (u,v) müssen wir nun aus den Raumkoordinaten (x,y,z) berechnen.

α sei der Winkel, den die x-Achse mit der y-Achse in der perspektivischen Darstellung einschließt. Dann gilt:

$u = x + x'$
$v = z + z'$

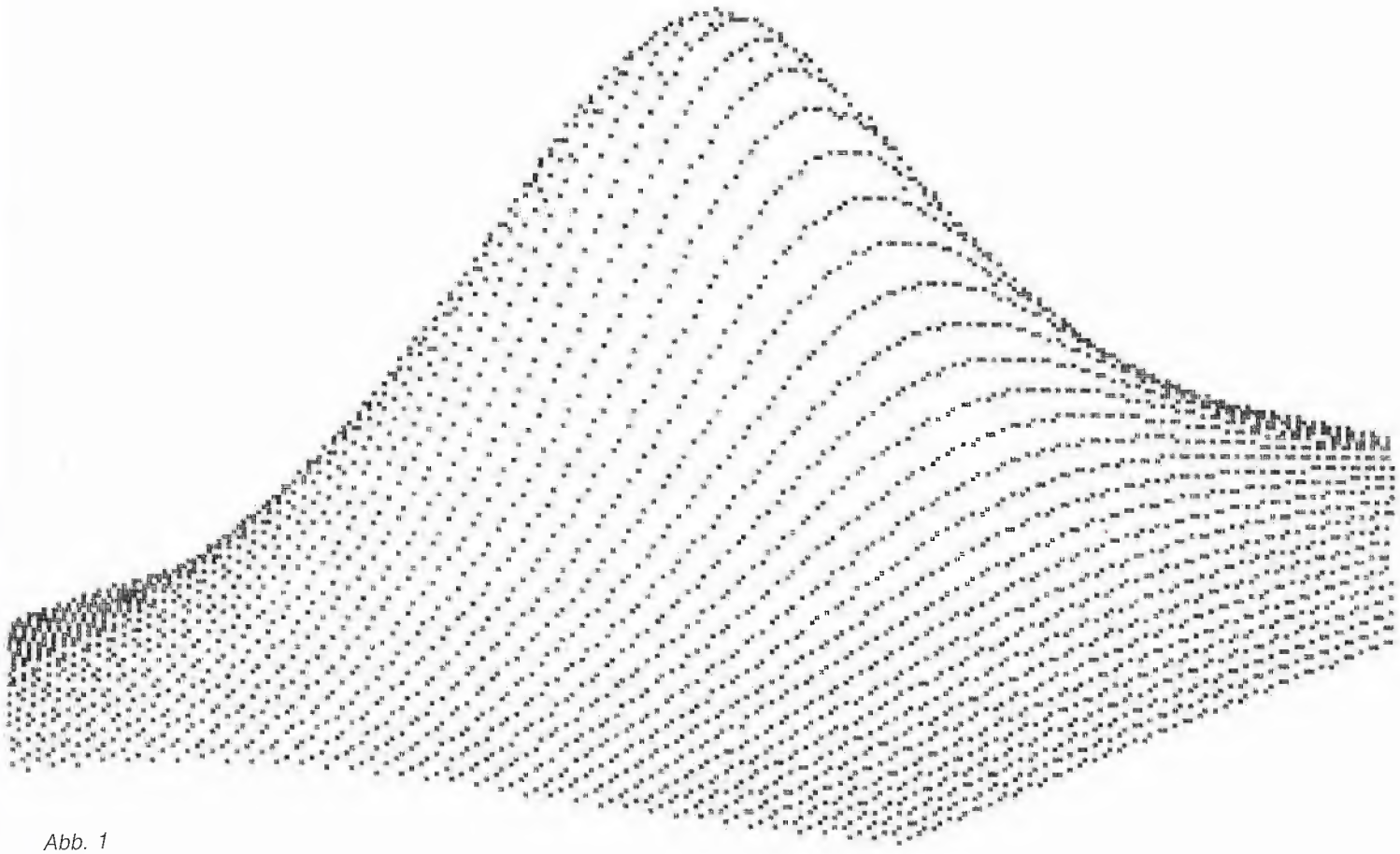


Abb. 1

ferner gilt:

$x' = y \cos \alpha$
$z' = y \sin \alpha$

In obige Gleichung eingesetzt erhält man:

$u = x + y \cos \alpha$
$v = z + y \sin \alpha$

Wie man die Lage eines einzelnen Bildpunktes nun prinzipiell berechnet, haben wir jetzt gesehen. Man könnte in das Schaubild nun natürlich noch ein Achsen-

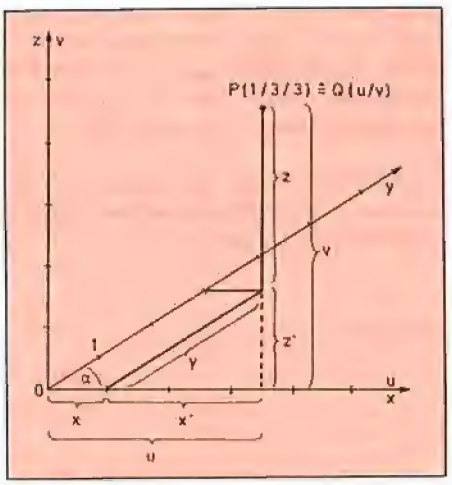


Abb. 2

kreuz einzeichnen; im folgenden Programm wurde dies jedoch aus ästhetischen Gründen unterlassen.

Das Programm

In den Programmzeilen 150-170 werden die Intervallgrenzen für die x-Werte sowie die untere Intervallgrenze der y-Werte eingegeben. Die obere Intervallgrenze der y-Werte berechnet das Programm in Zeile 270 selbst, indem die Intervallänge 7/10 der x-Intervallänge gewählt wird. Dadurch vermeidet man, daß bei zu großen y-Intervallängen der Computer unnötige Arbeit verrichten muß, weil die zugehörigen Bildpunkte gar nicht mehr auf dem Bildschirm erscheinen.
In Zeile 180 gibt man den Winkel α zwischen der x- und der y-Achse ein (siehe Abb. 1). In Zeile 210 wird die Zahl der Einheiten berechnet, welche das x-Intervall umfaßt, in 220 die Zahl der Bildschirmpunkte, die auf eine Einheit entfallen. Dadurch kann man in den Zeilen 240 und 250 die Bildschirmkoordinaten des Ursprungs berechnen. In Zeile 290 wird die Schrittweite optimiert, in welcher der Computer die Funktionswerte berechnen soll, um in möglichst kurzer Zeit ein optimales Bild zu erzeugen. Da auf der u-Achse gerade 280 Bildpunkte liegen, wird die gesamte x-Intervallänge durch 280 dividiert. Für SX = ZE / 280 würde der Computer nun die maximal mögliche Bildpunktezahl berechnen. Bei dem zu zeichnenden 3D-Bild würden die Punkte jedoch zu dicht liegen. Deshalb wird die Schrittweite noch mit dem Faktor 5 gestreckt.
In Zeile 320 werden die Anfangswerte für die Unterdrückung der unsichtbaren Bildpunkte gesetzt, in 350 wird die darzustellende Funktion Z(x,y) eingegeben. In den Zeilen 400 und 410 werden die Bildschirmkoordinaten des Bildpunktes berechnet, Zeile 420 schließt eventuell auftretende Fehlermeldungen (ILLEGAL QUANTITY) aus.
Zur Unterdrückung der unsichtbaren Bildpunkte in den Zeilen 440 und 450:
Wenn für ein festes U1 die V1-Werte unter den bisherigen V1-Werten liegen, so ist ein Bildpunkt unsichtbar, da er hinter den anderen liegt. In Applesoft-BASIC bedeutet dies jedoch, daß für ein festes U1 die V1-Werte größer sein müssen als die bisherigen Werte, wenn der Bildpunkt unsichtbar sein soll, da ja die Bildschirmkoordinaten von oben nach unten zunehmen. Wenn die Bildschirmkoordinaten eines Bildpunktes für ein festes U1 kleiner als die bisher kleinsten sind, dann ist der Bildpunkt sichtbar und VMAX(U1) wird V1 gesetzt. Ist V1 für ein festes U1 größer als das bisherige VMAX(U1), so liegt der Bildpunkt unter dem bisherigen Maximalwert und ist daher unsichtbar.

Bei dem in **Abb. 1** dargestellten Bild wurden folgende Werte eingegeben:
LX = -2, RX = 2, LY = -1.5, WI = 20.

Für den einen oder anderen Leser bliebe vielleicht noch zu erwähnen, daß er das Bild mit BSAVE BILD1, A$4000, L$2000 auf einer Diskette unter dem Namen „Bild1" abspeichern und mit
HGR2: PRINT CHR$(4);"BLOAD BILD1"
wieder auf den Bildschirm bringen kann.

Kurzhinweise

1. Zweck:
Darstellung dreidimensionaler Funktionen
2. Konfiguration:
II+ (mit G/K), IIe oder IIc;
DOS 3.3 oder ProDOS
3. Test:
RUN DARSTELLUNG.3D
4. Sammeldisk:
DARSTELLUNG.3D
(Applesoft-Programm)

DARSTELLUNG.3D

```
100 REM * Dreidimensionale Darstellung einer Funktion
110 REM * mit zwei unabhängigen Variablen
120 REM * Alfred Böhm, 1984
130 PRINT CHR$ (21): HOME : REM 80-Zeichenkarte abstellen
140 REM * EINGABETEIL *
150 INPUT "Linke Grenze für x: ";LX
160 INPUT "Rechte Grenze für x: ";RX
170 INPUT "Linke Grenze für y: ";LY
180 INPUT "Winkel zwischen x- und y-Achse          in Grad:
    ";WI
190 WI = WI * 3.14159 / 180: REM Umrechnung ins Bogenmaß
200 SI = SIN (WI):CO = COS (WI)
210 ZE = RX - LX: REM Zahl der Einheiten auf der x-Achse
220 PZ = INT (280 / ZE): REM Bildschirmpunkte für eine
    Einheit
230 REM Bildschirmkoordinaten des Ursprungs berechnen
240 U0 = INT ( - LX * PZ)
250 V0 = INT (191 + LY * SI * PZ)
260 REM Rechte Grenze des y-Intervalls berechnen
270 RY = LY + (RX - LX) * .7
280 HGR2 : HCOLOR= 7
290 SX = ZE / 280 * 5: REM Schrittweite in x-Richtung
300 SY = SX * SI: REM Schrittweite in y-Richtung
310 DIM VMAX(279)
320 FOR K = 0 TO 279:VMAX(K) = 191: NEXT K
330 FOR Y = LY TO RY STEP SY
340 FOR X = LX TO RX STEP SX
350 Z = 1 / (X * X + Y * Y + .5)
360 REM (U,V) ist die 2-dim. Abbildung von (x,y,z)
370 U = X + Y * CO
380 V = Z + Y * SI
390 REM (U1,V1) sind die Bildschirmkoordinaten von (U,V)
400 U1 = U0 + INT (U * PZ + .5)
410 V1 = V0 - INT (V * PZ + .5)
420 IF U1 > 279 OR V1 > 191 OR U1 < 0 OR V1 < 0 THEN GOTO 470
430 REM Die nächsten beiden Zeilen unterdrücken die
    unsichtbaren Punkte
440 IF V1 < VMAX(U1) THEN VMAX(U1) = V1
450 IF V1 > VMAX(U1) THEN GOTO 470
460 HPLOT U1,V1
470 NEXT X,Y
480 END
```

PEEKER Börse

Gelegenheitsanzeigen

Sie können unter dieser Rubrik zu einem besonders günstigen Preis

- Ihre Hardware und Software verkaufen
- Ihre Hard- und Software suchen
- Kontakte knüpfen und vieles mehr

Musteranzeige privat

1 Druckzeile à 32 Buchstaben nur DM 5,50 zuzügl. ges. MwSt. Mindestens 2 Druckzeilen
Beispiel:

> Verkaufe neuwertigen Typenraddrucker mit Apple-Interface. Preis auf Anfrage. Tel. 007

nur DM 18,81 inkl. MwSt.

Musteranzeige gewerblich

1 Druckzeile à 32 Buchstaben nur DM 11,– zuzügl. ges. MwSt. Mindestens 2 Druckzeilen
Beispiel:

> Neu im Angebot: Professionelle, separate Tastatur für Apple II plus 16 Funktionstasten und separatem Ziffernblock. Fa. Keyboard & Co.

nur DM 62,70 inkl. MwSt.

PEEKER

Peeker-Börse

AUFTRAG FÜR KLEINANZEIGEN

Bitte veröffentlichen Sie in der nächsterreichbaren Ausgabe nachstehenden Text unter folgender Rubrik:

☐ suche Hardware ☐ suche Software ☐ Verschiedenes
☐ biete Hardware ☐ biete Software ☐ Chiffre

Bitte jeweils 32 Buchstaben pro Zeile – einschließlich Satzzeichen und Wortzwischenräume. Bitte Absender nicht vergessen. Mindestens 2 Zeilen. Chiffregebühr DM 6,– zuzügl. MwSt.

PEEKER

Produkt-Karte

Zu der im **Peeker**, Heft _____ , Seite _____ erschienenen

Anzeige über _____

_____

bitte ich um detaillierte Information.

Ich wünsche ☐ Prospekt, Datenblatt ☐ Preisliste ☐ schriftliches Angebot ☐ tel. Rückruf

Menge	Produkt und Bestellnummer	à DM	gesamt DM

gebe ich nebenstehende Bestellung unter Anerkennung Ihrer in der Anzeige genannten Liefer- und Zahlungsbedingungen auf.

_____

Unterschrift (für Jugendliche unter 18 Jahren der Erziehungsberechtigte)

PEEKER

Info-Karte

Schreiben Sie uns, wenn Sie technische Fragen zu Aufsätzen im **Peeker** haben. Beachten Sie jedoch, daß die Redaktion keine Auskünfte über Bezugsquellen erteilt:

_____

_____

_____

_____

_____

_____

Peeker-Börse

Vorname, Name

Firma

Straße

Wohnort

PLZ/Ort

Bitte veröffentlichen Sie den umstehenden Text von _____ Zeilen à _____ DM in der nächsterreichbaren Ausgabe vom **Peeker**

Bei Angeboten: Ich bestätige, daß ich alle Rechte an den angebotenen Sachen besitze

Datum Unterschrift

Produkt-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

Anschrift der Firma angeben, bei der Sie bestellen bzw. von der Sie Informationen wünschen

Info-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

Peeker-Börse
Anzeigen-Service
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

POSTKARTE

Inserent

Straße

PLZ/Ort

POSTKARTE

Peeker
Redaktion
Dr. Alfred Hüthig Verlag
Postfach 10 28 69
6900 Heidelberg 1

Produkt-Karte

Wünschen Sie weitere Informationen zu einer der im Heft erschienenen Anzeigen?

Nichts einfacher als das. Produkt-Karte ausfüllen, frankieren und an den Inserenten (nicht an die Peeker-Redaktion) senden.

Vorher aber nicht vergessen:
Kreuzen Sie an, welchen Informationswunsch Sie haben.

Damit erleichtern Sie dem Hersteller eine gezielte Beantwortung Ihrer Anfrag

Zum Schluß tragen Sie auf der Rückseite die genaue Anschrift des Inserenten und Ihren Absender ein.

PEEKER

ucsd

Tips und Tricks in Pascal

Teil 5: Assembler-Ein/Ausgabe in UCSD-Pascal

von Dieter Geiß

Der folgende Aufsatz ist nicht nur für Assemblerfreaks gedacht. Bevor die Ein/Ausgabe in Assembler verstanden werden kann, muß zuerst die oberste Ebene in Pascal betrachtet werden. Von dieser Ebene werden wir langsam in die Ebene der Maschinensprache gleiten, um von dort aus auf alle Ein- und Ausgabeoperationen zugreifen zu können.

1. Die Pascal-Ebene

Es gibt in Pascal mehrere Ebenen, in denen sich alle Ein/Ausgabe-Operationen abspielen. Die oberste Ebene ist die Pascal-Ebene. Zu dieser gehören alle Befehle, die sich auf File-Operationen beziehen. Nachdem ein File mit dem RESET- oder REWRITE-Befehl geöffnet wurde, gibt es je nach Art des Files drei verschiedene Zugriffe auf dessen Daten. Für Files des Typs TEXT und INTERACTIVE gibt es die READ- und WRITE-Befehle. Für Files eines beliebigen Typs wie z.B. FILE OF INTEGER gibt es die GET- und PUT-Befehle, die immer einen Datensatz der Größe des entsprechenden Datentyps lesen oder schreiben. Die so definierten Files brauchen immer einen Puffer, der aber vom Compiler beim Einrichten einer solchen Variablen, z.B. durch die Definition

```
VAR F: FILE OF INTEGER;
```

bereitgestellt wird. Variablen eines solchen Files benötigen mindestens 300 Words, also 600 Bytes plus die Größe des Datentyps, also zwei Bytes für den Typ INTEGER. Der Puffer wird benötigt, damit immer ein Block eines Files von Diskette gelesen werden kann, und aus diesem Puffer solange Werte entnommen werden, bis ein neuer Block gelesen werden muß. In unserem Beispiel passen 256 INTEGER-Zahlen in einen Block, so daß beim Lesen oder Schreiben ein Laufwerk nach der 257. INTEGER-Zahl wieder anlaufen würde. Das Abspeichern der Daten im Puffer auf Diskette und das Lesen der Daten von Diskette in den Puffer muß der Programmierer nicht selbst übernehmen. Die entsprechenden Prozeduren GET und PUT, die im SYSTEM.PASCAL erklärt sind, übernehmen diese Arbeit. Sie sind recht kompliziert, was vor allem daran liegt, daß der Datentyp eines Files auch komplizierter sein kann als ein einfacher INTEGER-Typ. Bei höheren Datentypen wie ARRAYS oder RECORDS, die größer als ein Block sein können, muß die GET-Routine eventuell mehrmals auf Diskette zugreifen, bis ein Datensatz geladen wurde. Da sich das alles in Pascal abspielt, ist es entsprechend langsam.

Anders verhält es sich mit Files ohne Typ, die eine Definition wie

```
VAR F: FILE;
```

benötigen. Für sie reserviert der Compiler lediglich 40 Words, also 80 Bytes, denn sie benötigen keinen internen Puffer. Ihn muß der Programmierer zur Verfügung stellen, indem er ihn bei den beiden Befehlen BLOCKREAD und BLOCKWRITE angibt, so wie es im „Apple Pascal Language Reference Manual", Seite 43 beschrieben wird. Mit den Block-Befehlen können aber auch nur ganze Blöcke gelesen oder geschrieben werden. Die Geschwindigkeit läßt dann aber auch kaum zu wünschen übrig. Auch BLOCKREAD und BLOCKWRITE sind in Pascal implementiert, jedoch bei weitem nicht so kompliziert wie GET und PUT, was an den oben erwähnten Restriktionen liegt.

Fast eine Ebene tiefer, am Rande der Maschinensprache, aber immer noch in Pascal, finden sich die Unit-Befehle wie UNITREAD, UNITWRITE, UNITCLEAR, UNITBUSY, UNITWAIT und UNITSTATUS. Sie werden vom Pascal-System benutzt, um direkt auf eine Unit zu schreiben oder von einer Unit zu lesen. Die oben erwähnten GET, PUT, BLOCKREAD und BLOCKWRITE sind mit Hilfe der Unit-Prozeduren implementiert worden. Die Unit-Befehle sind die schnellsten Befehle für einen direkten Ein/Ausgabezugriff, wenn man nur in Pascal programmiert. Jeder dieser Befehle besteht im Prinzip aus nur einem Prozeduraufruf, der sofort vom P-Code-Interpreter ausgeführt wird und zur entsprechenden Maschinenroutine verzweigt. Als weiterer Unterschied ist zu bemerken, daß beim Benutzen der Unit-Befehle keine Ein/Ausgabe-Laufzeitfehler auftreten, im Gegensatz zu den GET-, PUT- BLOCKREAD- und BLOCKWRITE-Befehlen. Leider sind im Handbuch die Unit-Befehle nicht vollständig erklärt, obwohl man sie zur Assemblerprogrammierung genau kennen sollte.

Unitnummern

Vor der Erklärung der Prozeduren sollte man sich die Bedeutung der einzelnen Unitnummern ins Gedächtnis zurückrufen. In Pascal 1.1 gibt es zwölf und in Pascal 1.2 20 reguläre Unitnummern. Weitere 16 können selbst definiert und angeschlossen werden. Die Bedeutungen der ersten zwölf Nummern sind:

```
 #1: CONSOLE:
 #2: SYSTERM:
 #3: GRAPHIC:
 #4: Laufwerk 1 (Slot 6, Drive 1)
 #5: Laufwerk 2 (Slot 6, Drive 2)
 #6: PRINTER:
 #7: REMIN:
 #8: REMOUT:
 #9: Laufwerk 5
#10: Laufwerk 6
#11: Laufwerk 3
#12: Laufwerk 4
```

CONSOLE: ist der angeschlossene Bildschirm, wenn auf ihn geschrieben wird. Bei der Eingabe ist es die Tastatur, wobei das eingegebene Zeichen geechot wird. Die Standardfiles INPUT und OUTPUT sind an dieses Gerät angeschlossen.
SYSTERM: ist die Abkürzung für SYStem TERMinal und ist bei der Ausgabe ebenfalls der Bildschirm, bei der Eingabe die Tastatur, wobei das eingegebene Zeichen nicht geechot wird. Der Standardfile KEYBOARD ist an dieses Gerät angeschlossen.
GRAPHIC: sollte ein angeschlossenes Grafik-Terminal sein. Auf dieser Unit funktioniert allerdings nur ein einziger sinnvoller Befehl.
PRINTER: ist der angeschlossene Drukker, der nur in Slot 1 erkannt wird.
REMIN: ist die Abkürzung für REMote INput und bedeutet eine angeschlossene Modem-Karte, also z.B. die Super-Serial-Card von Apple. Unitnummer 7 ist für die Eingabe über diese Karte bestimmt. Auch ein externes Terminal kann angeschlossen werden. Die Karte wird nur in Slot 2 erkannt.
REMOUT: ist die Abkürzung für REMote OUTput und ist vorhanden, wenn REMIN: vorhanden ist. Unitnummer 8 ist für die Ausgabe über ein Modem bestimmt.
Die anderen Unitnummern bezeichnen bis zu sechs Laufwerke die in den Slots 6, 5 und 4 angeschlossen werden können. Pascal 1.2 weist weiterhin die Nummern 13 bis 20 auf, die jedoch nicht für normale Diskettenlaufwerke herangezogen werden können, sondern für große Massenspeicher bestimmt sind, z.B. Profile.

UNITREAD und UNITWRITE

Da sie dieselben Parameter haben, sollen sie gemeinsam erklärt werden. Ein Aufruf der Prozeduren hat die Form

```
UNITREAD (Nr., Ziel, Länge, Block, Modus)
UNITWRITE (Nr., Quelle, Länge, Block, Modus)
```

Dabei sind: „Unitnummer" eine der oben erwähnten Nummern, „Ziel" und „Quelle" Variablen beliebigen Typs, nicht nur des Typs „Array", wie im Handbuch fälschlicherweise angegeben. Ist die Variable jedoch ein Array, egal ob gepackt oder ungepackt, so kann der Array auch indiziert werden. Dasselbe gilt für einen String. „Länge" ist die Anzahl der Bytes, die ins „Ziel" oder von der „Quelle" übertragen werden sollen. „Block" gibt die Blocknummer an, was nur bei den Units 4, 5 und 9–12 einen Sinn ergibt, und „Modus" ist der Übertragungsmodus, der noch genauer untersucht werden soll, da auch er im Handbuch nicht bis in alle Einzelheiten erklärt ist. Die beiden letzten Parameter können auch weggelassen werden, wobei der Compiler an deren Stelle eine 0 setzt. Für verschiedene Unitnummern gibt es gewisse Unterschiede beim Aufruf von UNITREAD oder UNITWRITE.
Units 1, 2, 6, 7, 8: Der Parameter-„Block" ist ohne Bedeutung, da diese Units nicht blockorientiert sind. Die Anzahl der Bytes, die gelesen oder geschrieben werden, entspricht aber genau der Anzahl der gelesenen oder geschriebenen Bytes. UNITREAD kann durch Eingabe eines CTRL-C vorzeitig beendet werden, wenn das Modus-Wort entsprechend gesetzt wird. Beispiele:

```
VAR C: CHAR;
    S: STRING;
UNITREAD (2, C, 1);
{liest ein Zeichen von Tastatur ohne Echo}
S := 'Ausgabe einer Zeichenkette';
UNITWRITE (1, S [1], length (S));
{gibt den String S auf Bildschirm aus}
UNITWRITE (6, S [1], length (S));
{gibt S auf den Drucker aus}
UNITREAD (7, S [1], 80);
{liest von Modem maximal 80 Zeichen}
UNITWRITE (8, C, 1);
{gibt ein Zeichen an Modem aus}
```

Units 4, 5, 9–12: Die Blocknummer gibt die absolute Blocknummer der Diskette an. Beim Lesen entspricht „Länge" der Anzahl der tatsächlich gelesenen Bytes, wenn kein Disk-Fehler aufgetreten ist. Beim Schreiben jedoch muß man Vorsicht walten lassen. Es wird immer mindestens ein Sektor auf die Diskette geschrieben – d.h. es ist unmöglich, z.B. drei Bytes auf Diskette zu schreiben, ohne daß die anderen Bytes dieses Sektors überschrieben werden. Da ein Sektor immer 256 Bytes hat, sollte man seinen Puffer immer mindestens in dieser Größe wählen, noch besser ist ein Puffer, der eine vielfache Größe von 512 Bytes hat. Beispiele:

```
VAR DIR  : DIRECTORY;
  {Definition wie in Peeker 1/2 85}
    BLOCK: PACKED ARRAY [0..511] OF CHAR;
UNITREAD (4, DIR, SIZEOF (DIR), 2);
  {liest Inhaltsverzeichnis
    von Boot-Diskette ab Block 2 }
UNITWRITE (5, BLOCK, SIZEOF (BLOCK), 279);
  {schreibt einen Block von Daten
    auf den letzten Block auf Diskette}
```

Das Modus-Byte, das nur für nicht-blockstrukturierte Units Bedeutung hat, muß in einzelne Bits unterteilt werden:

Bit 0 und 1: keine Bedeutung
Bit 2 = 0: Bei der Ausgabe werden DLE-Zeichen konvertiert, d.h. das nach dem DLE folgende Zeichen wird, nach Abzug der Konstanten 32, als Anzahl der Leerzeichen interpretiert. In diesem Format werden führende Leerzeichen bei Textfiles komprimiert abgespeichert. Bei der Eingabe wird ein EOF (CTRL-C) erkannt und das Lesen vorzeitig abgebrochen.
Bit 2 = 1: DLE-Umwandlung und EOF-Erkennung werden nicht durchgeführt.
Bit 3 = 0: Nach einem Wagenrücklauf (ASCII CR) wird zusätzlich ein Zeilenvorschub (ASCII LF) an das ausgebende Gerät gesandt.
Bit 3 = 1: Es wird kein Zeilenvorschub an das ausgebende Gerät gesandt.
Bit 4 = 0: Bei der Eingabe werden die Sonderzeichen CTRL-A, Z, K, W und E abgefangen, die beim Apple ohne 80-Zeichenkarte benutzt werden, um Groß- und Kleinschrift zu simulieren.
Bit 4 = 1: Die Sonderzeichen werden nicht abgefangen.
Bit 5 = 0: Bei der Ein- oder Ausgabe werden die drei Sonderzeichen CTRL-S (Stopp der Ein- und Ausgabe) und CTRL-F (Unterdrücken (Flush) der Ausgabe) sowie CTRL-§ (ASCII NUL) abgefangen.
Bit 5 = 1: Die drei Sonderzeichen werden nicht abgefangen.

Um die Werte für das Modus-Wort zu berechnen, muß man sich überlegen, welche Bits man setzen will. Diese multipliziert man mit ihrer Wertigkeit und addiert sie auf. Lassen Sie sich nicht von den Angaben im Apple-Pascal-Handbuch täuschen. Die Werte, die dort angegeben sind, sind ganz einfach falsch. Beispiele:
Nach einem CR soll kein Zeilenvorschub ausgegeben werden, => Bit 3 = 1 => Modus ist 2 ↑ 3 = 8.
Wie oben und zusätzlich soll die DLE-Umwandlung nicht stattfinden => Bit 2 = 1 und Bit 3 = 1 => Modus ist 2 ↑ 2 + 2 ↑ 3 = 12.
CTRL-S soll nicht abgefangen werden => Bit 5 = 1 => Modus = 2 ↑ 5 = 32. **Listing 1** demonstriert dieses Beispiel, bei dem nicht wie sonst üblich mittels CTRL-S die Ausgabe angehalten werden kann.
Setzt man alle Bits, gibt also den Modus-Wert 60 an, so kann ein Zeichen ohne störende Umwandlungen eingelesen werden, was quasi einem LDA $C000 gleichkommt. **Listing 2** zeigt ein Programm, mit dem beliebige Zeichen an den Drucker geschickt werden können, auch die Steuerzeichen, die sonst nicht verfügbar sind.

UNITCLEAR

Die Prozedur UNITCLEAR hat nur eine initialisierende Bedeutung.
UNITCLEAR (1) oder UNITCLEAR (2)
setzt den Type-ahead-Buffer zurück und setzt den Bildschirm auf die linke Bildschirmhälfte, falls ohne 80-Zeichenkarte gearbeitet wird.
UNITCLEAR (3)
schaltet auf den Text-Modus zurück, falls der Grafik-Modus eingeschaltet war.
UNITCLEAR (4)
hat keine besondere Bedeutung, außer

daß man damit testen kann, ob sich ein Disk-Controller in einem der Slots 6, 5 und 4 befindet.
UNITCLEAR (6)
springt die Init-Routine des Druckers an. Es kommt auf das Interface in Slot 1 an, ob eine Aktion eingeleitet wird. Eine Aktion könnte das Löschen des Druckerpuffers sein.
UNITCLEAR (7) oder UNITCLEAR (8)
springt die Init-Routine des Modems an. Es kommt auf das Interface in Slot 2 an, ob eine Aktion eingeleitet wird.

Mit Hilfe von UNITCLEAR kann man leicht nachprüfen, ob eine Unit angeschlossen ist. Hat nämlich ein nachfolgender IORESULT-Aufruf einen von Null verschiedenen Wert, so kann daraus geschlossen werden, daß diese Unit nicht angeschlossen ist. Beim Wert Null kann dagegen bei blockstrukturierten Units nicht angenommen werden, daß sie angeschlossen sind. Das liegt daran, daß pro Interface-Karte zwei Laufwerke erlaubt sind. Ist nur ein Laufwerk angeschlossen, so erhält man bei UNITCLEAR (4), aber auch bei UNITCLEAR (5), ein IORESULT von Null. Nur ein nachfolgendes UNITREAD kann Aufschluß geben, ob ein Laufwerk angeschlossen ist oder nicht. **Listing 3** zeigt ein kleines Programm, das alle angeschlossenen Units zeigt.

UNITBUSY

Ein Aufruf von UNITBUSY ist am Apple ziemlich sinnlos. Die Funktion soll anzeigen, ob ein Gerät beschäftigt ist oder nicht. Da die Gerätetreiber des Apple aber nicht interrupt-gesteuert sind, gibt diese Funktion immer den Wert FALSE zurück.

UNITWAIT

Diese Prozedur wartet, bis ein Gerät seine Ein- oder Ausgabe beendet hat. Aus demselben Grund wie bei UNITBUSY hat die Prozedur allerdings keine Bedeutung.

UNITSTATUS

Um den Status der angeschlossenen Geräte zu überprüfen, ruft man UNITSTATUS mit drei Parametern auf:

```
UNITSTATUS (Unitnummer, Ziel, Kontrollwort)
```

Dabei beschreibt „Unitnummer" das Gerät, „Ziel" ist eine Variable beliebigen Typs, zumeist aber ein Array, der dann indiziert werden kann. „Kontrollwort" gibt die Arbeitsweise der Prozedur an. Im Kontrollwort sind nur die Bits 0 und 1 relevant. Es bedeuten:

Bit 0 = 0: Der Ausgabekanal wird befragt.
Bit 0 = 1: Der Eingabekanal wird befragt.
Bit 1 = 0: Der Aufruf hat die Wirkung einer Statusabfrage.
Bit 1 = 1: Der Aufruf hat die Wirkung eines Kontrolleingriffs.

Wie man sieht, legt Bit 0 fest, welcher Kanal eines Geräts, z.B. eines Modems, befragt oder kontrolliert werden soll, während Bit 1 bestimmt, ob nur eine Abfrage stattfinden oder ein anderer Zustand des Geräts eingestellt werden soll. Für Geräte, die fest am Apple angeschlossen sind, also CONSOLE:, SYSTERM:, GRAPHIC: und die Laufwerke, ist es einerlei, wie das Kontrollwort aussieht, da man bei diesen Geräten nichts kontrollieren kann. Nur wenn man eigene Treiber für neue Geräte anschließt, kann man den Bits im Kontrollwort Beachtung schenken. Beispielsweise könnte ein Programmierer durch einen Aufruf von

```
A := 1;
UNITSTATUS (9, A, 2);
```

und einem angeschlossenen Gerätetreiber für eine RAM-Karte als Pseudodisk für Unit 9 erreichen, daß diese auf „schreibgeschützt" gestellt wird. Selbstverständlich muß der Treiber den UNITSTATUS-Aufruf richtig verstehen und ausführen. Für die standardmäßig angeschlossenen Geräte ergeben sich folgende Funktionen:

```
VAR
Status : RECORD
         Anzahl          : INTEGER;
         Offener_Apfel   : BOOLEAN;
         Geschl_Apfel    : BOOLEAN;
         Shift_Taste     : BOOLEAN
         END;

UNITSTATUS (1, Status, 1) oder
UNITSTATUS (2, Status, 1);
```

Status.Anzahl gibt die Zahl der sich in der Warteschlange befindenden, noch nicht abgearbeiteten Zeichen an.
Die drei anderen Variablen werden nur bei Pascal 1.2 gesetzt und geben an, ob die Offene-Apfel-Taste, die Geschlossene-Apfel-Taste oder die Shift-Taste gedrückt wurde. Letztere kann nur mit eingebauter „Shift-Key-Modification" erkannt werden. Es wird der Eingabekanal befragt (Kontrollwort = 1), was aber nur bei Pascal 1.1 nötig ist. Bei Pascal 1.2 wird kein Wert auf das Kontrollwort gelegt.

```
VAR
DiskStatus: RECORD
            Anz_gepuff_Bytes   :INTEGER;
            Bytes_pro_Sektor   :INTEGER;
            Sektoren_pro_Spur  :INTEGER;
            Spuren_pro_Disk    :INTEGER
            END;

UNITSTATUS (4, DiskStatus, 0);
```

Die Definitionen im Record DiskStatus sind wohl selbsterklärend. Die Anzahl der gepufferten Bytes ist beim Apple immer 0. Beim Multiplizieren von Bytes_pro_Sektor mit Sektoren_pro_Spur und Spuren_pro_Disk ergibt sich die schon fast legendäre Zahl 143360, was genau der Speicherkapazität eines normalen Apple-Laufwerks in Bytes entspricht.

Für PRINTER:, REMIN: und REMOUT: wird bei einem Aufruf von

```
VAR I : INTEGER;
UNITSTATUS (6, I, 0);
```

lediglich die Variable I auf Null gesetzt.
Mit diesem Wissen ausgerüstet können wir getrost eine Ebene tiefer gelangen.

2. Die RSP-Ebene

RSP heißt **Runtime-Support-Package** und ist das Bindeglied zwischen der Pascal-Ebene und der untersten, der BIOS-Ebene (Basic Input Output System). Die Aufgabe des RSP ist es, die von Pascal angelieferten Parameter für das BIOS mundgerecht aufzubereiten. Das RSP ist wie das BIOS ein Teil des Interpreters und in Maschinensprache geschrieben. Es ist aber noch nicht die Ebene, in der eigene Maschinenprogramme Ein- oder Ausgabe-Routinen anspringen können. Um zu verstehen, was das RSP leistet, verfolgen wir einen UNITWRITE-Befehl von der Pascal- zur RSP-Ebene. Wir untersuchen die Pasacl-Zeile

```
UNITWRITE (2, A [3], 1, 0, 60);
```

Dabei sei A eine globale Variable vom Typ PACKED ARRAY [0..7] OF CHAR. Wie der P-Code dieser Zeile aussieht, mag uns an dieser Stelle nicht so genau interessieren. Wir wissen, daß Pascal eine Kellersprache ist und die Parameter einer Prozedur auf den Stack geschoben werden, bevor die Prozedur aufgerufen wird. Es befinden sich also auf dem Stack ($0100-Bereich):

```
2
Adresse von A
3
1
0
60 <-- TOS (Top of Stack)
```

Der dritte Parameter, die 3, ist der Index in die Variable A. Er ist meist Null, es sei denn, es wurde ein PACKED ARRAY OF CHAR als Quelle angenommen. Dann gibt der dritte Parameter den Index an, falls der Array indiziert wurde. Der Grund für die Einführung des unsichtbaren dritten Parameters ist darin zu suchen, daß in UCSD-Pascal keine ungeraden Adressen übergeben werden können, da sich alles auf Wortgrenzen abspielt.
Man beachte, daß der 6502-Stack nach unten wächst und das nächste PLA den Wert 60 vom Stack holen würde. Alle Parameter sind hier 2 Bytes groß. Mit diesen Werten auf dem Stack trifft der P-Code-Interpreter auf den Befehl $9E, also CSP (Call Standard Procedure), mit dem nachfolgenden Parameter $05 (UNITWRITE). Nach einem kurzen Blick in die P-Code-Tabelle und dann in die CSP-Tabelle verzweigt der Interpreter ins RSP. Dort wird zuerst der Modus-Parameter vom Stack geholt. Die einzelnen Bits werden interpretiert und die entsprechenden Flags für DLE-Umwandlung, LF nach CR usw. gesetzt. Dann werden die anderen Parameter in Zero-Page-Variablen abgespeichert. Ist die Länge kleiner oder gleich Null, so wird der UNITWRITE-Befehl abgebrochen, und es wird zur Interpreter-Hauptschleife gesprungen. Im anderen Fall wird nun die Gerätenummer auf ihre Gültigkeit abgecheckt. Ist sie ungültig (z.B. 43 oder 160), so wird der I/O-Fehlercode 2 (Bad Device) in eine Variable geschrieben, die der Programmierer mit der Funktion IORESULT abtesten kann. Schließlich wird getestet, um welches Gerät es sich handelt, also ob ein Diskettenzugriff, ein Zugriff auf ein User-Device oder ein Gerät ohne Block-Struktur vorliegt. Ein User-Device ist ein Gerät, das ein Programmierer mittels SYSTEM.ATTACH selbst anschließen kann, wenn er einen entsprechenden Treiber geschrieben hat. Danach wird zur entsprechenden gerätebehandelnden Routine gesprungen: Bei einem User-Device zum selbst installierten Treiber, bei einem normalen Laufwerk zu den eingebauten Disk-Treibern, bei einem selbst angeschlossenen Laufwerk, wie z.B. einer RAM-Disk, zu dem entsprechenden RAM-Disk-Treiber. Bei einem Gerät ohne Block-Struktur, wie in unserem Beispiel, wird nun folgendes gemacht:
In Abhängigkeit von der Unitnummer werden die Adressen für die entsprechenden Treiber (Bildschirm, Tastatur, Drucker, Modem) in Zero-Page-Variablen abgelegt. Falls eine der Adressen 0 ist, wie z.B. beim Lesen vom Gerät „Drucker", wird ein IORESULT von 3 abgelegt (Illegal Request). Ansonsten werden die Zeichen der Reihe nach gelesen oder geschrieben. Dabei werden alle gesetzten Flags berücksichtigt, also das Ausgeben eines LF nach einem CR, DLE-Umwandlung und das Erkennen von EOF bei einer Eingabe, was eine der Hauptaufgaben des RSP ist. Das RSP kann nicht von Assemblerprogrammen angesprungen werden, da es nicht als Unterprogramm vorliegt, sondern direkt zur P-Code-Interpreter-Hauptschleife zurückspringt. Die Treiber selbst liegen jedoch als Unterprogramme vor und können von einem Assemblerprogrammierer angesprungen werden. Dies bringt uns zur BIOS-Ebene.

3. Die BIOS-Ebene

Eines der großen Probleme bei der Einbindung von Assemblerprogrammen in Pascal war schon immer, daß es keinerlei Möglichkeit der Ein/Ausgabe im Assemblerprogramm selbst gab. Unter DOS 3.3 war die Sache anders. Man mußte nur die schon wohlbekannten Adressen $36, $37, $38, $39 kennen, um zu wissen, wohin man springen mußte, wenn man ein Zeichen ein- oder ausgeben wollte. Um auf Diskette zu schreiben, sprang man in die RWTS, deren Adresse und geforderten Parameter man ebenfalls kannte. Auch

den Drucker konnte man bedienen, wenn man wußte, um welches Interface es sich handelte.
In Pascal sind diese Aufgaben auch nicht weiter schwierig, wenn man nur die Adressen und die Parameter der Treiber kennt. Im Pascal werden die verschiedenartigsten Interface-Karten für Drucker, 80-Zeichen oder Modems erkannt, so daß es hier nicht ratsam erscheint, eigene Treiber für spezielle Karten zu schreiben, sondern die Treiber zu benutzen, die schon implementiert sind.
Das BIOS befindet sich gänzlich auf der Bank 2 der 16K-Karte. Da der P-Code-Interpreter sich auf der anderen Bank befindet, welche normalerweise immer aktiv ist, muß für einen BIOS-Aufruf kurz umgeschaltet werden. Man muß dies aber nicht selbst erledigen. In der letzten Page befinden sich Sprungbefehle zu allen BIOS-Funktionen. Bevor die entsprechende Funktion aufgerufen wird, wird auf Bank 2 umgeschaltet und danach wieder auf Bank 1. Die erste Tabelle mit Sprungbefehlen wird auch BIOS-BEFORE-FOLD-TABLE genannt. Sie liegt ab der Adresse $FF00 vor. Ab Adresse $FF59 steht die Tabelle, die auch BIOS-AFTER-FOLD-TABLE genannt wird. Diese sollte man aber nicht anspringen, es sei denn, man möchte selbst das Bank-Switching übernehmen. Die beiden Adressen sind auch in den Zero-Page-Variablen $EC, $ED und $EE, $EF abgespeichert. Dies gilt sowohl für Pascal 1.1 als auch für Pascal 1.2. **Tabelle 1** zeigt die Adressen und die Bedeutung der BIOS-BEFORE-FOLD-TABLE. Nun ist es nicht mehr schwierig, von Maschinensprache aus z.B. einzelne Zeichen an den Bildschirm zu bringen. Wichtig ist nur, welche Parameter auf dem Stack und in den Registern liegen. Für alle Aufrufe gilt aber folgendes: Im X-Register sollte der Code für die Art des Aufrufs sein. Für diesen Code gilt
0 = read
1 = write
2 = init
4 = status
Nach dem Aufruf steht im X-Register das Ergebnis für IORESULT. Man kann davon ausgehen, daß nach einem BIOS-Aufruf alle Register zerstört sind. Braucht man sie nach einem Aufruf noch, müssen sie vorher gerettet werden. Wir gehen alle Prozeduren durch und geben für jede wichtige Prozedur ein Beispiel an.

$FF00: CONSOLE READ

Ein Zeichen wird von der Tastatur gelesen, welches sich danach im Akkumulator befindet. Da der Aufruf nicht über das RSP geht, müssen die Flags für die Sonderzeichen CTRL-S, CTRL-F und CTRL-§ sowie die speziellen Zeichen für den Betrieb ohne 80-Zeichenkarte selbst gesetzt werden. Die Flags befinden sich bei Pascal 1.1 und 1.2 an der Stelle $BF1C. Die Variable nennt sich SPCHAR. Dabei gilt:

Bit 0 = 0: Die speziellen Zeichen für den 40-Zeichen-Bildschirm werden abgefangen.
Bit 0 = 1: Die speziellen Zeichen werden nicht abgefangen.
Bit 1 = 0: Die Kontroll-Zeichen CTRL-S, CTRL-F und CTRL-§ werden abgefangen.
Bit 1 = 1: Die Kontroll-Zeichen werden nicht abgefangen.

Will man also ein reines Zeichen von der Tastatur lesen, muß man eine 3 (Bit 0 und 1 gesetzt) in die Variable SPCHAR setzen. Vorher sollte man sich aber ihren alten Wert merken und nach dem Aufruf wieder restaurieren. Ein Beispiel für das Lesen eines Zeichens von der Tastatur ist in **Listing 4** dargestellt. Es werden alle Tasten angenommen, da die Bits für die speziellen Zeichen eingeschaltet werden.
An dieser Stelle muß ich etwas zu den Listings 4ff. sagen. Die Beispiele sind alle als Maschinenprozeduren geschrieben, die eine Definition
.PROC
am Anfang tragen. Um sie auszuführen, geben Sie diese mit dem Editor ein und speichern sie unter einem Namen, z.B. TEST1.TEXT. Sie brauchen kein Pascal-Rahmenprogramm, um die Testprozeduren auszuführen. Es steht zwar in keinem Handbuch, aber Maschinenprozeduren können direkt ausgeführt werden. Sie müssen sie zuvor aber noch linken, damit der Linker einen ausführbaren Codefile daraus macht. Als Hostfile geben Sie TEST1 an, und bei der Frage nach den Library-Files und dem Map-File drücken Sie einfach die RETURN-Taste. Auf die Frage nach dem Output-File geben Sie TEST1 oder RETURN ein. In letzterem Fall wird der ausführbare Codefile unter dem Namen SYSTEM.WRK.CODE auf der Boot-Diskette angelegt. Der gelinkte Codefile ist ein reines Assemblerprogramm und sofort ausführbar.

$FF03: CONSOLE WRITE

Das Zeichen im Akkumulator wird an den Bildschirm ausgegeben. An welche Stelle das Zeichen kommt, hängt von den Inhalten der Speicherzellen CH (Cursor Horizontal) und CV (Cursor Verical) ab, die sich bei verschiedenen 80-Zeichenkarten an unterschiedlichen Plätzen befinden können. Meistens liegen sie in den „Screenholes". Ist keine 80-Zeichenkarte angeschlossen, so liegt CH an der Stelle $00F4 und CV an der Stelle $00F5. Da die Ausgabe nicht über das RSP geht, müssen die DLE-Umwandlung sowie das Senden eines LF nach einem CR selbst übernommen werden, falls dies erwünscht ist. Ein Beispiel für CONSOLE WRITE ist in **Listing 5** gegeben.

$FF06: CONSOLE INIT

Das Eingabegerät wird initialisiert, was einem UNITCLEAR (1) entspricht. Auf dem Stack müssen die Break-Adresse und die Syscom-Adresse liegen. Diese befinden sich im Inhalt der Speicherzellen $BF1D und $BF1E (IBREAK) sowie $BF1F und $BF20 (ISYSCOM). Siehe auch **Listing 6**.

$FF2A: CONSOLE STATUS

Der Status des Eingabegeräts wird festgestellt. Vor dem Aufruf müssen folgende Parameter auf den Stack gelegt werden (vgl. UNITSTATUS):

Adresse des Puffers für die Information
Kontrollwort
Rückkehradresse <-- TOS

Die Rückkehradresse kommt durch ein JSR automatisch auf den Stack (siehe **Listing 7**).

$FF21: GRAPHIC WRITE

Ein Zeichen sollte an den Hires-Schirm ausgegeben werden. Diese Routine ist weder in Pascal 1.1 noch Pascal 1.2 implementiert. Sie führt zu einem RTS.

$FF24: GRAPHIC INIT

Der Text-Modus wird eingeschaltet.

$FF27: PRINTER READ

Da vom Drucker nichts gelesen werden kann, führt ein Sprung zu dieser Adresse in ein RTS.

$FF09: PRINTER WRITE

Das Zeichen im Akkumulator wird an den Drucker geschickt. Da der Sprung über das Interface und nicht direkt an den Drukker geht, ist es optimal, auf diese Weise Zeichen an den Drucker zu schicken (Interface-Unabhängigkeit). Für DLE-Umwandlung und CR-Problem gilt das gleiche wie für CONSOLE WRITE. Beispiel siehe **Listing 8**.

$FF0C: PRINTER INIT

Der Drucker wird initialisiert. Dies gilt nur für serielle Karten, Firmware-Karten und Communication-Karten. Parallel-Interfaces werden nicht angesprungen.

$FF2D: PRINTER STATUS

Der Status des Druckers wird festgestellt. Für die Parameter gilt das gleiche wie für

CONSOLE STATUS. Es werden lediglich die ersten beiden Bytes des Puffers auf Null gesetzt.

$FF18: REMOTE READ
Ein Zeichen wird von der Schnittstelle in Slot 2 gelesen, welches sich danach im Akkumulator befindet. Als Beispiel kann Listing 4 herangezogen werden. Es muß lediglich CONSOLE READ durch REMOTE READ ersetzt werden.

$FF1B: REMOTE WRITE
Das Zeichen im Akkumulator wird an die Schnittstelle in Slot 2 ausgegeben. Ansonsten gilt das gleiche wie für PRINTER WRITE. Als Beispiel kann Listing 5 herangezogen werden, wenn CONSOLE WRITE durch REMOTE WRITE ersetzt wird.

$FF1E: REMOTE INIT
Die Schnittstelle in Slot 2 wird initialisiert. Es gilt dasselbe wie für PRINTER INIT.

$FF33: REMOTE STATUS
Der Status der Schnittstelle in Slot 2 wird festgestellt – es gilt das gleiche wie für PRINTER STATUS.

$FF12: DISK READ
Eine bestimmte Anzahl von Bytes wird von einer physikalischen Diskettenstation gelesen. Auf dem Stack müssen sich folgende Parameter befinden (vgl. UNITREAD):

Modus
Blocknummer
Länge
Pufferadresse
Laufwerksnummer
Rückkehradresse <-- TOS

Der Modus hat hier keine Bedeutung, die Länge kann nicht größer als 32767 sein. Für die Laufwerksnummer gilt folgende Beziehung:
0 = #4:
1 = #5:
2 = #11:
3 = #12:
4 = #9:
5 = #10:
Die Rückkehradresse kommt durch ein JSR $FF12 automatisch auf den Stack. Ein Beispiel für DISK READ und DISK WRITE ist in **Listing 9** gegeben.

$FF0F: DISK WRITE
Eine bestimmte Anzahl von Bytes wird auf eine physikalische Diskettenstation geschrieben. Für die Parameter gilt das gleiche wie für DISK READ. Die Länge sollte durch 256 teilbar sein (vgl. UNITWRITE).

$FF15: DISK INIT
Es wird getestet, ob sich ein Disk-Interface im durch den Akkumulator angegebenen Slot befindet. Falls dies nicht der Fall ist, wird im X-Register der IORESULT 9 zurückgegeben, 0 sonst. Für die Slotnummern gelten folgende Zuweisungen:
Slot 6: Akku = 0
Slot 5: Akku = 1
Slot 4: Akku = 2

$FF30: DISK STATUS
Der Status eines Laufwerks wird bestimmt. Für die Parameter gilt das gleiche wie für CONSOLE STATUS. Die zurückgegebene Information entspricht der durch UNITSTATUS erhaltenen Information.

$FF36: CONSOLE CHECKING
Die Tastatur wird abgefragt. Es müssen keine Parameter übergeben werden. Ein Aufruf zerstört keine Register und kann benutzt werden, um es in rechenintensiven Programmen dem Benutzer zu ermöglichen, Zeichen in die Tastatur-Warteschlange einzufügen. CONSOLE CHEKKING wird bei jeglicher Ein/Ausgabe vom BIOS aufgerufen. Ein Assemblerprogrammierer wird allerdings gebeten, zur Adresse $BF0A zu springen, damit sichergestellt ist, daß CONSOLE CHECKING immer korrekt aufgerufen wird, da sich dessen Adresse einmal ändern könnte. Die Applestuff-Unit KEYPRESS ruft CONSOLE CHECKING auf.

$FF39: USER READ/WRITE/INIT/ STATUS
Bei jeder Ein/Ausgabeoperation von selbst angeschlossenen Geräten wird die Adresse $FF39 angesprungen. Das X-Register muß den „read/write/init/status"-Code wie oben beschrieben enthalten. In Abhängigkeit des Aufrufs müssen die Parameter wie bei DISK READ, DISK WRITE, DISK INIT und DISK STATUS auf dem Stack vorhanden sein. Im Akkumulator muß sich die Gerätenummer befinden, die von 128 bis 143 laufen kann. Die Adressen für die Treiber müssen sich ab der Adresse $FE80 (jeweils drei Bytes) befinden, z.B.
FE80: 4C 00 C2 ;JMP $C200
für ein Gerät mit der Unitnummer 128, dessen Treiber sich im RAM in Slot 2 befindet (wie z.B. eine Maus).

$FF3C: PSUBDRIVE
Pseudo-Disks (RAM-Disks) können angeschlossen werden, wenn die Anfangsadresse des Treibers eine der normalen Disknummer-Vektoren ab Adresse $FEB0 ersetzt. Dann wird vom RSP die Adresse $FF3C angesprungen, um diesen neuen Treiber zu handhaben. Der Anschluß neuer Treiber soll aber nicht Thema dieses Artikels sein, das würde hier zu weit führen. Es sei hier nur gesagt, daß im Akkumulator die Unitnummer stehen muß, im Y-Register die doppelte Unitnummer und im X-Register wie üblich der „read/write/init/status"-Code.

In eigener Sache

Zwei Dinge möchte ich noch nachtragen. Zum Ersten bedanke ich mich für die positive Resonanz auf meine Serie, die sich in vielen Leserbriefen wiederfindet. An dieser Stelle muß ich mich auch dafür entschuldigen, daß Sie, liebe Peeker-Leser, zwei Monate auf weitere Teile meiner Serie verzichten mußten. Ich hatte aus mehreren Gründen wie das Schreiben eines Buches und das Lernen auf wichtige Prüfungen keine Zeit, daran weiterzuarbeiten, werde aber versuchen, keine Lücken mehr im Peeker zu hinterlassen.
Die zweite Sache betrifft den Leserbrief von Herrn Adrian Meissner aus Heft 12/85. Herr Meissner vermißt in meinen Artikeln Hinweise auf die Quellen meines Wissens. Als Bücher werden von mir lediglich die beiden Pascal-Manuals der Firma Apple genannt. Leider muß ich zugeben, daß ich tatsächlich keine andere Literatur kenne oder benutzt habe, mit deren Hilfe ich mir mein Wissen aneignen konnte. Seit etwa vier Jahren arbeite ich mit dem Apple-Pascal-Betriebssystem. Nachdem die Handbücher nicht mehr genügend Information lieferten, untersuchte ich den P-Code-Interpreter und baute mir einen P-Code-Disassembler, mit dem ich Codefiles genau untersuchen konnte. Nach vielen Monaten Übung war es mir möglich, beliebige Codefiles zurückzuübersetzen, so daß ich mir Source-Files erstellen konnte. Als erstes übersetzte ich den File SYSTEM.PASCAL, der mir einen tiefen Einblick in das Wesen des in Pascal geschriebenen Pascal-Systems erlaubte. Später untersuchte ich den P-Code-Interpreter von Pascal 1.2 (128K-Version), auf dessen Wissen der Cruncher von Teil 3 aufbaut. Ich bedauere, daß ich, was Literatur angeht, Herrn Meissner nicht weiterhelfen kann und werde versuchen, in meiner Serie die größten Löcher zu stopfen. Wenn Sie spezielle Fragen haben, dann schreiben Sie doch eine kurze Notiz an die Redaktion. Ich werde versuchen, die Fragen in Gebiete einzuteilen, so daß sie in einem Teil meiner Serie bearbeitet werden können.
Hinweis: Im nächsten Beitrag soll der Aufbau von Files dargelegt werden.

Beispielprogramme

Listing 1:

```
program Listing1;
const CONSOLE = 1;
var   S: string;
      I: integer;

begin {Listing1}
  S:= 'Dies ist ein kurzer Text. ';
  S [length (S)]:= chr (13);
  for I:= 1 to 40 do
  unitwrite (CONSOLE, S [1], length (S), 0, 32)
end {Listing1}.
```

Listing 2:

```
program Listing2;

const SYSTERM = 2;
      PRINTER = 6;
var   C: char;

begin {Listing2}
  unitclear (6);
  if IOresult <> 0
  then writeln
  (chr (7), 'Kein Drucker angeschlossen')
  else
    begin
      C:= chr (0);
      repeat
        unitread (SYSTERM, C, 1, 0, 60);
        unitwrite (PRINTER, C, 1, 0, 60);
        writeln (ord (C))
      until ord (C) = 0
    end {else}
end {Listing2}.
```

Listing 3:

```
program Listing3;

const CONSOLE =  1;
      SYSTERM =  2;
      GRAPHIC =  3;
      DISK1   =  4;
      DISK2   =  5;
      PRINTER =  6;
      REMIN   =  7;
      REMOUT  =  8;
      DISK5   =  9;
      DISK6   = 10;
      DISK3   = 11;
      DISK4   = 12;
var   Unitnum : integer;
      Dummy   : char;

begin {Listing3}
  writeln ('Folgende Geraete',
   'sind angeschlossen:');
  writeln;
  for Unitnum:= CONSOLE to DISK4 do
  begin
    if Unitnum in [DISK1, DISK2,
     DISK3, DISK4, DISK5, DISK6]
    then unitread (Unitnum, Dummy, 1)
    else unitclear (Unitnum);
    if IOresult = 0 then
    begin
      write ('#', Unitnum: 2, ' = ');
      case Unitnum of
        CONSOLE: writeln ('CONSOLE:');
        SYSTERM: writeln ('SYSTERM:');
        GRAPHIC: writeln ('GRAPHIC:');
        DISK1  : writeln ('DISK1:');
        DISK2  : writeln ('DISK2:');
        PRINTER: writeln ('PRINTER:');
        REMIN  : writeln ('REMIN:');
        REMOUT : writeln ('REMOUT:');
        DISK5  : writeln ('DISK5:');
        DISK6  : writeln ('DISK6:');
        DISK3  : writeln ('DISK3:');
        DISK4  : writeln ('DISK4:')
      end {case}
    end {if}
  end {for}
end {Listing3}.
```

Listing 4:

```
                .PROC TEST1
CONSOLE_READ    .EQU OFF00
SPCHAR          .EQU 0BF1C
SAVESPCHAR      .EQU 00
LETTER          .EQU 01
                 LDA SPCHAR       ;rette altes SPCHAR
                 STA SAVESPCHAR
                 LDA #03          ;alle Flags gesetzt
                 STA SPCHAR
                 LDX #00          ;X = 0 => read
                 JSR CONSOLE_READ
                 STA LETTER       ;Zeichen abspeichern
                 LDA SAVESPCHAR   ;alte SPCHAR zurueckspeichern
                 STA SPCHAR
                 RTS
                .END
```

Listing 5:

```
                .PROC TEST2
CONSOLE_WRITE   .EQU 0FF03
COLS            .EQU 00
LINES           .EQU 01
                LDA #20.
                STA LINES
$0              LDA #40.
                STA COLS
$1              LDX #01         ;X = 1 => write
                LDA #"*"
                JSR CONSOLE_WRITE
                DEC COLS
                BNE $1
                LDX #01
                LDA #13.
                JSR CONSOLE_WRITE
                LDX #01
                LDA #10.
                JSR CONSOLE_WRITE
                DEC LINES
                BNE $0
                RTS
                .END
```

Listing 6:

```
                .PROC TEST3
CONSOLE_INIT    .EQU 0FF06
IREAK           .EQU 0BF1D
ISYSCOM         .EQU 0BF1F
                LDA IBREAK+1    ;Break-Adresse auf Stack
                PHA
                LDA IBREAK
                PHA
                LDA ISYSCOM+1   ;Syscom-Adresse auf Stack
                PHA
                LDA ISYSCOM
                PHA
                LDX #02         ;X = 2 => init
                JSR CONSOLE_INIT
                RTS
                .END
```

Listing 7:

```
                .PROC TEST4
CONSOLE_STATUS  .EQU 0FF2A
                LDA PUFFERADR+1 ;Puffer-Adresse auf Stack
                PHA
                LDA PUFFERADR
                PHA
                LDA #00         ;Kontrollwort auf Stack
                PHA
                PHA
                LDX #04         ;X = 4 => status
                JSR CONSOLE_STATUS
                RTS
PUFFER          .BLOCK 8        ;Ergebnis in Puffer
PUFFERADR       .WORD PUFFER
                .END
```

Listing 8:

```
                .PROC TEST5
PRINTER_WRITE   .EQU 0FF09
COLS            .EQU 00
LINES           .EQU 01
                LDA #20.
                STA LINES
$0              LDA #40.
                STA COLS
$1              LDX #01         ;X = 1 => write
                LDA #"*"
                JSR PRINTER_WRITE
                DEC COLS
                BNE $1
                LDX #01
                LDA #13.
                JSR PRINTER_WRITE
                LDX #01
                LDA #10.
                JSR PRINTER_WRITE
                DEC LINES
                BNE $0
                RTS
                .END
```

Listing 9:

```
                .PROC TEST6
DISK_READ       .EQU 0FF12
DISK_WRITE      .EQU 0FF0F
                LDA 0C050       ;Grafikmodus
                LDA 0C057
                LDA #0FF
                JSR FILL
                LDA #00         ;Modus
                PHA
                PHA
                PHA             ;Laufwerk 0 = Unit #4:
                PHA
                LDA #20         ;Puffer bei $2000
                PHA
                LDA #00
                PHA
                LDA #20         ;Laenge = $2000
                PHA
                LDA #00
                PHA
                LDA #01         ;Blocknummer = 264
                PHA             ;Achtung! Diskette
                LDA #08         ;muss an dieser Stelle
                PHA             ;frei sein
                JSR DISK_WRITE
                LDA #00
                JSR FILL
                LDA #00         ;Modus
                PHA
                PHA
                PHA             ;Laufwerk 0 = Unit #4:
                PHA
                LDA #20         ;Puffer bei $2000
                PHA
                LDA #00
                PHA
                LDA #20         ;Laenge = $2000
                PHA
                LDA #00
                PHA
                LDA #01         ;Blocknummer = 264
                PHA
                LDA #08
                PHA
                JSR DISK_READ
                LDA 0C051       ;Textmodus
                RTS
DEST            .EQU 00
FILL            LDY #00         ;fuellt Hires mit Akku-Inhalt
                STY DEST
                LDX #20
                STX DEST+1
$0              STA (DEST),Y
                INY
                BNE $0
                INC DEST+1
                DEX
                BNE $0
                RTS
                .END
```

Tabelle 1:
Einsprungadressen für BIOS-Aufrufe

```
FF00: CONSOLE READ
FF03: CONSOLE WRITE
FF06: CONSOLE INIT
FF09: PRINTER WRITE
FF0C: PRINTER INIT
FF0F: DISK WRITE
FF12: DISK READ
FF15: DISK INIT
FF18: REMOTE READ
FF1B: REMOTE WRITE
FF1E: REMOTE INIT
FF21: GRAPHIC WRITE
FF24: GRAPHIC INIT
FF27: PRINTER READ
FF2A: CONSOLE STATUS
FF2D: PRINTER STATUS
FF30: DISK STATUS
FF33: REMOTE STATUS
FF36: CONSOLE CHECKING
FF39: USER READ/WRITE/INIT/STATUS
FF3C: PSUBDRIVE
```

Kyan-Pascal

Grundkurs

von Ulrich Stiehl

1. Betriebssystem

Definition: Kyan-Pascal ist eine Standard-Pascal-Implementierung für den Apple II+/e/c, die aus dem Pascal-Quelltext zunächst einen Assembler-Zwischentext und daraus dann einen 6502-Objektcode erzeugt, der unter dem Betriebssystem ProDOS in Verbindung mit der Runtime-Library lauffähig ist.

Ab 1.2.86 wird die Version 2.0 ausgeliefert. Dieser Beitrag basiert noch auf der Version 1.2, doch werden hier nur Dinge behandelt, die auch für die neue Version gelten werden.
Der Grundkurs, dem im nächsten Peeker ein Aufbaukurs folgen wird, ist eine stark gestraffte, kyan-spezifische Version des „Pascal-Kompaktkurses" aus Heft 12/85, S. 33-48. Aus Platzgründen mußten Querverweise zu UCSD- und Turbo-Pascal sowie zu Applesoft entfallen. Wer bislang noch niemals in Pascal programmiert hat, aber bereits Applesoft kennt, sollte deshalb zusätzlich den „Kompaktkurs" zur Hand nehmen. Im übrigen kann man auf die einschlägigen Lehrbücher zurückgreifen, denn Kyan-Pascal ist eine getreuliche Standard-Pascal-Implementierung.

1.1. Konfiguration

Eine Kyan-Arbeitsdiskette sollte mindestens folgende Dateien enthalten:
PRODOS: Kyan-Pascal wird mit der jeweils neuesten ProDOS-Version ausgeliefert, z.Zt. Version 1.1.1.
HELLO.SYSTEM: Dies ist ein kleines Verbindungsprogramm (Hauptmenü), das nach dem

Booten von PRODOS geladen wird und den Wechsel zwischen ED, PC und anderen Systemprogrammen ermöglicht.
ED/E80: Kyan-Editor für 40-Zeichenbildschirm (ED) oder IIe/c- bzw. Videx-80-Zeichenkarte (E80).
PC: Pascal-Compiler.
LIB: Die Runtime-Library wird von dem Objektcode eines compilierten Kyan-Programms automatisch mitgeladen.

1 Wenn Sie einen Apple II+ mit einem 140K-Laufwerk besitzen und zusätzlich noch zwischen Kyan-Pascal und Applesoft wechseln wollen, empfehlen wir folgende Konfiguration Ihrer Kyan-Arbeitsdiskette:
PRODOS
BASIC.SYSTEM (für STARTUP)
HELLO.SYSTEM
STARTUP (für Applesoft)
ED oder E80
PC
LIB
Sie haben dann noch genügend Platz für in Arbeit befindliche Kyan-Programme.

2 Wenn Sie einen Apple IIc oder IIe mit 64K-Karte haben, so empfehlen wir folgende Konfiguration:
PRODOS
BASIC.SYSTEM (für STARTUP)
HELLO.SYSTEM
STARTUP (für PROFID)
PROFID
E80
PC
LIB
Mit unserem Kopierprogramm PROFID aus „ProDOS für Aufsteiger, Bd. 2" können Sie vollautomatisch die Dateien E80, PC und LIB auf die 64K-Karte kopieren, auf der dann noch ca. 20 Blocks für Kleinstprogramme frei sind. Größere Programme müssen Sie weiterhin auf der physischen Diskette compilieren. Alternativ können Sie auch PC auf der physischen Diskette belassen, so daß auf der 64K-Karte Großprogramme compilierbar sind.

3 Wenn Sie eine RAM-Karte mit mindestens 128K besitzen, haben Sie ausgesorgt. Dann können Sie alle Systemprogramme einschließlich der in Arbeit befindlichen Quelldateien mit PROFID auf die RAM-Karte kopieren. Der Wechseln zwischen E80 und PC sowie das Compilieren selbst sind sehr schnell. Wer zusätzlich Besitzer einer Accelerator- oder Speedemon-Karte ist, gelangt in einen wahren Geschwindigkeitsrausch.
Eine RAM-Karte ist für die Programmentwicklung effizienter als ein zweites Laufwerk. Selbstverständlich können Sie auch ein größeres Laufwerk oder eine Festplatte verwenden.

1.2. Programmwechsel

Bevor Sie mit Kyan-Pascal beginnen, sollten Sie sich etwas mit ProDOS im allgemeinen und mit dem Präfix-Befehl im besonderen vertraut machen (z.B. „ProDOS für Anfänger" aus Peeker, Hefte 2/84 und 4/85). Nehmen wir an, Ihre Arbeitsdiskette heiße „KYAN" und Ihre RAM-Disk „RAM". Dann können Sie vom BASIC.SYSTEM aus mit
–/KYAN/HELLO.SYSTEM
das Verbindungsprogramm starten (*mit* Strichbefehl!). Danach sehen Sie das Prompt-Zeichen und den Cursor rechts daneben, also
>□
Immer wenn dies der Fall ist, befinden Sie sich im sog. Hauptmenü, von dem aus Sie durch Eingabe eines Namens (*ohne* Strichbefehl!), z.B.
>/KYAN/PC
>/RAM/ED
>/KYAN/PROGRAMM.O
die Systemprogramme PC, ED, E80 oder eines Ihrer compilierten Pascal-Programme (mit Zusatz „.O") starten können. Wenn Sie gerade ED oder PC verlassen haben und nunmehr ein *Nicht*-Kyan-Programm, z.B. das BASIC.SYSTEM, starten wollen, so müssen Sie dies aus speichertechnischen Gründen *indirekt* über das HELLO.SYSTEM tun, also zuerst
>/KYAN/HELLO.SYSTEM
und direkt danach
>/KYAN/BASIC.SYSTEM
Die Kyan-Systemdiskette enthält ein zusätzliches Programm namens CD (= change directory), mit dem Sie ein neues Präfix festlegen können, z.B.
>/KYAN/CD
☎/RAM/
>PC
Danach können Sie so lange auf das Präfix „/RAM/" verzichten, bis Sie wieder auf die physische Diskette „/KYAN/" zugreifen wollen.

1.3. Editor

Der Kyan-Editor, der z.B. mit
>/KYAN/ED oder
>/RAM/E80 o.ä.
gestartet wird, ist ausgesprochen komfortabel und leicht zu erlernen. Der Editorspeicher umfaßt ca. 39.000 Zeichen. Nach dem Start über z.B.
>/KYAN/E80
sehen Sie das Editor-Menü und müssen nun zunächst den Namen des Quelltextes eingeben. Für ein neues Programm erfinden Sie einen Namen, z.B.
/KYAN/DEMO mit Präfix oder
DEMO ohne Präfix.
Danach gelangen Sie in den eigentlichen Editor. Für die ersten Experimente genügt es, wenn Sie sich auf einige wenige Befehle beschränken:

Hochpfeil: Cursor nach oben
Tiefpfeil: Cursor nach unten
Rechtspfeil: Cursor nach rechts
Linkspfeil: Cursor nach links
Del: Zeichen löschen von rechts
Ctrl-G: Zeichen löschen von links
Ctrl-T: Sprung zum Textanfang
Ctrl-V: Sprung zum Textende
ESC: zurück zum Editor-Menü

Sie befinden sich – wie beim Applewriter – permanent im Einfügemodus, d.h. alles was Sie tippen, wird an der momentanen Cursorposition eingefügt.
Der Kyan-Editor ist zwar ein Full-Screen-Editor, doch sollte eine Länge von 80 Zeichen/Zeile normalerweise nicht überschritten werden.
Mit ESC können Sie jederzeit in das Editor-Menü zurück. Von dort können Sie über
X (eXit with save)
Ihren Quelltext speichern und den Editor verlassen.
Der Editor hat noch zahlreiche andere Funktionen wie das Suchen und Ersetzen usw., worauf jedoch an dieser Stelle nicht eingegangen wird, weil wir mit einer Erweiterung der Befehle bei Version 2.0 rechnen.
Dem Listing KYANKURS kann man entnehmen, daß im Editor 6502-Quelltexte in den Pascal-Quelltext eingefügt werden können. Wie dies vor sich geht, wird im nächsten Peeker ausführlich beschrieben.

1.4. Compiler

Nehmen wir an, Sie hätten als Pascal-Neuling im Editor das Progamm

```
PROGRAM DEMO;
BEGIN
 Writeln ('The real danger is not');
 Writeln ('that computers will begin');
 Writeln ('to think like men,');
 Writeln ('but that men will begin');
 Writeln ('to think like computers.');
 Writeln ('Sydney J. Harris')
END.
```

eingegeben und unter dem Namen „/KYAN/DEMO" gespeichert. Danach starten Sie mit
>/KYAN/PC
den Pascal-Compiler, der Sie nun nach dem Namen des Quelltextes fragt, worauf Sie
>/KYAN/DEMO
eingeben. Die Compilierung erfolgt dann automatisch. Nun können Sie mit
>/KYAN/DEMO.O
Ihren compilierten Objektcode „DEMO.O" starten und testen. Compiliertes Programm *und* Daten können zusammen über 36.000 Bytes umfassen ($2000-$AFFF).
Auf die Darstellung der einzelnen Compiler-Optionen wird hier verzichtet, weil wir bei der Version 2.0 mit erweiterten Optionen rechnen.

2. Grunddefinitionen

2.1. Namen

Namen sind **Bezeichner** für Befehle, Variablen usw. Des näheren unterscheidet man die **reservierten Wörter**, z.B. AND, OR, END usw., von den **vordefinierten Bezeichnern** oder Standardbezeichnern, z.B. ABS, CHR usw., da letztere von der jeweiligen Pascal-Implementierung abhängen. Den Variablen usw. kann man eigene Namen geben, die dann **benutzerdefinierte Bezeichner** heißen.
Benutzerdefinierte Bezeichner müssen mit einem Groß- oder Kleinbuchstaben anfangen, dürfen danach jedoch auch Ziffern aufweisen. In Kyan-Pascal sind die ersten 8 Zeichen eines Bezeichners signifikant. Sonderzeichen ($, % usw.) und der Unterstreichungsstrich (_) sind nicht zulässig.
Kyan-Pascal kennt Konstanten-, Variablen-, Prozedur-, Funktions- und Programm-Bezeichner, von denen in diesem Grundkurs folgende behandelt werden:

```
PROGRAM, PROCEDURE

CONST, MAXINT
VAR

INTEGER, REAL, CHAR

EXP, LN
COS, SIN, ARCTAN
SQR, SQRT
ABS, ROUND, TRUNC

PRED, SUCC
CHR

READ, READLN
WRITE, WRITELN

IF-THEN, IF-THEN-ELSE
WHILE-DO, REPEAT-UNTIL
FOR-TO-DO, FOR-DOWNTO-DO

CASE-OF-END

DIV, MOD
NOT, AND, OR
```

2.2. Begrenzer

In Pascal gibt es **Sonderzeichen**, die entweder bestimmte Funktionen erfüllen (z.B. die mathematischen Operatoren +, -, ∗, /) oder ausschließlich der Begrenzung von Namen dienen (z.B. Leertaste und Return). Operatoren sind automatisch gleichzeitig Begrenzer. Einerseits läßt der Editor eine frei formatierte Eingabe des Quelltextes zu. Andererseits reagiert der Compiler etwas „unwirsch" auf fehlende Begrenzer. Ein Beispiel („WRITELN" = „write a line" ist ein Befehl, der eine Zeile oder hier ein Return ausgibt):
(1) WRITELN; WRITELN
(2) WRITELN;
WRITELN
(3) WRITELN;WRITELN
(4) WRI
TELN; WRI TELN
Befehle werden in Pascal durch „;" getrennt. Nach dem „;" kann man zusätzlich eine Leertaste (1) oder ein Return (2) einfügen oder auf beides verzichten (3). Man darf jedoch nicht *mitten in* ein Befehlswort ein Return oder eine Leertaste einfügen (4).
Begrenzer lassen sich einteilen in:
Einzeichen-Begrenzer: + - ∗ / = < > : . , ; ↑
Doppelzeichen-Begrenzer: := .. <= >= <>
Paarzeichen-Begrenzer: {-} (-) [-] '-'
Standard-Begrenzer: (Leertaste, Return)
Man merke sich:
1 Ein Name oder ein Doppelzeichen-Begrenzer darf niemals *in sich* durch einen Standard-Begrenzer unterbrochen werden, z.B.:
A := 1 ist richtig.
A :□= 1 ist falsch.
2 Ein Name muß stets links und rechts begrenzt werden, z.B.
□DIV□.

2.3. Programmaufbau

Ein Pascal-Programm besteht aus folgenden Teilen:
(1) Kopf (PROGRAM)
(2) Labels (LABEL)
(3) Konstanten (CONST)
(4) Typen (TYPE)
(5) Variablen (VAR)
(6) Prozeduren (PROCEDURE)
(7) Funktionen (FUNCTION)
(8) Hauptprogramm
Die Komponenten 1 und 8 sind obligatorisch, alle anderen fakultativ. Jede Komponente außer dem Hauptprogramm selbst beginnt mit einem reservierten Wort. In Kyan-Pascal muß die Reihenfolge strikt eingehalten werden; außerdem darf beispielsweise das reservierte Wort „VAR" nur ein einziges Mal vorkommen.

2.3.1. Kopf und Programm

```
PROGRAM DEMO1; {1}
BEGIN {8}
 WRITELN ('Anfang');
 WRITELN ('Ende')
END.
```

Der Kopf (1) besteht aus dem reservierten Wort „PROGRAM", gefolgt von einem benutzerdefinierten Programmnamen, gefolgt von einem „;", hier also
PROGRAM DEMO1;
Das eigentliche Programm oder genauer Hauptprogramm (8) beginnt mit dem Wort „BEGIN" und endet mit dem Wort „END", gefolgt von einem „.". Wenn der Compiler auf „END." mit Punkt stößt, hört er mit der Compilierung auf.
Zwischen „BEGIN" und „END." stehen die einzelnen Befehle, die *voneinander* durch „;" getrennt werden. *Vor* einem „END." ist deshalb das Semikolon entbehrlich.

Die Befehle eines Pascal-Programms werden im einfachsten Fall sequentiell abgearbeitet, also hier erst
WRITELN ('Anfang');
und dann
WRITELN ('Ende')
„BEGIN" und „END" haben eine Art Einklammerungsfunktion („Verbundanweisung"), worauf weiter unten eingegangen wird.
Hinweis: **Kommentare** werden eingeklammert, und zwar so {Kommentar} oder so (∗Kommentar∗).

2.3.2. Konstanten und Variablen

```
PROGRAM DEMO2; {1}
CONST {3}
 IK = 10;
 RK = 1.2345;
 CK = 'x';
VAR {5}
 IV: INTEGER;
 RV: REAL;
 CV: CHAR;
BEGIN {8}
 WRITELN (IK)
END.
```

Bei diesem erweiterten Demo werden zusätzlich Konstanten (3) und Variablen (5) definiert.

Die **Konstantendefinition** beginnt mit dem Wort „CONST", gefolgt von einer oder mehreren Gleichsetzungen, die jeweils durch „;" abgeschlossen werden (*auch* die letzte Gleichsetzung). Gleichsetzungen haben links vom „=" einen benutzerdefinierten Bezeichner und rechts vom „=" einen Wert. Im einzelnen gilt (Beispiele):
I = 10;
wird als Integer-Zahl erkannt.
R = 10.0;
wird als Real-Zahl erkannt.
C = 'A';
wird als Char = „Character" = Einzelzeichen erkannt.
Bei ganzen Zahlen, die Real-Zahlen sein sollen, muß man also einen Dezimalpunkt einfügen, damit der Compiler weiß, ob eine Integer- oder eine Real-Zahl gemeint ist.

Die **Variablendefinition** beginnt mit dem Wort „VAR", gefolgt von einer oder mehreren Typbestimmungen, die jeweils durch „;" abgeschlossen werden (*auch* die letzte!). Typbestimmungen haben links vom „:" einen benutzerdefinierten Bezeichner und rechts vom „:" eine Datentypbezeichnung. Es gibt u.a. folgende elementare Datentypen in Kyan-Pascal:
R: REAL;
Fließkomma-Zahl im Bereich +/-1E+/-99: 13stellige Mantisse mit 2stelligem Exponenten = intern 8 Bytes (BCD). Beispiel:
-1.234567890123E+99
I: INTEGER;
Ganzzahl mit Vorzeichen im Bereich +/-32767 = intern 2 Bytes.
C: CHAR;
Einzelnes ASCII-Zeichen, normalerweise mit Bit 7 off = intern 1 Byte.
Wenn mehr als eine Variable *desselben* Typs zu deklarieren ist, so kann man die Kurzform verwenden:
A, B, C, D, E: REAL;
Die Bezeichner werden hier durch Kommas getrennt.

In Pascal müssen grundsätzlich alle Konstanten und Variablen *vor* deren Benutzung „deklariert" werden, und zwar bereits vor dem ersten „BEGIN". Dies gilt auch für Laufvariablen von FOR-Schleifen.

Abschließend sei auf die Unterschiede zwischen „=", „:" und „:=" eingegangen:
- „=" wird u.a. erstens bei Konstantendeklarationen und zweitens bei Vergleichen benutzt, z.B.
CONST K = 10;
IF K = 10 THEN ...
- „:" wird u.a. bei Variablendeklarationen benutzt, z.B.
VAR R: REAL;
- „:=" wird bei Wertzuweisungen benutzt, z.B.
R := 123.456;

(Hinweis: Strings oder Zeichenketten werden in dieser Einführung ausgeklammert, da sie als ARRAY OF CHAR definiert werden müssen und die Verwendung von Include-Dateien für die Befehle CONCAT usw. erfordern. Eine vollständige String-Befehlsdatei werden wir im Aufbaukurs vorstellen.)

2.3.3. Prozeduren

```
PROGRAM DEMO3; {1}
PROCEDURE A1; {6}
BEGIN
 WRITELN ('Prozedur')
END;
BEGIN {8}
 A1
END.
```

In Pascal beginnen Unterprogramme mit dem reservierten Wort „PROCEDURE“, gefolgt von einem benutzerdefinierten Namen, gefolgt von „;“, also z.B.
PROCEDURE A1;
Das eigentliche Unterprogramm beginnt dann mit „BEGIN“ und endet mit „END;“ (im Geg. zum „END.“ beim Hauptprogramm). Der Aufruf des Unterprogramms bzw. der Prozedur erfolgt im Hauptprogramm durch einfache Angabe des Prozedurnamens. Näheres über Prozeduren und Funktionen sowie über Labels finden Sie im Kyan-Aufbaukurs.

3. Grundbefehle

Der KYANKURS enthält zu jeder Befehlsgruppe einige Übungsbeispiele, die Sie modifizieren sollten, damit Sie mit den Befehlen vertraut werden. Übertragen Sie zu diesem Zweck den KYANKURS von der DOS-3.3-Sammeldisk auf Ihre Kyan-Arbeitsdiskette mit dem ProDOS-CONVERT-Programm (oder mit unserem DOSTOPRO aus „ProDOS für Aufsteiger, Bd. 2“).

3.1. Bildschirm-Ausgabe

WRITE: Der WRITE-Befehl schickt die Werte bestimmter Datentypen (Integer, Real, Char und String) an das momentane Ausgabegerät, das hier stets der Bildschirm ist; andere Ausgabegeräte sind Drucker, Diskettendatei usw., die später behandelt werden. Auf das Befehlswort WRITE folgt in runden Klammern entweder *ein* Wert oder mehrere durch Kommas getrennte Werte. Beispiele:

```
{Je 1 Wert}
WRITE (1);
WRITE (123.456);
WRITE ('P');
WRITE ('Peeker');
{Je 2 Werte}
WRITE (1.1, 2.2);
WRITE ('AA', 'BB');
{Je 1 Konstante}
WRITE (IK);
WRITE (RK);
```

Man beachte die einfachen Anführungszeichen bei Stringwerten.

WRITELN: Im Unterschied zu WRITE schickt WRITELN in Kyan-Pascal ein Return nach dem *letzten* Wert, also quasi nach dem „)“, hinterher:
WRITELN (1);
ergibt 1 + Return.
WRITELN (1, 2);
ergibt 12 + Return.
Steuerzeichen können z.B. mit
R := CHR (13); {Return}
definiert werden, wobei R eine Char-Variable ist. Für ein nacktes Return genügt auch das einfache
WRITELN;

Home: Den Bildschirm löscht man in Kyan-Pascal mit
WRITELN (CHR (12));
wenn eine 80-Zeichenkarte verwendet wird. Der KYANKURS enthält eine kleine Assemblerroutine, damit der Home-Befehl auch in 40 Z/Z funktioniert.

Invers/Normal: Für inverse Bildschirmdarstellung gilt in Kyan-Pascal bei Verwendung der IIe/c-80-Zeichenkarte
WRITE (CHR (15)); {invers}
WRITE (CHR (14)); {normal}

GOTOXY: Der GOTOXY-Befehl fehlt in Kyan-Pascal. Der KYANKURS enthält deshalb eine kleine Assemblerroutine, die den Cursor mit GOTOXY (X,Y) positioniert, wobei X für die Spalte (0-79 bzw. 0-39) und Y für die Zeile (0-23) steht. Beispiele:
GOTOXY (0,0); {oben links}
GOTOXY (0,23); {unten links}

3.2. Formatierte Ausgabe

Der WRITE/WRITELN-Befehl läßt eine rechtsbündig formatierte Ausgabe (Print-Using) von Zahlen und Strings zu. Die allgemeine Form lautet
WRITELN (Ausdruck:G:N);
wobei G für die *Gesamtfeldlänge* und N für die *Nachkommastellen* (*nur* bei Fließkomma-Zahlen) steht. Beispiele (jedes „□“ steht für 1 Leertaste):
WRITELN (1:5);
gibt die Integer-Zahl 1 aus als
□□□□1
WRITELN ('A':3);
ergibt
□□A
WRITELN ('Peeker':10);
ergibt
□□□□Peeker.

Eine Kyan-Fließkomma-Zahl hat maximal 19 Stellen:
-1.234567890123E-12
Bei z.B.
R := 123.44444444;
WRITELN (R:10:2); wird
□□□□123.44 ausgegeben.

Sonderfälle:
1 Wenn eine Rundung in dem angegebenen G-N-Format nicht mehr möglich ist, werden x Nullen eingefügt, wobei x dem Exponenten entspräche. Beispiel:
R := 1E+80; WRITELN (R:10:2); ergibt
100000...00000.00,
d.h. eine Zahl mit 80 Nullen vor und 2 Nullen nach dem Dezimalpunkt, womit die Feldlänge weit überschritten wird.
2 Bei der unformatierten Ausgabe von Real-Zahlen steht in der *ersten* Stelle im Falle einer negativen Zahl ein „-“ und im Falle einer positiven Zahl eine Leertaste. Außerdem wird nur eine Stelle nach dem Dezimalpunkt ausgewiesen. Beispiel:
R := -1.2345;
WRITELN (R); ergibt -1.2E+00
und WRITELN (R:11); ergibt -1.2345E+00.

3.3. Wertzuweisung

Die Wertzuweisung zu einer Variablen V erfolgt in der Form
V := Ausdruck;
Links vom „:=“ steht eine Integer-, Real-, Char- oder String-Variable, und rechts vom „:=“ steht ein Literal („wörtlicher“ Ausdruck), eine Konstante, eine Variable oder ein Formelausdruck. Beispiele:

```
IV := 10;
RV := 10.5;
CV := 'A';
IV := IK + IK - 5;
```

In Kyan-Pascal gibt es u.a. die Systemkonstante **MAXINT** (= maximale Integer-Zahl 32767), die bei der Wertzuweisung verwendet wird, z.B.
IV := MAXINT;

3.4. Tastatur-Eingabe

READLN: Dieser Befehl liest eine Real-, Integer, Char- oder String-Variable über die Tastatur ein, wobei die Eingabe mit *Return* abgeschlossen werden muß, z.B.
READLN (R);
Bei READLN (R1, R2) kann nach R1 wahlweise ein Return *oder eine Leertaste* getippt werden, während nach R2 ein Return erforderlich ist. Da dies den Anwender unnötigt irritiert, vermeide man Mehrfach-READLNs.
Gibt man bei Integer- oder Real-Variablen „Quatsch“ ein, z.B. „AAA“, so wird der maximale negative Integer- oder Real-Wert zugewiesen, d.h.
-32768 für Integer
-9.0E-99 für Real.
Andere Pascal-Implementierungen „verabschieden“ sich in solchen Fällen mit einem Runtime-Error.

READ liest ein einzelnes Zeichen von der Diskette usw. ein. Für die Tastatur-Eingabe *ohne* Return verwende man besser die Assemblerroutine **RDKEY** (siehe KYANKURS).

3.5. Integer-Mathematik

+, - und ∗: Dies sind die einfachen Operatoren; hinzu kommt noch +/- als Vorzeichen, das ohne Leertaste direkt vor die Zahl gesetzt und eingeklammert werden muß. Beispiele:

```
IV := 10 + 20;
IV := 20 - 10;
IV := 10 * 10;
WRITELN ((-10) + (-20));
```

Merke: Ein Ausdruck darf nicht mit einem Vorzeichen beginnen.

Die Zuweisung (Fließkomma-Variable := Integer-Ausdruck) ist zulässig, z.B.
RV := 3 + 5;
Die umgekehrte Anweisung (Integer-Variable := Fließkomma-Ausdruck), z.B.
IV := 3.1 - 3.2;
ist verboten.
Man merke sich, daß Variablen beim Programmstart nicht automatisch auf 0 gesetzt werden. Das Mini-Programm

```
PROGRAM TEST;
VAR R: REAL;
BEGIN
 WRITELN (R)
END.
```

hat zur Folge, daß für R derjenige „Wert“ angezeigt wird, der sich zufällig im Speicher befindet.

DIV und MOD: Wenn gilt
Dividend : Divisor = Quotient + Rest
dann gilt
DIV = Quotient-Operator,

MOD = Rest-Operator.
Beispiele:
WRITELN (7 DIV 3); ergibt 2.
WRITELN (7 MOD 3); ergibt 1

ABS: Die Absolutfunktion eliminiert ein mögliches Vorzeichen, z.B.
WRITELN (ABS (-10)); ergibt 10.
SQR: Quadrat (nicht Quadratwurzel!) einer Zahl, z.B.
WRITELN (SQR (10)); ergibt 100.

PRED und SUCC: PRED = Predecessor = Vorgänger = Integer-Zahl - 1; SUCC = Successor = Nachfolger = Integer-Zahl + 1, z.B.
WRITELN (PRED (10)); ergibt 9.
WRITELN (SUCC (10)); ergibt 11.

ROUND und TRUNC: Hierbei handelt es sich um das Runden (ROUND) oder Abhacken (TRUNC von to truncate) einer Fließkomma-Zahl zu einer Integer-Zahl, z.B.
WRITELN (ROUND (10.8)); ergibt 11.
WRITELN (TRUNC (10.8)); ergibt 10.

3.6. Fließkomma-Mathematik

+, -, ∗, / sind die üblichen Operatoren für plus/minus/mal/geteilt, z.B.

```
RV :=   (1.1 +  2.2) * (3.3 - 2.2);
WRITELN (1.1 *  2.2  +  3.3);
WRITELN (1.1 * (2.2  +  3.3));
```

Man beachte die Prioritätsregeln: Punkt geht vor Strich, Klammer vor alles.

Potenz: Der „hoch"-Operator „X ↑ Y" fehlt in Kyan-Pascal und muß durch
EXP (Y ∗ LN (X));
simuliert werden.
WRITELN (EXP (3.0 ∗ LN (2.0)));
entspricht 2 ↑ 3.

SQR und SQRT: SQR steht für „Square" = Quadrat = X ∗ X, und SQRT steht für „Square Root" = Quadratwurzel von X, z.B.
WRITELN (SQR (3.0)); ergibt 9.0.
WRITELN (SQRT (9.0)); ergibt 3.0.

SIN, COS, ARCTAN: Dies sind die üblichen trigonometrischen Funktionen Sinus, Cosinus und Arcustangens. Die Berechnung erfolgt im Bogenmaß (= Pi/180; Gegensatz: Gradmaß = 360 Grad). Als Umrechnungsfaktor verwende man
F := 0.01745329251993;
Es gilt dann (Beispiele):
WRITELN (SIN (30 ∗ F):3:1);
ergibt 0.5.
WRITELN (COS (60 ∗ F):3:1);
ergibt ebenfalls 0.5.
Die **TAN**-Funktion fehlt in Kyan-Pascal und muß durch
RV := SIN (X) / COS (X);
simuliert werden.

LN, EXP: LN steht für die natürliche Logarithmus-Funktion und EXP für die Exponentialfunktion von e, z.B.
WRITELN (LN (100) / LN (10):3:1);
ergibt 2.0.
WRITELN (EXP (1):15:13);
ergibt 2.718281828556.

3.7. Verzweigungen und Schleifen

<, =, >, <=, >=, <>: Hierbei handelt es sich um die Vergleichsoperatoren (kleiner, gleich, größer, kleiner/gleich usw.). Man beachte, daß „><", „=>" illegal wären.

NOT, AND, OR: Dies sind die booleschen, logischen oder Wahrheitswert-Operatoren. Daneben gibt es noch den Datentyp BOOLEAN und die Konstanten TRUE und FALSE, worauf an dieser Stelle nicht eingegangen wird. Man beachte, daß die booleschen **vor** den Vergleichsoperatoren rangieren. Man benutze deshalb Klammern und programmiere
IF (1 = 1) AND (2 = 2) THEN...
und nicht
IF 1 = 1 AND 2 = 2 THEN ...
denn dies würde bedeuten
IF 1 = (1 AND 2) = 2 THEN...

IF-THEN-ELSE

Der sequentielle Befehlsfluß eines Pascal-Programms wird durch Verzweigungen, Schleifen und Unterprogramme durchbrochen. Beim IF-THEN-(ELSE-)Befehl muß man drei Befehlsteile unterscheiden:
– THEN-Befehle T1, T2...
– ELSE-Befehle E1, E2...
– Weitere Befehle W1, W2...

IF-THEN läßt sich umschreiben mit „Wenn Bedingung erfüllt ist, dann THEN-Befehl(e) T1, T2... ausführen, ansonsten direkt die weiteren Befehle W1, W2... ausführen." Je nachdem, ob von THEN ein einziger Befehl oder mehrere Befehle abhängen, unterscheidet man:
IF Bedingung THEN
T1;
W1;...
oder
IF Bedingung THEN
BEGIN
T1;
T2
END;
W1;...
Im letzteren Fall ist die BEGIN-END-Klammerung (= Verbundanweisung) erforderlich. Betrachten wir hierzu folgende Beispiele:

```
IF 1 = 2 THEN
 WRITELN ('T1');
 WRITELN ('T2');
WRITELN ('W1');
```

Hier werden T2 und W1 ausgegeben, was unerwünscht ist.

```
IF 1 = 2 THEN
BEGIN
 WRITELN ('T1');
 WRITELN ('T2')
END;
WRITELN ('W1');
```

Hier wird nur W1 ausgegeben, was erwünscht ist.

Merke: Wird die Bedingung erfüllt/nicht erfüllt, dann wird die THEN-Anweisung/THEN-Verbundanweisung ausgeführt/nicht ausgeführt. Nach einem „THEN" können theoretisch beliebig viele Befehle durch „BEGIN-END" verbunden werden. Vergißt man die BEGIN-END-Klammerung bei *mehreren* THEN-Befehlen, so fährt Pascal *in jedem Fall mit dem zweiten Befehl* nach dem THEN fort. Die Einklammerung wird vor allen Dingen dann kritisch, wenn verschachtelte IF-THEN-Konstruktionen verwendet werden.

IF-THEN-ELSE läßt sich umschreiben mit „Wenn Bedingung erfüllt ist, dann THEN-Befehl(e) T1, T2... ausführen, andernfalls ELSE-Befehl(e) E1, E2... ausführen, und danach in jedem Fall mit den weiteren Befehlen W1, W2... fortfahren". Man merke sich, daß vor den Wort „ELSE" das „;" verboten ist. Betrachten wir nun folgendes Beispiel:

```
IF A = B THEN
BEGIN
 WRITELN ('T1');
 WRITELN ('T2')
END {hier kein ";"!}
ELSE
BEGIN
 WRITELN ('E1');
 WRITELN ('E2')
END;
WRITELN ('W1');
```

Wenn A = B wahr ist, dann werden T1, T2 und W1 ausgegeben. Wenn A = B falsch ist, dann werden E1, E2 und W1 ausgegeben. Würde das Wort „ELSE" fehlen, dann würden in jedem Fall auch E1 und E2 ausgegeben. „ELSE" wird also benötigt, um *vor* den weiteren Befehlen W1, W2... anstelle der Wahr-THEN-Alternative T1, T2... die Falsch-ELSE-Alternative E1, E2... ausführen zu können.

CASE-OF-END

IF-THEN und IF-THEN-ELSE sind Entscheidungen, die von dem Wahrheitsgehalt *einer* Bedingung, d.h. einem einzigen logischen Vergleich, abhängen. Bei der CASE-Anweisung stehen *keine* logischen Bedingungen, sondern *feste Werte* in bezug auf eine vorgegebene Datentyp-Variable – *Selektor* genannt – zur Auswahl. Beispiel:

```
PROGRAM FAELLE;
VAR I: INTEGER;
BEGIN
WRITE ('ZAHL 1-3:');
READLN (I);
CASE I OF {CASE-Anfang}
 1: WRITELN (1);
 2: WRITELN (2);
 3: WRITELN (3)
 END {CASE-Ende}
END. {Programm-Ende}
```

FOR-TO/DOWNTO-DO

In Pascal lautet die Aufwärtszählschleife (als Beispiel)
FOR I := 1 TO 10 DO WRITELN;
und die Abwärtszählschleife
FOR I := 10 DOWNTO 1 DO WRITELN;
Wenn *mehrere* Befehle ausgeführt werden sollen, so ist die BEGIN-END-Verbundanweisung erforderlich. Beispiel:

```
FOR I := 1 TO 3 DO
BEGIN
 WRITELN ('A');
 WRITELN ('B')
END;
```

gibt dreimal „A" *und* dreimal „B" aus.

Spezialfälle:

FOR I := 1 TO 10 DO;
ist eine Leerschleife (nacktes „;“).
FOR I := 1 TO 3 DO
FOR J := 1 TO 5 DO
WRITELN (I, ' ', J);
ist eine verschachtelte Schleife. Stellen Sie fest, was ausgegeben wird!
FOR CV := 'A' TO 'Z' DO
WRITE (CV);
ist in Pascal möglich und gibt hier die Buchstaben von A bis Z in einer Zeile aus. Allgemein formuliert läßt die Pascal-FOR-Schleife als Laufvariablen sog. *skalare* Variablen zu. Dies sind u.a. Integer- und Char-Variablen, aber keine Real-Variablen. Deshalb ist die Schrittweite auch stets 1.
FOR I := 11 TO 10 DO
WRITELN (I);
bewirkt gar nichts, denn eine FOR-Schleife wird übersprungen, wenn der Anfangswert den Endwert übersteigt.

WHILE-DO

Diese Schleife heißt in Worten „Solange die WHILE-Bedingung noch erfüllt ist, führe die DO-BEGIN-END-Befehle aus“. Beispiel:

```
A := 1.0;
WHILE A < 10.5 DO
BEGIN
 WRITELN (A);
 A := A + 0.5
END;
```

Dieses Demo gibt die Zahlen 1.0, 1.5...10.0 aus.
Zwischen WHILE und DO steht wie zwischen IF und THEN ein logischer Ausdruck. Die WHILE-Schleife wird praktisch immer als BEGIN-END-Verbundanweisung konstruiert. Man beachte, daß die WHILE-Schleife übersprungen, d.h. *kein einziges Mal* durchlaufen wird, wenn die Bedingung bereits beim ersten Durchlauf nicht erfüllt ist. So wird z.B.
I := 2; WHILE I = 1 DO...
übersprungen.

REPEAT-UNTIL

Diese Schleife heißt in Worten „Führe die REPEAT-Befehle solange aus, bis die UNTIL-Bedingung nicht mehr erfüllt ist“. Beispiel:

```
A := 1.0;
REPEAT
 WRITELN (A);
 A := A + 0.5
UNTIL A >= 10.5;
```

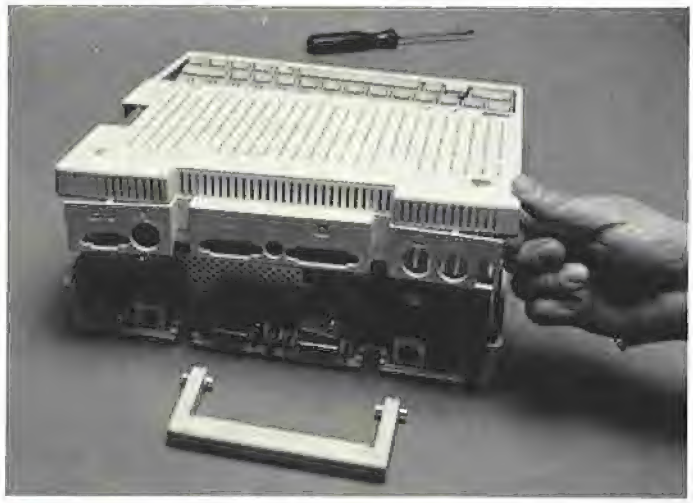
Dieses Demo gibt die Zahlen 1.0, 1.5...10.0 aus.
Der logische Ausdruck steht hier am Ende, d.h. nach UNTIL, so daß die REPEAT-Schleife im Gegensatz zu den FOR- und WHILE-Schleifen *mindestens einmal* durchlaufen wird. Wie das Demo zeigt, läßt sich eine WHILE-Schleife durch eine REPEAT-Schleife ausdrücken und umgekehrt. Im Gegensatz zur WHILE-Schleife ist bei der REPEAT-Schleife *niemals* eine BEGIN-END-Verbundanweisung erforderlich, weil REPEAT...UNTIL selbst eine Verbundanweisung darstellt. Wie *vor* dem END eines BEGIN-END-Verbundes muß auch *vor* dem UNTIL eines REPEAT-UNTIL-Verbundes kein „;“ stehen.

```
{-----------------------------------}
{1.0. Kopf/Konstanten/Typen/Variablen}
{-----------------------------------}
PROGRAM KYANKURS;

CONST IK = 100; RK = 100.5; CK = 'P'; SK = 'Peeker';
TYPE  STRING = ARRAY [1..6] OF CHAR;
      {TYPE und ARRAY sind im Grundkurs nicht erklaert!}
VAR   IV: INTEGER; RV: REAL; CV: CHAR; SV: STRING;
      IH: INTEGER; RH: REAL;
      R:  CHAR;    T:  CHAR;

{-----------------------------------}
{Hinweis:                           }
{Im Peeker 4/86 wird detailliert    }
{gezeigt, wie man unter Kyan-Pascal }
{Assemblerroutinen einbindet.       }
{-----------------------------------}
PROCEDURE HOME;
BEGIN
#A
    STX T      ;Stets X in Temp ($0010) retten!
    JSR $FC58  ;HOME
    LDX T      ;Nachher stets X wieder laden!
#
END;           {Ersetzt RTS}

PROCEDURE GOTOXY (X,Y: INTEGER);
BEGIN
#A
    STX T       ;X-Register speichern
    LDY #3
    LDA (SP),Y  ;YL: VTAB
    CMP #24
    BCS A1      ;groesser als 23!
    JSR $FB5B   ;TABV
    LDY #5
    LDA (SP),Y  ;XL: HTAB
    CMP #80
    BCS A1      ;groesser als 79!
    STA $057B   ;XL: HTAB 0-79 - 80 Z/Z IIe/c
    STA $0024   ;XL: HTAB 0-23 - 40 Z/Z II+/e/c
A1  LDX T       ;X-Register laden
#
END;           {Ersetzt RTS}

FUNCTION RDKEY: CHAR;
BEGIN
#A
    STX T       ;X-Register speichern
    JSR $FD0C   ;RDKEY: Tastaturabfrage
    AND #$7F    ;Bit 7 loeschen
    LDY #3
    STA (SP),Y  ;in Funktionswert poken
    LDX T       ;X-Register laden
#
END;           {Ersetzt RTS}

{-----------------------}
{2.1. Bildschirm-Ausgabe}
{-----------------------}
PROCEDURE BILDSCHIRMAUSGABE;
BEGIN
HOME; WRITELN; WRITELN (R);

WRITELN ('Hans', ' Meier');
WRITELN ('Hans', R, 'Meier', R);
WRITELN (CHR(15), 'Apostroph '' so', CHR(14), R); WRITELN;
WRITE ('A'); WRITE ('-'); WRITELN ('B'); WRITELN;
RV := 12345.6789; WRITE (RV, '/', RV, '/', RV);

GOTOXY (29, 19); WRITE ('Hier');
GOTOXY (29, 18); WRITE ('Darueber');
GOTOXY (0,0); WRITE ('Oben links');
GOTOXY (0, 19); WRITELN ('Zeile 20')
END;

{------------------}
{2.2. Formatierung}
{------------------}
PROCEDURE FORMATIERUNG;
BEGIN
WRITELN ('AUSDRUCK:G:N', R);
RV := 123.456789;
WRITELN (RV, '/', RV:12:3, '/', RV:12:4, '/', RV:12:5);
RV := -RV;
WRITELN (RV, '/', RV:12:3, '/', RV:12:4, '/', RV:12:5);
WRITELN;

WRITELN (R, 'AUSDRUCK:G', R);
WRITELN ('Zehn', ' ':10, 'Spaces'); WRITELN;

IV := -1;
WRITELN (IV, '/', IV:6, '/', IV:7, '/', IV:8); WRITELN;
```

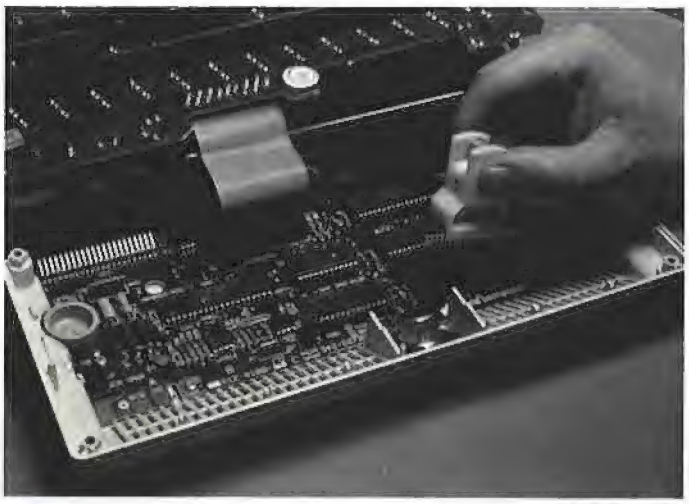
```
WRITELN (SK, '/', SK:10, '/', SK:15, '/', SK:20)
END;

{--------------------}
{2.3. Wertzuweisungen}
{--------------------}
PROCEDURE WERTZUWEISUNG;
BEGIN
WRITELN ('''Literals''', R);
RV:= 765.432;  IV:= 765; CV:= 'Z';
WRITELN (RV);  WRITELN (IV); WRITELN (CV);

WRITELN (R, 'Konstanten', R);
RV:= RK; IV:= IK; CV:= CK; SV := SK;
WRITELN (RV, '/', IV, '/', CV, '/', SV);

WRITELN (R, 'Systemkonstante MAXINT:', R);
IV:=  MAXINT; WRITELN (IV); IV:= -MAXINT; WRITELN (IV)
END;

{---------------------}
{2.4. Tastatur-Eingabe}
{---------------------}
PROCEDURE TASTATUREINGABE;
BEGIN
WRITELN ('READLN fuer Char, String, Real, Int!', R);
WRITE (R, 'Char: '); READLN (CV); WRITELN (CV);
WRITE (R, 'String: '); READLN (SV); WRITELN (SV);
WRITE (R, 'Real: '); READLN (RV); WRITELN (RV);
WRITE (R, 'Integer: '); READLN (IV); WRITELN (IV);
WRITELN;

WRITE ('RDKEY: '); CV := RDKEY; WRITELN (CV)
END;

{----------------------}
{2.5. Integer-Mathematik}
{----------------------}
PROCEDURE INTEGERMATHE;
BEGIN
WRITELN ('+, - und *', R);
IV:= 10 + 20; WRITELN ('10 + 20 = ', IV:6);
IV:= 20 - 10; WRITELN ('20 - 10 = ', IV:6);
IV:= 10 * 10; WRITELN ('10 * 10 = ', IV:6);

WRITELN (R, 'Quotient und Rest: '); WRITE (IK);
WRITE (' geteilt durch '); IH:= 7; WRITE (IH);
WRITE (' = '); IV:= IK DIV IH; WRITE (IV);
WRITE (' Rest '); IV:= IK MOD IH; WRITELN (IV);

WRITE (R, 'Absolutbetrag /-I/: ');
IH:= -7; IV:= ABS (IH); WRITELN (IH, ' -> ', IV);

WRITE (R, 'Quadrat: I * I: ');
IH:= 20; IV:= SQR (IH);
WRITE ('Quadrat von ', IH); WRITELN (' ist ', IV);

WRITE (R, 'Vorgaenger/Nachfolger: ');
IH:= 20; WRITE (IH); WRITE (': ');
WRITE ('Vorgaenger ', PRED (IH)); WRITE (', ');
WRITELN ('Nachfolger ', SUCC (IH));

WRITE (R, 'Runden: ');
RV:= -666.666; IV:= ROUND (RV); WRITE ('Real ', RV);
WRITE (' auf Integer gerundet ', IV);
WRITELN (' und abgehackt ', TRUNC (RV))
END;

{-------------------------}
{2.6. Fliesskomma-Mathematik}
{-------------------------}
PROCEDURE FLIESSKOMMAMATHE;
BEGIN
RV:= 123.12345678; RH:= 123.12345678;
WRITELN (RV:12:6, ' + ', RH:12:6, ' = ', RV + RH:12:6);
WRITELN (RV:12:6, ' - ', RH:12:6, ' = ', RV - RH:12:6);
WRITELN (RV:12:6, ' * ', RH:12:6, ' = ', RV * RH:12:6);
WRITELN (RV:12:6, ' / ', RH:12:6, ' = ', RV / RH:12:6);
WRITELN;

WRITE ('X ↑ Y = EXP (Y * LN (X)): ');
RV:= 3.0; RH:= 5.0;
WRITELN (RV, ' ↑ ', RH, ' = ', EXP (RH * LN (RV)):6:2);
WRITELN;

WRITELN (R, 'Punkt vor Strich, Klammer vor alles', R);
WRITELN  (' 3 + 4  *  5 - 13  = ', 3 + 4  *  5 - 13);
WRITELN  ('(3 + 4) * (5 - 13) = ', (3 + 4) * (5 - 13));

WRITELN (R, 'Diverse Funktionen', R);
WRITE ('Von Ausgangszahl '); RV:= 5.6789;
WRITELN (RV:7:4, ' -> '); WRITELN
(' ABS: ', ABS  (RV):8:4,
 ' SQR: ', SQR  (RV):8:4,
 ' SQRT:', SQRT (RV):8:4,
 ' SIN: ', SIN  (RV):8:4, R,
 ' COS: ', COS  (RV):8:4,
 ' ATAN:', ARCTAN (RV):8:4,
 ' Tan: ', SIN  (RV)/COS (RV):8:4,
 ' EXP: ', EXP  (RV):8:4, R,
 ' LN:  ', LN   (RV):8:4)
END;

{-------------------------------}
{2.7. Verzweigungen und Schleifen}
{-------------------------------}
PROCEDURE VERZWEIGUNGEN;
BEGIN
WRITELN ('IF-Vergleiche:', R);
IF (1 = 1) AND (1 <> 2) THEN
BEGIN
 WRITE ('1 = 1 ja');
 WRITE (', ')
END
ELSE WRITE ('So?');
   IF NOT ((2 < 1) OR (1 > 2)) THEN
   WRITE ('2 < 1 nein, ') {kein ';'!}
   ELSE WRITE ('Ach?');
      IF ('B' > 'A') THEN
      WRITELN ('"B" > "A"')
      ELSE WRITELN ('Wie?');

WRITELN (R, 'FOR nur mit Integer-Step 1',
 R, 'Fuer Real-Step WHILE + REPEAT', R);
FOR IH:= 1 TO 5 DO
 WRITE (IH, ' * ', IH, ' = ', IH * IH, ', ');
WRITELN;
FOR IH:= 5 DOWNTO 1 DO
 WRITE (IH, ' + ', IH, ' = ', IH + IH, ', ');
WRITELN;

WRITELN (R, 'Version mit WHILE prueft vorher', R);
RH:= 0.5;
WHILE RH < 5.5 DO
BEGIN
 WRITE (RH:5:2, ' '); RH:= RH + 0.5
END;
WRITELN;

WRITELN (R, 'Version mit REPEAT prueft nachher', R);
RH:= 0.5;
REPEAT
 WRITE (RH:5:2, ' '); RH:= RH + 0.5
UNTIL RH >= 5.5;
WRITELN
END;

{-------------------}
{3.0. Hauptprogramm}
{-------------------}
BEGIN
R := CHR(13); {Return}
REPEAT

HOME;
WRITELN ('***** KYAN-KURS1 *****', R, R,
'1. Bildschirm-Ausgabe', R,
'2. Formatierte Ausgabe', R,
'3. Wertzuweisung', R,
'4. Tastatur-Eingabe', R,
'5. Integer-Mathematik', R,
'6. Fliesskomma-Mathematik', R,
'7. Verzweigungen und Schleifen', R,
'8. Ende', R);

T := ' '; WHILE (T < '1') OR (T > '8') DO T := RDKEY;

HOME;
CASE T OF
'1': BILDSCHIRMAUSGABE;
'2': FORMATIERUNG;
'3': WERTZUWEISUNG;
'4': TASTATUREINGABE;
'5': INTEGERMATHE;
'6': FLIESSKOMMAMATHE;
'7': VERZWEIGUNGEN;
'8': WRITELN ('Ende...')
END;

IF T <> '8' THEN BEGIN WRITELN; CV := RDKEY END

UNTIL T = '8'
END.
```

Einbau der CP/M-3.0-Karte in den Apple IIc

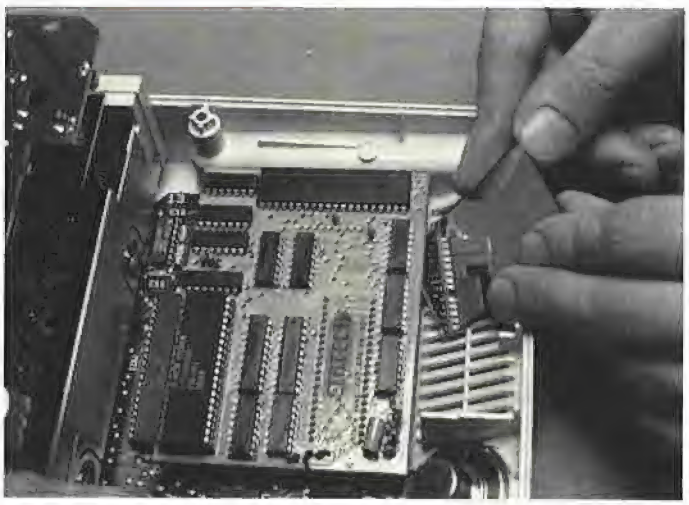
Apple-IIc- und Mac-Besitzer haben es besonders schwer, wenn sie ihre Maschine erweitern wollen, da der Einbau von Erweiterungen nicht ganz problemlos ist, zumal man die Apple-Garantie verliert. Wie man eine Erweiterungskarte einbaut, zeigen die nachfolgenden Bilder, die sich auf die CP/M-3.0-Karte der Firma Semjan in Frankfurt beziehen, die übrigens die Garantie im Falle eines Selbsteinbau übernimmt.



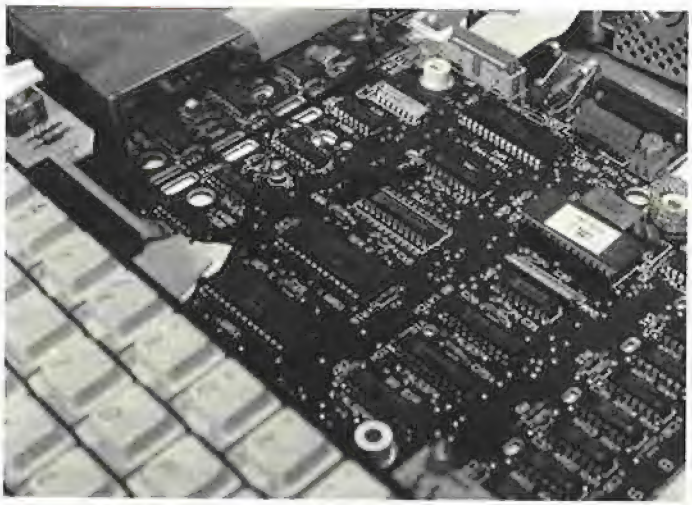
1. Nach dem Lösen der Bodenschrauben muß die Gehäuserückseite herausgeklappt werden, um das Oberteil abzunehmen.



2. Unter der herausgehobenen Tastatur wird der 65CO2-Prozessor sichtbar, der mit einer IC-Zange entnommen und in den freien Sockel der Zusatzplatine gesteckt wird.



3. Beim Einsetzen der Z80-Karte muß besonders darauf geachtet werden, daß die Pins exakt in der freigewordenen IC-Fassung liegen. Ein kleiner Spiegel kann hier helfen.



4. Zuletzt muß nur noch das zusätzliche Verbindungskabel angeschlossen werden. Hierzu ist das Laufwerk herauszuklappen.

Leserbriefe

Imagewriter
Ich finde Ihre Zeitschrift außerordentlich gut in jeder Beziehung. Mein Wunsch ist, etwas ausführlicher über den Imagewriter zu berichten.
Robert Brunner, CH-8400 Winterthur
(Anm.d.Red.: Siehe „Imagewriter kurz und bündig" in 12/85, S. 9 und „Superdump" in 6/85, S. 22. Ein weiterer Aufsatz über Imagewriter-Zeichensatz ist in Vorb.)

Premium-Softcard
Zum einjährigen Bestehen Ihrer Zeitschrift hiermit der wahrscheinlich 623. Glückwunsch. Meine Zufriedenheit mit Ihrem „Einjährigen„ bringe ich durch die beigefügte Abo-Karte zum Ausdruck. Hoffentlich finde ich im Heft Nr. 10 auch Antworten für meine Probleme: Ich benutze auf einem IIe-Kompatiblen eine Microsoft-Premium-Softcard. Tabellenkalkulationen und Textverarbeitung bereiten damit eine wahre Freude. Es tauchen allerdings Probleme – wahrscheinlich wegen des Exotencharakters dieser Karte – auf. Eine RAM-Disk (z.B. AP 17) wird nicht erkannt! An eine Erweiterung des Systems mit 2 x 80-Tracks Laufwerken wage ich jetzt nicht mehr zu denken. Wahrscheinlich ergeben sich dabei erhebliche Verständigungschwierigkeiten zwischen dem Disk-Controller (Erphi, Ehring oder IBS) und dem CP/M der Softcard. Nun, ich lasse mich überraschen. Vielleicht tragen Ihre Aktivitäten dazu bei, die Softcard zu popularisieren.
Dr. Mirus, Berlin
(Anm.d.Red.: Ein technischer Beitrag über Premium-Softcard liegt bereits vor und erscheint demnächst. Der Autor dieses Aufsatzes hat auch einen RAM-Disk-Driver entwickelt, der allerdings aus Platzgründen nicht veröffentlicht wird.)

Premium-Softcard
So, nun ist Ihr Peeker nicht nur ein Jahr alt, sondern seit einem Jahr bin ich sehr zufriedener Leser desselben. Seit kurzem aber habe ich ein Problem, und da es evtl. auch andere Peeker-Leser interessieren könnte und ich bisher weder von Microsoft in Deutschland Informationen noch von Erphi die Postanschrift erhielt, wende ich mich an Sie. Es handelt sich um die Premium Softcard IIe in Verbindung mit dem AFDC von Erphi. Wie komme ich wieder an die 80 Spuren, nachdem sich die Geschwindigkeit verdreifacht hat (alte CP/M-Karte)? Ja, es wäre überhaupt interessant zu erfahren, wie man Patches des 6502-BIOS auf der Floppy verankert und den Z80 bedient.
Franz Offenbächer, München

BASF-Disketten
Ihrem Erfahrungsbericht über die BASF-Flexidisk 1D kann ich nur zustimmen. BASF-Disketten werden auch von mir nicht mehr gekauft, da bei mir die Ausfallquote über 30% lag.
Günter Schulz, Dinslaken

Die gleichen Erfahrungen wie Sie habe ich auch mit BASF-Disketten 2/96 auf 160 Tracks Laufwerken gemacht. Die Disketten ließen sich erst nach mehrmaligen Versuchen formatieren. Auch hier wurden Reklamationen immer nur abgewimmelt. Also B = bloß, A = alles, S = scheinbar, F = Fehler?
Hartmut Klein, Bremen
(Anm.d.Red.: Die Peeker-Redaktion wurde inzwischen von zwei BASF-Ingenieuren besucht. Eine Fehlerquote von ca. 1% wurde bestätigt. Allerdings muß man unterscheiden zwischen harten und weichen Fehlern: Ein harter Fehler liegt vor, wenn die Kunststoffscheibe physisch defekt ist; in solchen Fällen kann niemals formatiert werden. Ein weicher Fehler liegt vor, wenn sich eine winzige Unreinheit auf der Kunststoffscheibe befindet; wenn man in solchen Fällen mit Spezialprogrammen bei den als scheinbar defekt ermittelten Spuren etliche Sekunden lang Formatierungsversuche unternimmt, ist die Diskette oft zu retten, womit sich die von uns statistisch ermittelte Fehlerquote entsprechend senkt. In der Redaktion sowie im Software-Service können wir jedoch aus Zeitgründen nicht jede Diskette einzeln untersuchen, bevor wir sie in Gebrauch nehmen und erachten deshalb auch weiche Fehler als echte Fehler, wenn bei *konventionellen* Formatierungsprogrammen ein Programmabbruch erfolgt. Übrigens haben wir 100 BASF-Qualimetric-Disketten zu Testzwecken erhalten und werden demnächst darüber berichten. us)

Operator I
Im „Erfahrungsbericht Operator I" (Peeker 9/85, S. 60) ist dem Autor ein Fehler unterlaufen: Das einfache Abführungszeichen (Hex 60) läßt sich doch mit der Tastatur erzeugen, und zwar mit CONTROL-SHIFT-7. Leider gibt auch der Tastenbelegungsplan in der Bedienungsanleitung eine falsche Auskunft, so daß man nur zufällig oder durch systematisches Durchtesten aller Tasten auf diese Möglichkeit stößt.
Dieter Steinwender, Hamburg

Apple-III-Kontaktadresse
Nachdem Herr Stiehl in den vergangenen Monaten mehrfach darauf hingewiesen wurde, daß sich Minderheiten betreffend weniger populärer Systeme, als dies die Apple-II-Familie darstellt, beim Verlag melden sollten, möchte ich als einer der wenigen Apple-III-Anwender diesen Aufruf wiederholen. Da der Vertrieb der Apple-III-Familie durch Apple Deutschland nun endgültig eingestellt worden ist, erscheint es mir um so wichtiger, daß Meinungen und Erfahrungen bezüglich Software- und insbesondere Hardwareentwicklungen nicht den so oft zitierten Weg in die „Mottenkiste" gehen, sondern den noch verbliebenen Anwendern publik gemacht werden sollten. Insofern bitte ich alle Apple-III-User nochmals an dieser Stelle: „Laßt den Apple III nicht untergehen. Es gibt sicherlich noch einige unter Ihnen, die mit dem System viel Spaß und Freude haben, sei es privat oder auch kommerziell. Nur durch Erfahrungsaustausch ist es möglich, den Apple III am Leben zu erhalten". Ich bin bereit, meine Adresse als Kontaktadresse zur Verfügung zu stellen und mir zugesandte Beiträge so gut wie möglich an andere weiterzugeben. Schreiben Sie an:
Jürgen Janson, Bismarckstr. 20, 6750 Kaiserslautern, Tel: 0631/65302

Operator IIe
Ich habe mich gefreut, als ich Ihren Bericht über die Tastatur Operator IIe von AFC Köln im Peeker gelesen habe. Nachträglich möchte ich noch zu Ihrem Bericht bemerken, daß der Einbau des Interfaces der Tastatur auf der Platine für handwerklich Begabte recht einfach ist. Nachteilig zu den Tastaturen ist noch zu sagen, daß es leider keine Overlays gibt und daß man auf den ersten Blick nicht erkennen kann, ob man sich im deutschen oder amerikanischen Zeichensatz befindet. Speziell zur Operator IIe wäre noch zu sagen, daß es bei einigen Programmen, die die Apfeltasten benötigen, zu Problemen kommen kann. Dies ist z.B. bei der Textverarbeitung von Appleworks der Fall, wenn man von der Tastatur aus (nicht über das Hilfsmenü Druckparameter) einen Text unterstreichen will (mit Ctrl-). Diese Funktion ist leider so ohne weiteres nicht ansprechbar. Als Lösung würde ich eine Funktionstastenbelegung mit dem Wert 1F hexadezimal vorschlagen. Weiterhin nachteilig, aber nicht störend, ist, daß man die Funktionstasten nicht mit Funktionen belegen kann, die die Apfeltasten benötigen (z.B. Appleworks, Quickfile). Alles in allem kann ich als Anwender jedoch sagen, daß die Operator IIe-Tastatur ein doch gutes Preis-Leistungsverhältnis darstellt und ich mit ihr sehr zufrieden bin.
Martin Bucholz, Dietzenbach

Dazzle Draw
Diesem Testbericht ist nachzutragen, daß Dazzle Draw bei einer Konfiguration von Apple IIe, Taxan-Monitor und 64K-80-Zeichenkarte von Taxan keine Farben, die man gute Farbdarstellung nennen kann, erzeugt. So gesehen betrachte ich die Behauptung der Betriebsanleitung: „...den Gebrauch von 16 Farben in der revolutionären Technik der doppelt hochauflösenden Grafik" als gigantische Übertreibung. Was man sieht, ist im „adjust color-feature" die komplementäre Darstellung von Farben und in den anderen Features sonst nur violette und grünliche Farbtöne. Diese Software ist somit nur auf monochromen Bildschirmen verwendbar. Mich würde nun interessieren, wo und wie der Tester die Farben gesehen hat. Oder ist womöglich jemand farbenblind gewesen?
Gunter Eisermann, Hüttenfeld

Apple-III-Besitzer
Als Apple III-Besitzer und „Nur-Pascal-Programmierer" bin ich natürlich schon daran interessiert, daß die Zeitschrift Peeker möglichst viele Pascal-Programme bringt, zumal meiner Ansicht nach die Umsetzung von Pascal nach Basic einfacher vonstatten geht als umgekehrt oder gar die Umsetzung von Applesoft-Basic nach MBasic. Nach meinen Erfahrungen mit Fortran IV, COBOL und dem Eurocom II-Basic bin ich vom Apple-III-Pascal-System begeistert, besonders hinsichtlich der Möglichkeit, häufig benötigte Unterprogramme aus einer Text- oder Unit-„Bibliothek" einzubinden und der Pflegbarkeit der Programme auch nach Monaten und Jahren. Die von mir erstellten Programme belegen im allgemeinen 40 bis 70 Blocks, die Texte dazu über 100 Blocks. Man stelle sich nur vor, ein solches Programm in Basic nach 12 Monaten ändern zu müssen! Allerdings läßt die Unterstützung seitens der Firma Apple mehr als zu wünschen

übrig. (Das von mir im März 1984 zurückgelieferte fehlerhafte SOS-Reference-Manual habe ich bis heute nicht erhalten!) Meine Bitte daher an Sie: ab und an mal ein paar Informationen über den Apple III im Peeker.
Dipl.-Ing. Reinhard Schäfer, Pohlheim
(Anm. d. Red.: Kontaktadresse s.o. Wir können auch gerne Kurzbeiträge zum Apple III veröffentlichen, die wir aber mangels Gerät nicht selbst prüfen geschweige denn schreiben können.)

Verbesserungsvorschläge
Um Ihre schon gute Zeitschrift weiter zu verbessern, möchte ich folgende Verbesserungsvorschläge anbieten:
1. Bei Hardwaretests sollten, soweit vorhanden, mehrere Produkte miteinander verglichen werden. In Heft 4/85 wurde z.B. nur der Erphi-Controller vorgestellt. Aus meiner Sicht wäre ein Vergleich zwischen Erphi- und z.B. dem Ehring-Controller besser gewesen. Gleiches gilt für die in Heft 9/85 besprochene Operator-Tastatur (Vergleich mit ACS, BROSE, PREH usw.).
(Anm. d. Red.: Wir können immer nur das testen, was uns für 1–2 Monate ausgeliehen wird.)
2. Da, wie auch im Peeker schon des öfteren angedeutet, der Apple II heute nicht mehr den Stand der Technik darstellt und es Gerüchte um einen 16-Bit-Nachfolger gibt, schlage ich einen Ideenwettbewerb mit Vorschlägen zu den technischen Daten für solch einen Nachfolger vor. Dabei könnten schon erste Erfahrungen mit dem Atari 520 ST und dem Commodore Amiga berücksichtigt werden. Diese Ideen könnte Apple vielleicht als Anregungen für den fälligen „großen Wurf" dienen. Sollte Apple die Anregungen nicht aufnehmen (man wird sehen, wohin das führt!), könnten auch andere Hersteller diese Ideen aufgreifen (z.B. Gepard oder IBS).
Stefan Niedergesäs, Berlin

Anm. d. Red.: IBS bestimmt nicht! Denn dort hat seit Herbst 1985 nur noch die Gläubiger-Bank das Sagen.)

CP/M-RAM-Disk
Zu der Zeitschrift Peeker, die ich regelmäßig mit großem Interesse lese, möchte ich Ihnen einige Anmerkungen und Wünsche mitteilen. Zunächst: das Konzept der Zeitschrift finde ich ausgezeichnet. Angesichts zahlreicher anderer, offenbar vom Gesetz des schnellen und leichten Geldes geprägter Computerzeitschriften, die mit großsprecherischen Ankündigungen von „Tips und Tricks" auf der Titelseite immer wieder enttäuschen, angesichts auch der räuberischen Preispolitik mancher Softwarevermarkter ist es äußerst erfreulich zu erleben, daß im Peeker viele komplexe, nützliche Programme nicht nur preisgünstig zur Verfügung gestellt, sondern auch vollständig veröffentlicht werden. Durch den Quelltext wird zum einen der Gebrauchswert dieser Programme erhöht (man kann entsprechend den eigenen Bedürfnissen patchen), zum anderen wird dem interessierten Computerbenutzer das beträchtliche Vergnügen serviert, das mit dem Kennenlernen guter Programme verbunden sein kann. Durch den Peeker habe ich allerlei erfahren, was ich immer schon mal hatte wissen wollen, z.B. Details von CP/M, die Funktionsweise von Diskettenlaufwerken u.a.m. Am Peeker schätze ich weiterhin die Kombination von detaillierter und fundierter Hintergrundinformation mit didaktisch hervorragend gemachten Einführungen, z.B. in ProDOS oder in die 6502-Assemblersprache. Bei den Artikeln für „Fortgeschrittene" könnte allerdings manchmal, wie mir scheint, mit wenig Aufwand die Verständlichkeit auch für nicht hochversierte Leser noch erhöht werden.

Beispiel: Die Beschreibung des CP/M-RAM-Disk-Drivers, Peeker 6/85, S. 60. Beim Studium des Programms staunte ich als ein nur wenig in Assemblerprogrammierung erfahrener Leser darüber, daß einige DB Statements mit 6502 Assemblersprache „kommentiert" werden; das hätte ich gern genauer verstanden. Der Begleittext erwähnt kursorisch das „Poken des Drivers", worunter ich mir nur etwas Nebulöses vorstellen kann. Vielleicht ließe sich so etwas in ein, zwei zusätzlichen Sätzen (möglichst ohne Anführungszeichen) klären. Es ist mir (daraufhin?) auch nicht gelungen, den Driver für ein 60K-CP/M (Version 2.23) anzupassen, obwohl ich auf Seite 58 desselben Artikels erfuhr, daß ich dazu mit Hilfe eben dieses Artikels in der Lage sein sollte. Ein Wunsch von mir an Peeker wäre der nach einer Hilfestellung (oder, falls erforderlich, nach einer Programmvariante) für eine solche Anpassung. Ansonsten möchte ich sagen, daß ich gerade diesen RAM-Disk-Driver dauernd mit großem Nutzen verwende und ihn wegen der hervorragenden Platzausnutzung auf der RAM-Disk für sehr gut gelungen halte.
Georg Pepping, Essen

(Anm. d. Red.: Der CP/M-Beitrag war wegen der kombinierten Z80- und 6502-Programmierung ausgesprochen schwierig. Ab 1986 bringen wir regelmäßig didaktisch aufbereitete Grundlagenaufsätze. Bei manchen Spezialaufsätzen müssen wir jedoch aus Platzgründen auf eine detaillierte Kommentierung verzichten, weil diese Aufsätze sonst gar nicht erscheinen könnten.)

GBASIC: Grafik und Text
Jörg Lange hat natürlich recht, wenn er in seinem Artikel darauf hinweist, daß es bei Verwendung einer 80-Zeichenkarte unter GBASIC „i.d.R." nicht möglich ist, Grafik und 4 Zeilen Text darzustellen, da CP/M grundsätzlich diese Zusatzkarte zur Ausgabe für Text benutzt. Abhilfe schaffen die folgenden BASIC-Programme, die es gestatten, wahlweise die 80-Zeichenkarte an- bzw. abzuschalten, indem sie das BIOS entsprechend poken.

```
10 POKE &HDAFD,1: REM Directory in 2 Spalten
15 POKE &HF3DD,0: REM Kleinschreibung möglich
20 POKE &HF3BB,0 REM keine 80-Zeichenkarte da
25 POKE &HDC42,&H44: REM Output auf Apple Screen,
30 POKE &HDC43,&HDC: REM JP $DC44 im BIOS
35 X=PEEK(&HE058): REM Softswitch 80-Zeichenkarte aus
40 PRINT „40 Z/Z"
```

```
10 POKE &HDAFD,3: REM Directory in 3 Spalten
15 POKE &HF3BB,4: REM 80-Zeichenkarte ist da
20 POKE &HDC42,&H04: REM Output auf 80-Zeichenkarte,
25 POKE &HDC43,&HDD: REM JP $DD04 im BIOS
30 X=PEEK(&HE059): REM Softswitch 80-Zeichenkarte an
40 PRINT „80 Z/Z"
```

Die folgenden Assembler-Programme erreichen dasselbe:

40 Zeichen	80 Zeichen
LD A,01	LD A,03
LD (DAFD),A	LD (DAFD),A
LD A,00	LD A,04
LD (F3DD),A	LD (F3BB),A
LD A,00	LD A,04
LD (F3BB),A	LD (DC42),A
LD A,44	LD A,DD
LD (DC42),A	LD (DC43),A
LD A,DC	LD (E059),A
LD (DC43),A	JP 0000
LD (E058),A	
JP 0000	

Es bleibt noch anzumerken, daß diese Programme unter CP/M 2.20 mit 56K TPA auf APPLE II Plus (oder Kompatible) laufen, nicht jedoch auf BASIS 108. Bliebe zum Schluß noch die Anregung, in einer der nächsten Ausgaben des Peekers einmal auf die Besonderheiten von CP/M 2.25 einzugehen.
Dr. Wolfgang Eichel, Braunschweig

Druckfehler im Programm „Designer"

In dem Listing des Pascal-Programms DESIGNER aus Peeker 1/86, ist auf S. 44, 2. Spalte der 4. Absatz (Procedure Input) wegen eines Montagefehlers nach unten gerutscht und muß oben vor „If Key" eingefügt werden.

Prometric-Motherboard

Erfahrungsbericht
von Dr. H. Vogel

Dieses neue Produkt eines Apple-kompatiblen Rechners aus deutscher Fertigung beinhaltet auf der hochintegrierten DIN-A3-großen Multilayer-Platine neben dem Apple-kompatiblen Rechner mit 64K RAM eine serielle und parallele Schnittstelle, eine 80-Zeichenkarte mit RGB/PAL-Modul, zwei schnelle CPUs (65C02 und Z80B) mit jeweils 64K RAM sowie eine aufsteckbare 256K-RAM-Pseudodisk (Version B3). Durch Jumper kann zwischen zwei Sprachen, die in drei 2764-EPROMs abgelegt sind, gewählt werden (z.B. Applesoft und AFORTH). Auf der Rückseite der Platine sind alle Anschlüsse herausgeführt; dadurch entsteht im Gehäuse kein Stecker- bzw. Kabel-„Wirrwar". Durch die Bauweise auf einer Platine entsteht auch kaum ein Wärmestau, der beim Apple und anderen Nachbauten zum Einsatz eines Ventilators zwingt.

Programme und Betriebssysteme

Direktes Booten von DOS, C-DOS, ProDOS, UCSD-Pascal 1.0, GFORTH und AFORTH neben Spielprogrammen wie Flugsimulator II, SARGON III und Utilities wie Key perfect und Locksmith 5.0 usw. war möglich. Mittels eines Patches auf der mitgelieferten Systemdiskette („SPEEDIX") liefen die Programme auch mit der 65C02-CPU anstandslos. Ebenfalls durch einen Patch mittels Systemdiskette waren CP/M 2.2, Wordstar, dBase II, MBASIC, GBASIC (letztere gepatcht), Turbo-Pascal und andere CP/M-Programme lauffähig. Zur Zeit ist der Betrieb von CP/M plus noch nicht möglich. Nur für den Apple IIe geschriebene Programme wie Quickfile sind mit der derzeit zur Verfügung stehenden Firmware nicht lauffähig. An einer Änderung der Firmware wird gearbeitet, um auch diese Programme booten zu können. Zur Nutzung der 80-Track-Laufwerke (in Verbindung mit einem Ehring-Controller) sowie der Pseudodisk (Version B3) ist eine Kopie der jeweiligen Betriebssystemdiskette an die Firma ESS oder deren Vertragshändler einzusenden; danach erhält man dann die gepatchte, auf Prometric-Rechner angepaßte Version.

65C02

Die durchgeführten Benchmarktests ergaben eine um den Faktor 3,5 kürzere Abarbeitungszeit von Programmen.

Z80B

Programme wie Turbo-Pascal und Benchmarktest waren bei der Programmabarbeitung um den Faktor 3 schneller als bei Verwendung der Microsoft-Z80-Karte. Da das Z80B-Modul IBS-kompatibel sein soll, wäre der 64K-Speicher als Pseudofloppy von der 6502-CPU aus nutzbar. Durch BIOS-Änderung des CP/M plus dürfte dies auch lauffähig sein; da hierbei jedoch „langsamere" 6502-Speicher benutzt werden müssen, würde der Geschwindigkeitsvorteil der Z80B-CPU teilweise wieder zunichte gemacht. Mit seiner zur AP22 (Z80B-Karte) kompatiblen Sandwich-Karte AP34 hat IBS eine Möglichkeit aufgezeigt, CP/M plus ohne Geschwindigkeitsverlust zu installieren – ein Weg, auch für das Prometric-Motherboard?!

80-Zeichen- und RGB-/PAL-Modul

Die 4 möglichen Zeichensätze werden in einer 7x9-Matrix dargestellt. Die Farbwiedergabe des PAL-Modus über einen farbtüchtigen HF-Modulator auf einem normalen Farbfernseher ist als gut zu bezeichnen. Besser waren die Farben über RGB auf einem RGB-Monitor. Mangels hochauflösendem RGB-Farbmonitor war es nicht möglich, das RGB-Modul auf seine 80-Z/Z-Darstellung und Farbtüchtigkeit hin zu testen. Falls eine solche Möglichkeit bestehen sollte, ergibt dies einen weiteren Pluspunkt für dieses Motherboard.

Serielle und parallele Schnittstelle

Keine Beanstandungen konnten beim Betrieb der seriellen Schnittstelle bis 19200 Baud festgestellt werden; in ihrer Funktionsweise entspricht sie der AP2 von IBS.
Die Parallel-Schnittstelle wird mit passender Firmware für den EPSON-Drucker geliefert; Grafik-Wiedergabe ist möglich.

Fazit

Zum ersten Mal wird ein Apple-kompatibles Motherboard vorgestellt, das neben der bisher gewohnten Integration eines Z80-Prozessors zwei schnelle CPUs beinhaltet, welche die Programmabarbeitung wesentlich beschleunigen. Gegenüber dem Einzelkauf eines Apple-IIe-kompatiblen Rechners mit gleicher Konfiguration liegt der Preis des Prometric-Motherboards um rund DM 1500,- niedriger. Dabei ist noch nicht der Zeitaufwand berücksichtigt, der beim Einzelkauf angesetzt werden muß, um alle Komponenten aufeinander abzustimmen, was bei Prometric wegfällt. Dieses Board könnte unter den Apple-kompatiblen Rechnern einen Standard setzen. Ob als Zusatzkarten für die verbliebenen Slots auch hochintegrierte, von ESS entwickelte Module angeboten werden (z.B. ein intelligenter Floppy-Controller mit Winchester- und Streamer-Anschluß sowie Hardware-Uhr) oder Sandwich-Karten (z.B. wie die IBS-AP34), bleibt abzuwarten. Der System-Urlader deutet jedoch schon in diese Richtung. Der Mangel an Kompatibilität zum Apple IIe kann wahrscheinlich durch Firmware-Änderung teilweise behoben werden. Durch Austausch des 6502-Prozessors gegen die 2MHz-Version des 65C02 waren bei der Assembler-Programmierung auch die neuen zusätzlichen Befehle verwendbar. Möglich dürfte demnach auch ein Austausch gegen die 1MHz-Version des 65C802 sein.

Bemängelt werden muß, daß das mitgelieferte Handbuch in einigen Punkten Lücken aufweist (z.B. über die Handhabung der mitgelieferten Programme auf der Systemdiskette zum Patchen).

Anmerkung zu den beim Test verwendeten Laufwerken:

Mit Erphi- bzw. CAT160-(Cumana-)Controllern in Zusammenhang mit BASF-6138-Laufwerken war kein Programm bootbar, jedoch mit den TEAC-55F-Drives. Das einzige mit Ehring-Controller in Verbindung mit BASF-6138-Laufwerken nicht bootbare Programm war Pro-DOS. Nach Durchsicht des Technical Manuals und Auskunft eines Technikers eines BASF-OEM-Händlers dürfte die Nicht-Lesbarkeit von Programmen auf den BASF-6138-Laufwerken an der zu kurzen Dauer des Lesesignals liegen (500 ns + 20%), welches vom Controller gar nicht oder falsch an der Computer weitergeleitet wird. Bei TEAC-Laufwerken soll die Lesesignaldauer mindestens 0,8 bis 1 betragen.

Microbuffer II

getestet von Dr. H. Kersten

Die Microbuffer-Karte von Practical Peripherals ersetzt die sonst übliche parallele Schnittstelle im Apple II/IIe. Sie besitzt
a) einen eingebauten Drucker-Spooler mit einer Speicherkapazität von 16K oder optional 32K (4164er RAMs, Nachrüsten auf 32K ist leicht möglich),
b) ROM-Software für verschiedene Textformatierungen,
c) druckerabhängige Grafik-Software in einem weiteren ROM, ist also als grafikfähiges Interface zu verwenden.
Die ROMs sind alle vom Typ 2716. Geliefert wird die Karte mit einer 15seitigen Bedienungsanleitung (in der vorliegenden Version in Englisch), in der die Installation, die Steuersequenzen für die Text- und Grafik-Optionen, sowie die Funktion der Karte unter den Betriebssystemen DOS, UCSD-Pascal, CP/M erläutert wird. Der Preis liegt den einschlägigen Inseraten zufolge bei DM 300,- bis 350,-. Insbesondere bei Neuanschaffung eines Druckers samt Interface fällt der Mehrpreis gegenüber einem Standard-Interface wenig ins Gewicht – zumal einiges geboten wird.
Getestet wurde die Karte auf einem Apple II+ mit Epson-Drucker FX80+ (Karte in Slot 1). Beeindruckende Testerfahrung ist zunächst die hohe Übernahmegeschwindigkeit: Ein ca. 30K langer Text wurde unter Applewriter in etwa 35 Sekunden in den Spooler-Puffer übertragen, d.h. auch große Datenmengen blockieren den Rechner nur für kurze Zeit. BASIC-Listings rollen beim „Druck" so schnell über den Bildschirm, als wäre kein Drucker angeschlossen. Nun zu einigen Details:

Die Stellung eines DIP-Switches auf der Karte bestimmt, ob beim Betätigen der Reset-Taste der Spooler-Puffer gelöscht oder unbeeindruckt weiter ausgedruckt werden soll. Mit einem weiteren Schalter läßt sich die Spooler-Funktion generell ein- oder ausschalten.

Für die Text- und Grafik-Funktion ist ein spezielles Steuerzeichen nötig: Ctrl-I unter DOS und Pro-DOS, Ctrl-Q unter UCSD und CP/M (kann abgeändert werden!). Weitere darauf folgende Zeichen wählen die gewünschte Option aus, z.B.:

– Start eines Selbsttestes mit Prüfmusterausgabe auf Drucker;
– strukturiertes Drucken eines BASIC-Listings (bei jedem „:" wird ein Zeilenvorschub ausgegeben und die neue Zeile eingerückt);
– Einstellen des linken Druckrandes, der Zeilenbreite und der Seitenlänge (mit automatischem Zeilen-/Seitenvorschub);
– Ausdrucken des augenblicklichen Inhaltes des 40-Zeichen-Schirms (oder Teilen davon);
– Löschen des Spooler-Puffers (stoppt unnötigen Papierverbrauch bei Fehlfunktionen/-bedienung von Programmen);
– Ausgabe der Hires-Pages 1 und 2, entweder vollständig oder mit üblichem Textfenster;
– Invertieren der Grafikpunkte (weiß/schwarz), zwei verschiedene Schwärzungsstufen;
– Drehen der Grafik um 90 Grad (vertikales Drucken), Vergrößern des Bildes.

Diese (und weitere) Optionen können auch von Programmen aus aufgerufen werden (BASIC- und Assemblerbeispiele sind in der Anleitung enthalten).

Nicht möglich ist es dagegen, den Inhalt des 80-Zeichen-Schirms auszugeben oder Lores zu drukken. Ein weiteres Problem ergibt sich bei solchen Programmen, die bestimmte Drucker-Optionen (z.B. Zeichensätze, Schrifttypen, ...) direkt über Ctrl-Codes ansprechen wollen. Hier kann es zu Überschneidungen mit den Steuersequenzen der Microbuffer-Karte kommen. Abhilfe schafft die sogenannte „transparente" Betriebsart. Wird diese durch ein Steuerzeichen aktiviert, verhält sich die Karte in der Folge wie eine „normale" Druckerschnittstelle. (Die eingebaute Intelligenz legt sich quasi schlafen.)

Ein weiterer Trick ist der sogenannte Maintain-Mode: Alle eingestellten Optionen/Parameter bleiben auch bei einem erneuten PR#1 oder sogar beim Booten eines anderen Betriebssystems erhalten. Dies ist insbesondere wichtig für einige Stand-alone-Betriebssysteme, bei denen man nach dem Booten meist keine Chance mehr zur Eingabe irgendwelcher Steuercodes hat.

Auch wenn die eine oder andere Option bereits zum Standard von Druckerschnittstellen gehört, so muß man hier doch bei der breiten Palette von Möglichkeiten sagen, daß die Microbuffer II ihr Geld wert ist. Man sollte beim Kauf allerdings darauf achten, ob der Mehrpreis für die 32K-Option gerechtfertigt ist: Es werden zusätzlich nur 2 RAMs 4164 plus Sockel benötigt.

Erphi FSS 280

(Unveränderte Firmenmitteilung)

Das Disksubsystem für Apple IIe mit 2 x 640 KByte Speicherkapazität

Wer kennt es nicht, das Problem mit der zu geringen Diskettenkapazität des Apple II? Es gibt kaum einen Anwender, der nicht auch schon einmal an die Grenzen der Speicherkapazität seiner 140KByte-Laufwerke gestoßen ist. Lange Zeit gab es außer Festplattenlaufwerken keine sinnvolle Alternative. Die hohen Anschaffungskosten und die ungelösten Datensicherungsprobleme haben viele Anwender vor dem Kauf einer Festplatte zurückschrecken lassen. Mittlerweile kann man am Markt die Entwicklung beobachten, daß vermehrt Laufwerke und Controller für höhere Speicherkapazitäten angeboten werden. Viele dieser Produkte jedoch sind nur mit gewissen Einschränkungen lauffähig, denn die Kompatibilität mit den für Apple erhältlichen Betriebssystemen ist nicht immer sichergestellt. Seit kurzem gibt es jetzt im einschlägigen Apple-Fachhandel das erphi FSS 280 – das Disksubsystem für Apple II mit 2 x 640 KByte Speicherkapazität. Das Gerät arbeitet mit allen wesentlichen Apple-II-Betriebssystemen, wie DOS 3.3, DiversiDOS, CP/M 2.2, Pro-DOS 1.1, Pascal 1.1 und Pascal 1.2. (weitere CP/M-Systeme sind z.Zt. in Vorbereitung).

Die Firma erphi electronic GmbH bietet hier auf Basis des ebenfalls von erphi entwickelten Autopatch-

Controllers ein Disksubsystem an, das durch Kompatibilität, Preis/Leistungsverhältnis, Zuverlässigkeit und Benutzerfreundlichkeit besticht. Der empfohlene Verkaufspreis für dieses Gerät beträgt DM 2.495,–.
Nun wird der aufmerksame, fachkundige Leser sagen: Das ist doch nichts Neues, das kann ich doch selber viel billiger bauen. Da kaufe ich mir den Autopatch-Controller und ein oder zwei 80-Track-Laufwerke und dann habe ich doch dasselbe.
Weit gefehlt! Alle Standard-80-Track-Laufwerke sind für das MFM-Aufzeichnungsverfahren optimiert und damit meistens nicht kompatibel zum Apple-Aufzeichnungsverfahren. Es müssen Hardware-Modifikationen durchgeführt werden, die von Produkt zu Produkt verschieden sind. Hier muß ein Kondensator, dort ein Widerstand eingelötet werden, in anderen Fällen müssen Leiterbahnen unterbrochen werden und Widerstandsnetzwerke ausgetauscht werden usw.

Zwei wesentliche Fragen, die sich hier aufwerfen:

1.) Sind die durchgeführten Änderungen wirklich optimal, d.h. hat das Laufwerk danach einen unter dem Apple-Aufzeichnungsverfahren optimalen Störabstand? Hierzu ist zu bemerken, daß sich in einem Laufwerk umfangreiche Analogschaltungen befinden und zwischen Fehlfunktion und Optimum eine erhebliche Bandbreite liegt.

2.) Was ist dann mit der Herstellergarantie? Die existiert nicht mehr, denn alle Hersteller lehnen es für gewöhlich ab, bei eigenmächtigen Hardware-Eingriffen Garantieleistungen zu gewähren.

Nachdem sich unser Anwender Laufwerke und Controller gekauft hat, wird er feststellen, daß er noch Gehäuse für die Laufwerke benötigt und, ach ja, Spannungsversorgungs- und Datenkabel braucht er auch noch. Die Spannungsversorgung ist kein Problem, da es ja vorgefertigte Kabel gibt, und auf dem Autopatch-Controller sind die Anschlußpunkte dafür vorgesehen. Das Datenkabel ist auch schon fertig, also nur noch zusammenstecken und fertig.
Wenn es doch nur so einfach wäre! Nach mehrstündigem Betrieb, auch im Standby, können in schlecht belüfteten Gehäusen Temperaturen gemessen werden, die bis zu 30 Grad C über der Umgebungstemperatur liegen. Wenn man nun noch weiß, daß einerseits Diskettenmaterial und Laufwerksmechanik unterschiedliche Ausdehnungskoeffizienten haben, bei einem 80-Spur-Laufwerk aber andererseits schon hundertstel mm Abweichung eine entscheidende Rolle spielen können, kann man sich vorstellen, was hier bezüglich der Datensicherheit riskiert wird. Laufwerkshersteller spezifizieren die obere Betriebstemperatur ihrer Geräte mit max. 45 Grad C.
Vorsicht ist auch bei der Spannungsversorgung aus dem Rechnernetzteil geboten, wie das folgende Beispiel zeigt. Standardlaufwerke haben eine gemeinsame Motor-On-Leistung, beim Diskettenzugriff laufen beide Motoren gleichzeitig an. Sind Disketten eingelegt, können hierbei auf der 12-V-Leitung Ströme von deutlich über 1 A fließen. Vielleicht ist der Apple auch voll bestückt mit diversen Zusatzkarten, dann spielt das Netzteil nicht mehr mit und versagt wegen Überlastung seinen Dienst. Na, denkt der findige Bastler, dann nehme ich eben ein größeres Gehäuse, ein Netzteil, baue noch einen Lüfter ein und dann muß es doch funktionieren. Da wird dann ein einfaches Kunststoff- oder Blechgehäuse gekauft, eines der billigen Schaltnetzteile und ein einfacher Lüfter, alles zusammengebaut und fertig. Nun abgesehen davon, daß der Lüfter sich äußerst geräuschvoll Gehör verschafft und daß es durch die Störstrahlung des Netzteils zu Schreib/Lesefehlern kommen kann, funktioniert das Gerät auch. Aber mit welchen Mühen und finanziellem Aufwand!
Da Probleme dieser Art von vielen Anwendern beklagt werden, hat erphi electronic das FSS 280 entwickelt. Zum Lieferumfang des FSS 280 gehören der Laufwerksteil, der Autopatch-Controller, das Handbuch, die Utility-Diskette und die notwendigen Kabel. Die Eigenschaften des Autopatch-Controllers sind bereits des öfteren in verschiedenen Artikeln beschrieben worden, so daß hier nur auf den Laufwerksteil des FSS 280 eingegangen wird.
In einem formschönen, Apple-farbenen Gehäuse sind zwei Diskettenlaufwerke, ein längsgeregeltes Netzteil mit eingebautem Netzfilter und ein flüsterleiser Lüfter eingebaut. In der Abstimmung der einzelnen Systemkomponenten aufeinander liegt das Geheimnis der Zuverlässigkeit und Qualität des FSS 280. Die beiden Laufwerke vom Typ Philips X 3134 A sind Doppelkopflaufwerke mit 2 x 80 Spuren. Die Motorregelplatine und das Analogboard werden bei erphi so verändert, daß eine möglichst hohe Laufkultur und eine absolute Kompatibilität zum Apple-Aufzeichnungsverfahren erreicht wird.

Anmerkung: Die im freien Handel erhältlichen X 3134 sind nicht direkt mit den erphi-modifizierten Laufwerken vergleichbar. Der Unterschied ist schon bei der Geräuschentwicklung direkt wahrnehmbar.
Das eingebaute Netzteil ist auf die spezifischen Eigenschaften der Laufwerke abgestimmt. Es hat keinen störenden Einfluß auf das Schreib/Lesesignal. Der eingebaute Netzfilter und die interne Verkabelung entsprechen den VDE-Vorschriften. Der eingebaute Lüfter arbeitet absolut geräuschlos und sorgt dafür, daß selbst nach 24 Stunden Dauerbetrieb die Temperatur im Laufwerksgehäuse nicht über 40 Grad C ansteigt. Die einzelnen Baugruppen des Systems, also Laufwerke, Netzteil, Lüfter, Controller und Verkabelung werden einzeln vorgetestet. Nach dem Zusammenbau der einzelnen Komponenten wird jedes Disksubsystem einem aufwendigen Endtest unterzogen. Diese Sorgfalt ist der wesentliche Grund für den erzielten Qualitätsstand des FSS 280. Deshalb gewährt erphi auch 12 Monate Garantie.

Placierung von Apple-Computern gemäß "Bestseller-Liste" (Skala 1–10) der Zeitschrift "Chip" mit Erscheinen von Apple IIc und Macintosh

IIe	1 2 3 3 2 2 2 2 2 1 2	4 3 3 5 5 5 5 3 3 2	2,9
IIc	– – – – 8 8 10 4 5 5 4	5 4 4 6 6 10 7 7 10 –	6,7
Mac	– – 10 10 3 6 7 6 6 6 6	7 6 5 9 9 6 6 6 9 9	6,9
Mon.	F M A M J J A S O N D	J F M A M J J A S O	5,5
	Februar–Dezember 1984	Januar–Oktober 1985	Mittel

Der Apple III war einmal (März 1984: Platz 8) und die Lisa ebenfalls einmal (März 1984: Platz 10) genannt.
In den Monaten Februar bis Juli 1984 wurden IIe und II+ zusammengefaßt.
In den Monaten Mai 1985 (Platz 10) und Juli (Platz 8) wurden auch Apple–II–Kompatible aufgeführt.
Quelle: Zeitschrift „Chip" ab 2/1984. Dieses Februarheft enthält auf S. 72–73 Hinweise zur Panel–Erhebung.

Inserentenverzeichnis Peeker 2/86

Der nächste Peeker Heft 3/1986 erscheint am 17. 2. 1986

Aus postalischen Gründen dürfen wir bei Produktberichten keine Firmenanschriften mitteilen.

Peeker-Redaktion

peeker

Vorschau Heft 3/86

(Änderungen vorbehalten)

Grundlagen

Binäres Rechnen mit Papier und Bleistift

Teil 2: Multiplikation und Division

Applesoft

Behandlung reeller Zahlen in Applesoft

Technik

Meßwertverarbeitung

Assembler

Windowing

Fenstertechnik beim Apple II

ProDOS

MAKESUB

Erstellung zusammenhängender Subdirectories

UCSD

Superschnelle Bildschirmausgabe

in Apple-Pascal

Kyan

Kyan-Pascal-Aufbaukurs

Datentypen, Unterprogramme, Strings, Dateien

CP/M

Wordstar, FX-80 und DDT

Premium-Softcard

Speicherverwaltung unter CP/M

Hobby

Kreiszahl Pi

auf 15 000 Stellen

Testberichte

Macintosh Plus

(Apple)

Okidata-Drucker ML-192

(Okidata)

CP/M-3.0-Karte für Apple IIc

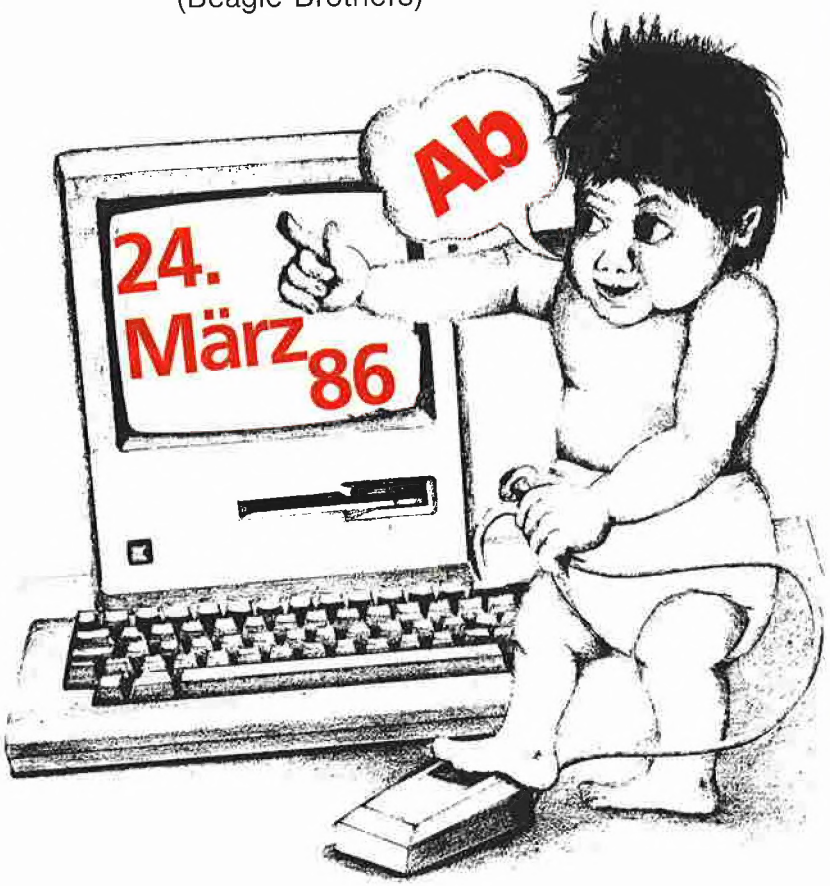
(Semjan)

Mockingboard

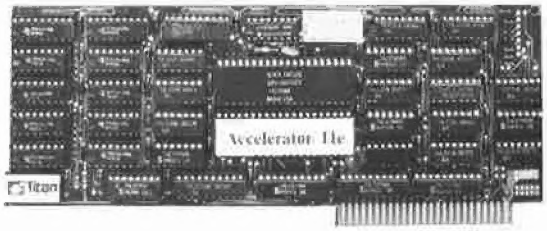
Musik- und Sprachkarte
(Sweet Micro Systems)

DCODE

Applesoft-Hilfsprogramme
(Beagle Brothers)



Accelerator™ IIe macht Ihren Apple® II, II Plus oder IIe dreieinhalbmal schneller.



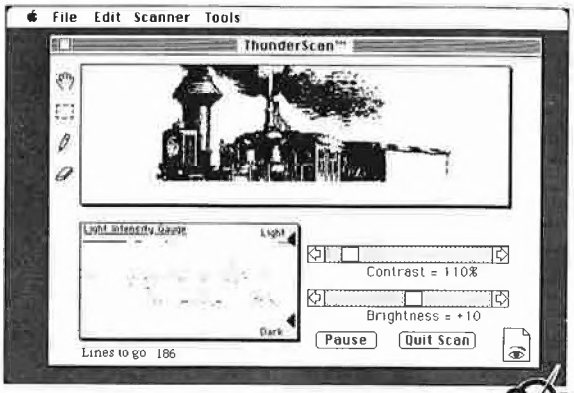
Jetzt laufen VisiCalc®, Apple Writer, PASCAL, BASIC, Datenbanken usw. endlich ohne langen Zeitverlust. Stecken Sie einfach die ACCELERATOR IIe Karte in irgendeinen Slot und beobachten Sie, wir Ihr Apple loslegt!
ACCELERATOR IIe besitzt seinen eigenen schnellen 6502 Prozessor und 80 K-Byte Hochgeschwindigkeitsspeicher, einschließlich einer eingebauten schnellen Sprachkarte und schnellem RAM-Speicherplatz für die ROM-Sprache.
Direkt von Pandasoft (Titan Distributor für Deutschland) oder bei Ihrem Applehändler.

pandasoft Dr.-Ing. Eden

Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

ThunderScan.™

Ein neues optisches Lesegerät, das beliebige Vorlagen in MacPaint überträgt: Fotos, Zeichnungen, Landkarten und Illustrationen werden in den Apple-Imagewriter eingespannt und von einem Lesekopf, der das Farbband ersetzt, abgetastet.



- 32 Graustufen
- 80 Punkte/cm Auflösung
- Übertragungsmaßstab 25 % – 400 %
- Vorlagen bis 20 x 25 cm
- Nachträgliche Veränderung des Kontrasts und der Helligkeit.

ThunderScan

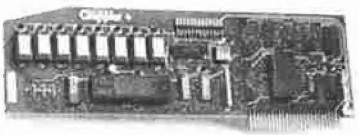
pandasoft Dr.-Ing. Eden

Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

Orange™ Printer Interface **Druckerinterfaces** für Apple II+/e/c/IIi Interfaces auf dem **neuesten Stand der Technik. Kompatibel** mit allen gängigen Druckern wie: APPLE, EPSON, STAR, NEC, OKIDATA usw. Passende Treibersoftware wird über Dip-Switch ausgewählt.

Grappler + Printer Interface **Grafikfähiges Druckerinterface** das keine Wünsche mehr offen läßt. Über **2 Dutzend Kommandos** ermöglichen die volle Kontrolle über alle Möglichkeiten Ihres Druckers. Jetzt auch mit **IIe Features: Double Hires Grafics** und **80 Zeichen Dump** mittels Druckerpuffer nachrüstbar über Bufferboard.

Buffered Grappler + Besitzt alle Vorzüge des Grappler +, hat aber zusätzlich einen integrierten **16 K Druckpuffer,** der auf **32 oder 64 K aufrüstbar** ist.



Serial Grappler Printer Interface **Serielles Druckerinterface** speziell für den **Apple Imagewriter.**

HOTLINK Seriell-nach-Parallel-Wandler für den IIc im Kabel integriert.

GRAPPLER C wie Hotlink, jedoch zusätzlich Imagewriter Emulation und Grafik Software-Diskette.

pandasoft Dr.-Ing. Eden

Uhlandstraße 195 · 1000 Berlin 12 · Mo–Fr 10–18 Uhr, Sa 10–13 Uhr
Telefon: 0 30/31 04 23 · Telex: 1 85 859

Sie haben einen Apple...

wir haben die Software...

und die Hardware...

wir haben die Bücher...

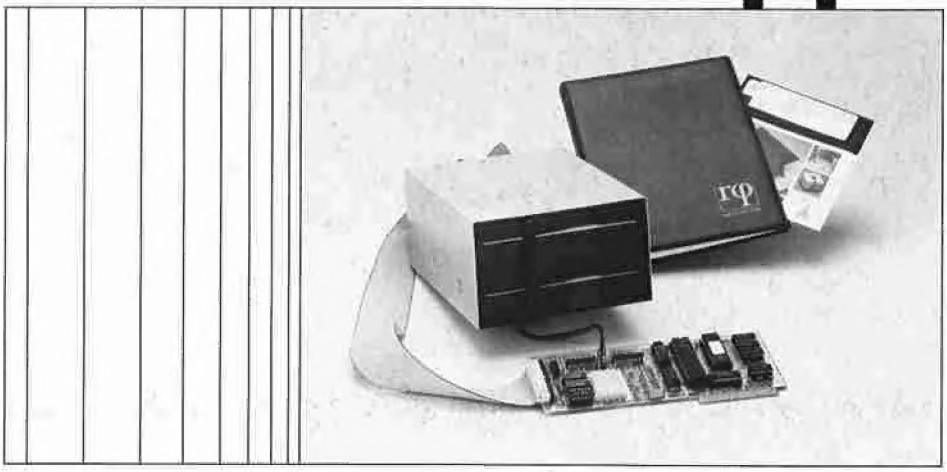
und die Zeitschriften...

***Fordern Sie unseren Gratiskatalog an!**
ALLES FÜR DEN APPLE II+, IIe, IIc UND MACINTOSH

pandasoft Dr.-Ing. Eden
UHLANDSTR. 195 · D-1000 BERLIN 12
TEL.:(030) 310 423 · TELEX: 18 58 59
Autorisierter Apple Fachhändler MICROSOFT Distributor

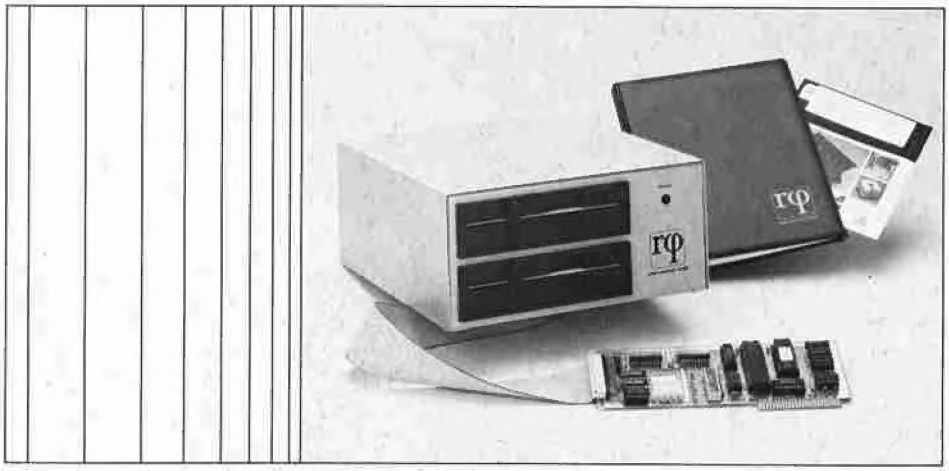
Ich besitze einen Apple. Bitte schicken Sie mir Ihren kostenlosen Katalog
Name .....
Adresse: .....

Anschlußfertig für Apple® II, //e...



erphi
Doppellaufwerke DL 280
2 Laufwerke mit je 640 kB formatiert im Gehäuse incl.
erphi
Autopatch-Controller
Handbuch und Diskette mit Dienstprogrammen

Verkaufspreis incl. MwSt.
DM 1.298,–



erphi
Floppy-Subsystem FSS 280
geeignet für kommerzielle Anwendungen
2 Laufwerke mit je 640 kB formatiert im Gehäuse mit eigener Stromversorgung incl.
erphi
Autopatch-Controller
Handbuch und Diskette mit Dienstprogrammen
Verkaufspreis incl. MwSt.
DM 1.698,–

Noch universeller und noch komfortabler...

Betriebssysteme
DOS 3.3
DiversiDOS 2-c, 4-c,
ProDOS 1.0.1, 1.1.0, 1.1.1
Pascal 1.1, 1.2
CP/M 2.20, 2.23, 2.26

SLOT-Unabhängig
bei DOS, DiversiDOS, ProDOS

Die bisherigen Vorzüge bleiben erhalten, wie

Autopatch-Boot
automatische Erkennung und Erweiterung der Betriebssysteme und Laufwerkskapazitäten während des Bootvorgangs (Betriebssystem auf der Boot-Diskette bleibt unverändert)

Originalsystem-Boot
von herkömmlichen Apple®-Disketten unabhängig vom Laufwerksformat weiterhin möglich

Problemloses Übertragen
herkömmlicher Apple®-Software auf Disketten höherer Kapazität durch SIM 35-Hilfsprogramm (simuliert 35-Spur-Laufwerk unabhängig vom tatsächlichen Laufwerksformat)

rφ
erphi electronic
GmbH

erphi
electronic
GmbH
Dammweg 3
D-8011 Großhelfendorf
Telefon (0 80 95) 441
Telex 528 021 erphi d

ikw