Nr. 2/87 Februar
DM 6.50, sfr. 6.50, öS 50, Lit 5900, hfl. 7.50

peeker

MAGAZIN FÜR MIKROCOMPUTER

Neuer 6502-Assembler
24-Nadel-Grafikausdruck
Pattern-Matching
Long-Integer-Routinen
Micol-BASIC

65816-Kompaktkurs

Hüthig
PUBLIKATION

Aktuelle Computerbücher

1985, 305 S., 6 Abb., kart.,
DM 58, – , ISBN 3-7785-1150-5
Begleitdiskette DM 48, –
doppelseitig beschrieben
ISBN 3-7785-1290-9

Dieses Buch wendet sich als lehrbuchhafter Kurs an alle, die professionelle hochaufgelöste Grafiken auf dem Apple erzeugen wollen. Der erste Teil beginnt mit einem Abriß des Aufbaus der HGR-Seiten aus der Sicht des Programmierers. Danach wird das Programm HRCG (HI-RES Character Generator, Apple, Inc.) eingehend analysiert, und es werden sinnvolle Ergänzungen vorgestellt. Schrittweise wird die Nutzung des HRCG erarbeitet bis hin zur beliebigen Bewegung eines statistischen Objekts auf einer der HGR-Seiten.
Der zweite Teil baut auf dem ersten auf und führt über die Definition mehrerer Objekte und simultaner Bewegung hin zu einem Arcade-Spiel, das für die meisten kauflichen Action-Spiele in der meisterhaften Grafik als Vorbild dienen kann. Grundkenntnisse in 6502-Assembler sollten vorhanden sein.

1985, 470 S., kart., DM 68, –
ISBN 3-7785-1134-3

„Die ProDOS Analyse" ist die umfangreichste und detaillierteste Darstellung, die jemals ein Apple-Betriebssystem erfahren hat.
Wer die „Innereien" von ProDOS bis zum letzten Byte, z.T. bis ins letzte Bit kennenlernen möchte, braucht dieses Buch. Das komplette Betriebssystem (Urlader, MLI, Disk-Driver, RAM-Disk-Driver und Uhr-Routine) mit Ausnahme des BASIC-SYSTEM wird mit umfangreichen Kommentaren und Übersichtstabellen disassembliert.
Auch die nicht im „Technical Reference Manual" aufgeführten Eigenschaften von ProDOS werden analysiert und beschrieben, z.B. die vertrackten eingebauten Testroutinen zur Identifikation der verschiedenen Apple II Modelle und *eventueller Nachbaugeräte*.
Programmierer, die ProDOS versionsabhängig „patchen" möchten, erhalten hier den genauen Überblick, wo was geändert werden muß, damit dies keine negativen Konsequenzen hat.

Arne Schäpers, Das BASIC SYSTEM, 1986, ca. 300 S., kart., DM 54, – ISBN 3-7785-1407-5
Sie wollen die erweiterten Möglichkeiten von ProDOS und dem BASIC-SYSTEM voll ausnutzen? Nur hier finden Sie

- *das Lesebuch:* eine schrittweise Erklärung der Zusammenarbeit zwischen Applesoft und dem Monitor, des Aufbaus der Stringverwaltung, mögliche Umleitungen der Ein-/Ausgabe und der Eingriffsmöglichkeiten in diese Abläufe. Die gezeigten Mechanismen werden durch zahlreiche (und sehr kurze) Beispielprogramme untermauert;
- *die Analyse:* eine minutiöse Sezierung des BASIC-SYSTEM zusammen mit der Kommandoschnittstelle zu ProDOS. Die Hauptfunktionen (Vektorbehandlung, TRACE-Kontrolle, FRE-Kommando, Global Page und benutzte Speicherstellen) sind jeweils in eigenen Abschnitten erläutert, ein kommentiertes Listing schließt diesen Teil ab;
- *Tips & Tricks:* die verborgenen Möglichkeiten des BASIC-SYSTEM („Input Anything", andere Dateinamen als STARTUP, RESET ohne CLEAR), der Aufbau „externer Kommandos". Dieser Teil enthält ein komplettes Rahmenprogramm für die Erstellung eigener Kommandos sowie die Erweiterung MONITOR, mit der (analog zu DOS 3.3) die Ausgabe von Kommandos und der Dateiverkehr auf dem Bildschirm sichtbar gemacht werden können.

☐ Schäpers, **Bewegte Apple Grafik,** ISBN 3-7785-1150-5, DM 58, –

☐ Begleitdiskette, DM 48, – ISBN 3-7785-1290-9

☐ Schäpers, **Pro-DOS-Analyse** ISBN 3-7785-1134-3, DM 68, –

☐ Schäpers, **Das BASIC-SYSTEM,** ISBN 3-7785-1407-5, DM 54, –

BESTELLCOUPON

Gewünschte Bücher bitte ankreuzen und an Dr. Alfred Hüthig Verlag, Postfach 102869, 6900 Heidelberg, schicken.

Name _____

Straße _____

Ort _____

Datum _____ Unterschrift _____

Hüthig

EDITORIAL

Neue Preise ab 1. 2. 1987

Normale Peeker-Sammeldisketten

#1, #2, #3*, #4, #5*, #6, #7, #9*, #10, #11, #12, #13, #14, #15, #17, #18, #19, #21, #22, #23, #24, #25, #27
je Diskette DM 22,–
*(fast) vergriffen

Spezielle Peeker-Sammeldisketten

#8 mit Diversi-DOS,	DM 26,–
#16 mit Macroeditor,	DM 26,–
#20 mit Pic-Edit,	DM 26,–
#26 mit Dossembler,	DM 26,–

Software mit Handbüchern

Fast-Writer	
ProDOS-Version,	DM 98,–
DOS-3.3-Version,	DM 98,–
Turtle-Graphics-Paket,	DM 98,–
Superquick,	DM 38,–
DISK40,	DM 38,–
INPUT 2.0,	DM 38,–
ProDOS-Editor,	DM 38,–
Double-Hires-Tools	
Applesoft-Version,	DM 28,–
Kyan-Pascal-Version,	DM 28,–

vergriffen

Softbreaker
Superplot
Turtle-Graphics-Quelltexte
Pic-Edit-Quelltexte

Kyan-Pascal mit Tools

keine Preisänderungen, jedoch nur noch lieferbar, solange Vorrat reicht

Wir möchten Ihnen heute mitteilen, daß der Peeker mit dem März-Heft eingestellt wird, denn der Apple II ist in Deutschland seit Mitte 1985 nach und nach durch andere Geräte verdrängt worden, insbesondere durch die IBM-PCs im Profi-Bereich und die Atari-Computer im Privat-Bereich. Der Apple IIgs hat die in ihn gesetzen Erwartungen leider nicht erfüllt, denn nach der Pressemitteilung vom 15. 9. 1986 ist das designierte Nachfolgemodell für den Apple II sprichwörtlich in der Versenkung verschwunden. Einer der Gründe hierfür ist ein defekter Custom-Chip, der wegen der Flat-Pack-Bauweise nicht einfach durch einen neuen Baustein ersetzt werden kann, zumal das Redesign eines Custom-Chips nicht von heute auf morgen vonstatten geht. Laut „PC-Woche", Heft 1/1987, müssen deshalb bei den bereits in den USA ausgelieferten 25 000 Geräten die gesamten Platinen ausgetauscht werden. Dies führt naturgemäß zu Verzögerungen bei der Software-Entwicklung bis weit über die Mitte dieses Jahres hinaus. Dann werden jedoch 32-Bit-Rechner, z. B. von Atari und IBM, auf den Markt drängen, so daß an einem 8/16-Bit-Zwitter in der Art des IIgs keiner mehr interessiert sein wird.

Es verwundert deshalb nicht, daß von den 180 potentiellen, deutschen Soft- und Hardware-Entwicklern, die wir vor Weihnachten angeschrieben haben, nur 3 Firmen IIgs-Produkte in Arbeit haben. Auch unseren 800 Kyan-Club-Mitgliedern müssen wir leider mitteilen, daß das IIgs-Kyan-Pascal-Projekt wegen der unsicheren IIgs-Zukunft nicht mehr weiterverfolgt wird.

Aus alledem wird verständlich, daß wir den Peeker nicht mehr fortführen können. Für die beiden letzten Hefte haben wir uns jedoch noch einige Bonbons ausgedacht, die Ihnen den Abschied versüßen sollen. Auf unseren Sammeldisketten #25 bis #27 (Januar bis März) erscheinen der professionelle 6502-Assembler „DOSSEMBLER" von Steffen Holzinger sowie das Dateiprogramm „DB-Meister". Ferner konnten wir mit der Firma Frank & Britting ein neues, sehr günstiges Festplatten-Angebot aushandeln, das jedoch bis zum 31. 3. 1987 befristet ist. Schließlich haben wir die Preise für die Produkte aus dem Hüthig Software Service per 1. 2. 1987 gesenkt. Beachten Sie jedoch auch hier, daß die Programme nur noch eine begrenzte Zeit lieferbar sein werden, so daß Sie sich vorsorglich bis Ende März eindecken sollten.

Einige bereits geplante Projekte werden nicht mehr realisiert. Dazu gehört insbesondere der GFABASIC-Kompaktkurs sowie die in diesem Zusammenhang angekündigte Übungsdiskette. Auch die Hilfsprogramme und Quelltexte zum „DB-Meister" werden nicht mehr veröffentlicht. Die Haupt-Module sind jedoch auf den Sammeldisketten #25 bis #27 enthalten. Ferner wird der 65816-Kurs noch komplett erscheinen, da er von allgemeinem Interesse ist. Im übrigen haben wir auch das Peeker-Aufsatzverzeichnis fortgeführt, das für die Monate 10/1986 bis 3/1987 auf der Sammeldiskette #27 enthalten sein wird.

Die Atari-ST-Besitzer unter unseren Peeker-Lesern wird es freuen zu erfahren, daß ab Februar 1987 in unserem Verlag ein „schnelleres ST-Diskettenmagazin" namens „FaSTer Disk Mag" erscheinen wird, das nur in Diskettenform erhältlich ist und neben ST-Programmen mit entsprechender Dokumentation auch beispielsweise Einführungskurse zu Pascal und C als Textfiles enthalten wird (Preis für 1 „FaSTer Disk Mag"-Diskette DM 24,80; Preis für 6 Disketten DM 130,–).

Ulrich Stiehl

INHALT

„Frühjahrsputz"

Impressum

Peeker
Magazin für Mikrocomputer
4. Jahrgang 1987
ISSN 0176-9200

Verleger und Herausgeber:
Dipl.-Kfm. Holger Hüthig
Geschäftsführung Zeitschriften:
Heinz Melcher
Chefredakteur: Ulrich Stiehl (us)
Redaktion: Dagmar Berberich
Anzeigenleitung: Karl M. Dietzow
Anzeigendisposition: Diana Walter

Hüthig
PUBLIKATION

Telefonnummern:
Zentrale: 06221/489-0
Redaktion: 06221/489-352
Anzeigen: 06221/489-206
Abonnement: 06221/489-283
Software: 06221/489-231
Bücher: 06221/489-353
(Bestellungen bitte nur schriftlich)

Abonnement:
Der Abonnent kann seine Bestellung innerhalb von 7 Tagen schriftlich durch Mitteilung an den Dr. Alfred Hüthig Verlag GmbH, Postfach 10 28 69, 6900 Heidelberg 1, widerrufen. Zur Fristwahrung genügt die rechtzeitige Absendung des Widerrufs (Datum des Poststempels). Das Abonnement verlängert sich zu den jeweils gültigen Bedingungen um ein Jahr, wenn es nicht zwei Monate vor Jahresende schriftlich gekündigt wird. Die Abonnementsgelder werden jährlich im voraus in Rechnung gestellt, wobei bei Teilnahme am Lastschriftabbuchungsverfahren über die Postscheckämter und Bankinstitute eine vierteljährliche Abbuchung möglich ist.
Nichterscheinen infolge höherer Gewalt berechtigt nicht zu Ansprüchen gegen den Verlag.

PEEKER

Heft 2/1987

Assembler

DOSSEMBLER
Ein ProDOS-Editor-Assembler
für den Apple II
von Steffen Holzinger 6

65816/65802-Kompaktkurs
für Apple IIgs, Teil 1
von Arne Schäpers 12

BRK – nur zum Anhalten?
von Michael G. Schneider 43

Drucker

Grafikausdruck anschaulich erklärt
Mit Apple- und Atari-Algorithmen
für den 24-Nadel-Drucker-LQ-800
von Ulrich Stiehl 29

Individueller Zeichensatz
für den Epson FX-80
von Dipl.-Ökonom Edgar Meyzis 38

Applesoft

Leistungsfähiger Pattern-Matcher
für Applesoft-Strings
von Dipl.-Inform. Dieter Greiner 52

UCSD

Long-Integer-Routinen
in UCSD-Pascal
von Thomas Schneider 58

Readin-Prozedur für Ganzzahlen
Eine Utility für UCSD-Pascal
von Norbert M. Doerner 60

UCSD-Pascal-Bildschirm-Dump
für die Ehring-V80-Karte
von Dr. U. Hesse 61

ProDOS

ProDOS-RAM-Disk-Driver
für die OHO-512K-RAM-Karte
von Stefan Hußfeldt 62

Produkte

MICOL-BASIC

Eine strukturierte Compiler-Sprache
getestet von Paul Batt 66

Bücher 68

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

Auslieferung für die Schweiz:
Delta-Verlag
Herr R. de Forest
Gugelmattstraße 31
8967 Widen
Telefon 057 / 33 86 86

Vertrieb:
Erscheinungsweise: 12 Hefte jährlich,
Erscheinungstag jeweils 1 Woche vor Monatsbeginn.
Jahresabonnement Inland DM 75,–, einschl. MwSt und Versandkosten.
Jahresabonnement Ausland DM 75,– plus DM 20,– Versandkosten.
Einzelheft DM 6,50
Vertrieb Handel:
Vertriebsleitung:
Walter Menzel, Tel. (0 62 21) 48 92 80

Bankverbindungen:
Zahlungen: an den Dr. Alfred Hüthig Verlag GmbH, D-6900 Heidelberg 1: Postgirokonten: Ludwigshafen 4799-673, BLZ 545 100 67; Österreich: Wien 7555 88; Schweiz: Basel 40-24417-4; Niederlande: Den Haag 1 457 28; Italien: Mailand 5 968 92 08; Belgien: Brüssel 07 230 26-85; Dänemark: Kopenhagen 603 4969; Norwegen: Oslo 199 4243; Schweden: Stockholm 5477 76-5
Bankkonten: Landeszentralbank Heidelberg 67 207 341; BLZ 672 000 00; Deutsche Bank Heidelberg 02 65 041; BLZ 672 700 03; Bezirkssparkasse Heidelberg 204 51, BLZ 672 500 20.

Herstellung:
Produktionsleitung: Gunter Sokollek
Gestaltung: Rainer Schmitt
Titelbild: Werner Hable
Satz und Druck:
Heidelberger Verlagsanstalt
Printed in Germany

DOSSEMBLER

Ein ProDOS-Editor-Assembler für den Apple II

von Steffen Holzinger

Bedienungsanleitung

Der DOSSEMBLER, den nur *Fortsetzungsbezieher* der Sammeldisketten als Treueprämie automatisch mit der Sammeldisk #26 zu diesem Heft 2/87 *zusätzlich* erhalten, ist ein vollständiger 6502/65C02-Assembler, der unter dem ProDOS-Betriebssystem läuft und somit einen 64K-Apple-II+ (mit oder ohne Videx-Karte) oder einen IIe/c (mit oder ohne 80-Zeichenkarte) voraussetzt. Da die Sammeldisk #26 im DOS-3.3-Format vorliegt, verfahren Sie wie folgt:

1. Dateien ASM.SYSTEM (= eigentlicher DOSSEMBLER-Editor/Assembler) und ASM.GP (DOSSEMBLER-Global-Page) mit CONVERT oder DOSTOPRO auf Ihre ProDOS-Arbeitsdiskette konvertieren.
2. Da durch die Konvertierung BIN-Dateien entstehen, müssen Sie diese in SYS-Dateien umwandeln:

```
BLOAD ASM.SYSTEM, A$2000
DELETE ASM.SYSTEM
CREATE ASM.SYSTEM, TSYS
BSAVE ASM.SYSTEM, TSYS, A$2000, L14848
BLOAD ASM.GP, A$0300
DELETE ASM.GP
CREATE ASM.GP, TSYS
BSAVE ASM.GP, TSYS, A$0300, L144
```

Danach können Sie den DOSSEMBLER mit

```
-ASM.SYSTEM
```

starten. Dies sollten Sie jetzt auch tun, damit Sie die nachfolgende Anleitung Schritt für Schritt nachvollziehen können.

1. Datumeingabe

Ist eine Hardware-Uhr eingebaut, so wird nach dem Start des Assemblers (s.o.) deren Datum und Zeit angezeigt und verwendet. Ist in einem vorherigen Systemprogramm (z.B. Appleworks) das Datum definiert worden, wird es angezeigt und kann bei Bedarf verändert werden. War vorher kein Datum definiert (z.B. nach dem Booten), kann der Benutzer das Datum eingeben, wobei das Geburtsdatum des Programmautors als Beispiel dient. Unkorrekte Eingaben werden mit einem Piepser quittiert und können wiederholt werden. Danach erscheint am Bildschirm das Hauptmenü.

2. Hauptmenü

2.1. Eingaben im Hauptmenü

Verlangt das Hauptmenü eine Eingabe (z.B. eines Pfadnamens), so verfährt die Eingaberoutine wie folgt:

– Ist ein Vorgabewert vorhanden, kann er durch „Return" vollständig übernommen werden oder mit → editiert werden.
– Die Pfeiltasten ← und → bewegen den Cursor, die Del-Taste (bzw. Ctrl-P beim Apple II+) löscht das Zeichen links vom Cursor.
– „Return" übernimmt die gesamte Zeile unabhängig von der Cursorposition, während Ctrl-T ab der Cursorposition abschneidet.
– Wird ein Pfadname verlangt, akzeptiert das Programm nur Buchstaben, Ziffern sowie Punkt (.) und Schrägstrich (/).
– Die maximale Eingabelänge beträgt beim 40-Zeichen-Terminal 22 Zeichen, beim 80-Zeichen-Terminal 40 Zeichen.
– Durch Drücken von „Esc" kann die Eingabe gestrichen werden.

2.2. Hauptmenü-Optionen

Die Optionen des Hauptmenüs werden durch Tippen des jeweiligen Anfangsbuchstabens ausgewählt.

<**W**> Work-File definieren:
Existiert der angegebene File, wird er geladen. Existiert er nicht, erscheint „New file".
<**M**> Main-File definieren:
Der „Main-File" ist die Hauptdatei einer Anzahl von Include-Files und wird, falls definiert, vom Assembler als Anfangsdatei verstanden. Ist er nicht definiert, benutzt der Assembler den Work-File (analog zu Turbo-Pascal).
<**P**> Präfix ändern:
Mit diesem Befehl kann das momentane ProDOS-Präfix geändert oder gelöscht werden. Das Programm prüft nach, ob das betreffende Directory auffindbar ist.
<**E**> In den Editor gehen:
Falls kein Work-File definiert ist, wird zunächst <W> ausgeführt. Danach wird der eigentliche Editor aufgerufen.
<**A**> Den Assembler aufrufen:
Falls weder Work-File noch Main-File definiert sind, wird nach dem Source-File gefragt. Danach fragt Dossembler nach dem Namen des ersten erzeugten Object-Files. Vorgabewert ist, falls der Platz ausreicht,

der Source-Name und das Suffix „.OBJ0". Je nach den für die Assemblierung gewählten Ausgabekanälen fragt das Programm noch nach dem Namen des Listing-Files. Vorgabewert ist der Source-Name mit dem Suffix „.PRN". Nach dieser Prozedur wird der eigentliche Assembler aufgerufen.

<**O**> Ausgabekanäle wählen:
Für den Editor und das Assembler-Listing können voneinander unabhängige Ausgabekanäle ein- und ausgeschaltet werden. Dazu gehören der Bildschirm, ein eventuell vorhandener Drucker und (beim Assembler) ein Listing-File. Mit „Y" und „N" kann jeweils ein- und ausgeschaltet werden, „Return" bestätigt die Wahl. Wird der Drucker angewählt, so muß er bereits hardwaremäßig eingeschaltet sein.

<**S**> Work-File speichern:
Speichert den im Speicher befindlichen Work-File auf Diskette.

<**D**> Directory ausgeben:
Eingabe des Directory-Pfadnamens. Vorgabewert ist das definierte Präfix. Durch Eingabe von „#s,d" kann das Volume-Directory der Unit in Slot s, Drive d ausgegeben werden. Ist das Directory etwas länger, wird es in verschiedene Bildschirmseiten aufgeteilt. Dabei kann jeweils mit „Esc" abgebrochen werden. Ist der Drucker durch die <O>-Option eingeschaltet worden, wird das Listing auf dem Drucker ausgegeben.

<**Q**> Programm verlassen:
Das Programm springt in die ProDOS-Reboot-Kommando-Routine; eine neue Anwendung kann ausgewählt werden.

2.3. Hauptmenü-Fehlermeldungen

2.3.1. Allgemeine Fehlermeldungen

Zu den allseits bekannten Fehlermeldungen gehören „I/O Error", „Write protected", „Volume or directory not found", „File not found", „File locked" sowie „Volume full". Ihre Bedeutung muß hier nicht näher beschrieben werden.

2.3.2. Spezielle Fehlermeldungen

„File too large" erscheint, wenn versucht wird, einen File einzuladen, der länger als der momentan verfügbare Speicher ist.

„Wrong file type" bricht den Versuch ab, einen File einzuladen, der nicht vom Typ „TXT" ist.

„Not a directory file" tritt auf, wenn versucht wird, einen File als Directory auszugeben, der weder vom Typ „DIR" noch ein Volume-Directory ist.

„DOS Error! Code = $xx" erscheint bei allen MLI-Fehlermeldungen, die nicht durch spezielle Meldungen abgedeckt werden. Die Bedeutung der Fehlernummer xx kann in jedem ProDOS-Buch nachgesehen werden (z.B. U. Stiehl: ProDOS für Aufsteiger, Bd. 1).

3. Der Editor

Der Editor ähnelt dem Big-Mac-Editor, einige Befehle wurden jedoch weggelassen, andere hinzugefügt oder erweitert. Alle Kommandos können in Groß- oder Kleinbuchstaben eingegeben werden. Die Befehlseingaberoutine funktioniert wie bei der Routine im Hauptmenü, siehe 2.1.

3.1. Befehls- und Syntaxtabelle

Kleingeschriebene Befehlswortteile können weggelassen werden, ebenso spitz eingeklammerte Syntaxteile. Es bedeuten: „filename": Pfadname eines Source-Files „str": Durch beliebige Zeichen begrenzter String (im Gegensatz zum Big Mac wahlweise mit Apostrophen oder Gänsefüß-

Befehls- und Syntaxtabelle

Syntax mit Beispiel(en)	Funktion
Add <filename> ADD /VOL/FILE	Anhängen von Text oder File an den Text im Speicher.
Change loc strl str2 C 1Ø-2Ø .A.B.	Ersetzt "Strl" durch "Str2". Bei "Some" muß jeweils mit "Y" oder "N" geantwortet werden.
COPY zeile <-zeile> TO zeile COPY 1Ø-2Ø TO 5Ø	Textblock kopieren.
Delete loc D 1,2,4Ø-	Zeilen löschen.
Edit loc <str> E .EQU. E 1Ø-2Ø	Zeilen, in denen "str" vorkommt, editieren.
Find loc <str> F 5Ø- .ORG.	Zeilen, in denen "str" vorkommt, ausgeben.
Insert zeile <filename> I 14 I 14 /VOL/FILE	Text oder File vor "zeile" einfügen.
List loc L 1Ø-2Ø	Zeilen auflisten
MONitor MON	In den Monitor springen (Rückkehr in den Editor mit Ctrl-Y).
MOVE zeile <-zeile> TO zeile MOVE 1Ø-2Ø TO 5Ø	Textblock verschieben.
NEW NEW	Speichertext löschen.
PRinterOFf PROF	Drucker ausschalten.
PRinterON PRON	Drucker einschalten.
Print loc P 1Ø-2Ø	Zeilen ohne Zeilennummer auflisten.
REName filename REN /VOL/TEST	Work-File umbenennen.
Replace loc <filename> R 1Ø-2Ø /VOL/FILE	Zeilen löschen, dann Text oder File einfügen.
Tabs t1,t2,t3 T 1Ø,2Ø,3Ø	Tab-Werte für den Editor neu setzen (T 1,1,1 löscht die Tabs)
TERMinal num TERM 3	Terminal "num" einschalten: num = Ø: Apple II+ 4Ø Zeichen num = 1: Videx-8Ø-Zeichenkarte num = 2: Apple IIe/c 4Ø Zeichen num = 3: Apple IIe/c 8Ø Zeichen Bitte die Terminals nur einschalten, wenn sie verfügbar sind.
TRuncOFf TROF	Kommentare sollen mit aufgelistet werden.
TRuncON TRON	Kommentare sollen abgeschnitten werden.
USeR USR	Springt nach $3F5, dort kann eine eigene Kommando-Routine installiert werden.
Where zeile W 1Ø	Gibt die Speicheradresse von "zeile" aus.

chen begrenzt!)
„zeile“: Zeilennummer (dezimal)
„num“: Zahlenangabe (dezimal)
„loc“: Zeilen, auf die sich das Kommando bezieht

Beispiel für „loc“:
List (ganzen Text listen)
List 10 (nur Zeile 10 listen)
List 10,20 (Zeilen 10 und 20 listen))
List -10 (Von Anfang bis Zeile 10)
List 10- (Von Zeile 10 bis Ende)
List 10-20 (Zeilen 10 bis 20)
List -10,20,30,40-50,70- (entsprechend)

Befehlstabelle

Ein eingeschalteter Drucker wird bei folgenden Kommandos aktiv:

Change, Find, List, Print, Where.
Die Ausgabe kann im Editor mit der Leertaste zeilenweise unterbrochen und mit Ctrl-C ganz abgebrochen werden.
Der Editor wird mit „Esc“ verlassen.

3.2. Zeileneditor

Bei den Kommandos Add, Insert, Replace und Edit kommt der Benutzer in Berührung mit dem Zeileneditor. Er besitzt die in der Tabelle gelisteten Control-Funktionen.

3.3. Fehlermeldungen im Editor

Neben den im Hauptmenü verwendeten Fehlermeldungen existieren noch spezielle Editor-Fehlermeldungen:

„Syntax error“ erscheint, wenn der eingegebene Befehl nicht existiert oder nicht die erforderlichen Parameter mitgeliefert wurden.
„Overflow error“ zeigt, daß bei Zeilennummern oder Zahlenkonstanten ein Wert über 65519 eingegeben wurde.
„Range error“ meldet, daß Parameter außerhalb des zulässigen Rahmens liegen, z.B. nicht existente Zeilennummern.
„Memory full“ erscheint, wenn durch Texteingabe oder andere speicherplatzmindernde Kommandos der Textpuffer vollständig gefüllt wurde.

4. Der Assembler

4.1. Allgemeines

Der Assembler kennt alle Opcodes des 6502 sowie die beiden künstlichen Opcodes BLT und BGE als Ersatz für BCC und BCS. Beim Einschalten der 65C02-Option „lernt“ der Assembler noch die neuen Opcodes des 65C02, die im Peeker 1/84, Seite 6ff. beschrieben wurden. Insgesamt beherrscht Dossembler damit 15 verschiedene Adressierungsarten.
Zu beachten ist, daß kein Label „A“ definiert werden sollte, da es mit der Akkumulator-Adressierung verwechselt werden kann. Ein akku-adressierter Befehl kann entweder als „ASL A“ oder einfach als „ASL“ geschrieben werden, beides ist korrekt.
Eine Zeile darf höchstens 255 Zeichen lang sein. Zwischen Spalten muß mindestens *ein* Leerfeld oder Ctrl-I stehen. Der Source-Code darf in Groß- oder Kleinbuchstaben geschrieben werden.

4.2. Pseudo-Opcodes

Spitz eingeklammerte Syntaxteile sind optional.
Es bedeuten:
„expr“: arithmetischer Ausdruck (s. 4.3.1.)
„e.expr“: erweiterter arithmetischer Ausdruck (s. 4.3.2.)
„str“: durch Sonderzeichen begrenzter String
„file“: Pfadname
„status“: ON oder OFF
„label“: Label
Näheres kann der gesonderten Tabelle entnommen werden.

Die Opcodes DSECT und DEND wurden vom Apple-Editor/-Assembler übernommen. Für die Verwendung ein Beispiel:

```
        DSECT
        ORG     $5Ø
ADRL    DS      2
ADRH    EQU     ADRL+1
TEMPØ   DS      1
TEMP1   DS      2
        DEND
```

Es ist möglich, beispielsweise Labels ohne feste Adressenzuordnung zu definieren. Muß das Datensegment später verschoben werden, kann in der ORG-Zeile eine neue Adresse angegeben werden. Allerdings muß nicht jedes Label umdefiniert werden. Ähnliche Anwendungen sind entsprechend denkbar.

4.3. Arithmetische Ausdrücke

4.3.1. Einfache arithmetische Ausdrücke

Der Assembler verarbeitet arithmetische Ausdrücke mit den 5 Operatoren + (Addition), - (Subtraktion), * (Multiplikation), / (ganzzahlige Division) und & (bitweises logisches UND). Die Operanden können Konstanten (dezimal: ohne Markierung, hexadezimal: mit führendem „$“, dual: mit führendem „%“), Labels (* gilt als PC) oder Textzeichen sein.
Ein rechnerischer Überlauf wird nicht erkannt. Ausdrücke werden von links nach rechts durchgerechnet, Klammern sind nicht erlaubt.

Beispiele:

```
ORG 1Ø
ORG $1Ø
ORG %1Ø

ORG ADR
ORG ADR+$1Ø
ORG ADR+$1Ø-%1Ø1Ø

ORG 'A
ORG 'A'+ADR
ORG ADR+1Ø-TMP/%1Ø1Ø*$1Ø+'A'
usw.
```

4.3.2. Erweiterte arithmetische Ausdrücke

Erweiterte arithmetische Ausdrücke geben dem Benutzer die Möglichkeit, den Ausdruck zu verfeinern: Ein vorgestelltes „<“ zeigt an, daß nur das Low-Byte des Ausdrucks gewertet werden soll. Entsprechend gilt „>“ und „/“ für das High-Byte. Ein „#“ kann jederzeit vorangestellt werden.

Beispiele:

```
DFB <LABEL
DFB #>LABEL+4
DFB #/LABEL

DFB #-1Ø
DFB *+1Ø-LABEL
DFB $1Ø-2Ø+%1Ø1Ø
usw.
```

4.4. Assembler-Fehlermeldungen

„Illegal line format“:
Eine Zeile ist länger als 255 Zeichen (oder endet nicht auf $0D).
„Illegal file type“:
Der Source-File ist nicht vom Typ „TXT“.
„MLI Error #$xx“:
Der ProDOS-MLI-Fehler Nr. xx trat auf.

Zeileneditor

Taste(n)	Funktion
Ctrl-B	Springt an Zeilenanfang
Ctrl-D	Löscht Zeichen unter Cursor
Ctrl-E	Springt an Zeilenende
Ctrl-F chr	Setzt Cursor auf nächstes "chr" (1 Zeichen)
Ctrl-H oder <-	Setzt Cursor ein Zeichen nach links
Ctrl-I oder TAB	Schaltet Einfügemodus ein (Ausschalten durch ein beliebiges Ctrl-Zeichen)
Ctrl-M oder RETURN	Übernimmt die gesamte Zeile
Ctrl-R	Stellt ursprüngliche Zeile wieder her (nur bei EDIT)
Ctrl-T	Schneidet Zeile ab Cursorposition ab
Ctrl-U oder ->	Setzt Cursor ein Zeichen nach rechts
Ctrl-X	Bricht Editieren ab
Ctrl-P oder DEL	Löscht das Zeichen links vom Cursor
ESC	Schaltet Tabulierung ab der dritten Spalte ein/aus

„Include file stack overflow“:
Es wurde versucht, mehr als vier Include-Files ineinander zu verschachteln.
„Bad label“:
Ein Label enthält illegale Zeichen, z.B. + - * /
„Symbol table full“:
Die Symboltabelle ist voll, die Grenzen des Assemblers überschritten.
„Bad expression“:
Ein arithmetischer Ausdruck ist fehlerhaft aufgebaut.
„No such label“:
Ein undefiniertes Label wurde verwendet.
„Duplicate symbol“:
Es wurde versucht, ein Label zweimal zu definieren.
„Bad DSECT/DEND“:
Es wurde versucht, DSECT/DEND-Konstruktionen zu schachteln.
„Bad opcode“:
Ein nicht existierender Opcode wurde verwendet.
„Bad address mode“:
Eine zwar existierende, aber für diesen Opcode nicht zugelassene Adressierungsart wurde verwendet.
„Bad operand“:
Eine nicht existierende Adressierungsart wurde verwendet.
„Bad branch“:
Die Distanz für einen relativen Sprung war zu groß.

4.5. Abbruch

Die Assemblierung kann durch die Leertaste zeilenweise angehalten und durch Ctrl-C ganz abgebrochen werden.
Jeder MLI-Fehler führt zum Abbruch.

5. Sonstiges

5.1. RESET

Das Drücken von RESET bewirkt, daß das Programm (normalerweise ohne Datenverlust) verschiedene Initialisierungen vornimmt und danach nach seiner Startdiskette sucht. Diese muß unbedingt vorhanden sein.
RESET sollte nicht während eines Diskettenzugriffs oder während der Abarbeitung von Editor-Kommandos benutzt werden.

5.2. Hardware-Konfiguration

Das Programm setzt einen Apple II ab Version II+ voraus, auf dem ProDOS lauffähig ist. Eine 80-Zeichenkarte wird nicht benötigt, kann aber benutzt werden (Apple oder Videx). Ein Drucker wird ebenfalls nicht benötigt, ist aber voll in das Programm integriert. Eine Uhr wird nicht benötigt, aber erkannt und benutzt. Jeder ProDOS-kompatible Massenspeicher ist verwendbar. Ein Kleinschreib-Umrüstsatz (II+) wird erkannt.
Eine 80-Zeichenkarte wird in Slot 3 erwartet, die Videx-Karte kann aber nach einer Installation auch in einem anderen Slot untergebracht werden, ebenso wie das Drucker-Interface, das in Slot 1 erwartet wird. Die Slot-Werte sind in der Global Page eingetragen.

5.3. Global Page

Alle Programmkonstanten und Konfigurationsdaten sind in der Dossembler Global Page „ASM.GP“ gespeichert und können dort verändert werden. Die Beschreibung der Global Page würde den Rahmen dieses Artikels sprengen.

Pseudo-Opcodes

Syntax/Beispiel	Funktion
label EQU e.expr	Definition eines Labels
ABC EQU *-5	
INCLUDE file	Include-File einbinden
INCLUDE /VOL/FILE	
ASM file	"file" assemblieren (neuer ASM-Start)
ASM /VOL/FILE	
ASC str	String zeichenweise einbinden
ASC 'Peeker'	
DCI str	wie ASC, beim letzten Zeichen Bit 7 invers
DCI 'Dossembler'	
STR str	wie ASC, jedoch mit führendem Längenbyte
STR 'Apple'	
LST status	Assembler-Listing ein/ausschalten
LST OFF	
MSB status	Bit 7 setzen/löschen (bei Strings)
MSB ON	
DO e.expr	Folgendes nur assemblieren, wenn "e.expr" = 1
DO TEST	
FIN	Ende des konditionalen Assemblierens
FIN	
ELSE	Letzte DO-Bedingung negieren
ELSE	
END	Assemblierung hier abbrechen
END	
HEX hl <,h2,h3,...>	Hex-Zahlen einbinden
HEX Ø1,Ø2,Ø3	(Komma kann auch weggelassen werden)
ORG expr	PC auf "expr" setzen, neuer OBJ-File
ORG $2ØØØ	
PHS expr	PC auf "expr" setzen
PHS $3ØØØ	
OBJ expr	bei Dossembler bedeutungslos
OBJ $4ØØØ	
DSECT	Beginn einer "dummy section", s. 4.2.
DSECT	
DEND	Ende einer "dummy section", s. 4.2.
DEND	
DS expr	Leerraum definieren
DS $3Ø	
DFB e.expr <,e.expr,...>	Einzelne Datenbytes einbinden
DFB 1Ø,$2Ø,'B',5-3	
DW e.expr <,e.expr,...>	Doppelbytes einbinden (low byte first)
DW TEMP,$1234	
DA e.expr <,e.expr,...>	" " "
DA TEMP,$1234	
DDB e.expr <,e.expr,...>	" " (high byte first)
DDB TEMP,$1234	
PREFIX file	Setzt ProDOS-Präfix auf "file"
PREFIX /DISK	
TR status	OBJ-Abschneiden ein aus
TR ON	(z.B. bei langen Strings)
PAGE	gibt FF ($ØC) auf Drucker aus
PAGE	
SKP e.expr	gibt "e.expr"-mal CR ($ØD) aus
SKP 5	
SYS	definiert nächste OBJ-Files als Typ "SYS"
SYS	
REL	" " " " " "REL"
REL	
BIN	" " " " " "BIN"
BIN	(Standardeinstellung)
CHR str	Speichert erstes Zeichen von "str" als
CHR '-'	Trennzeichen
RPT e.expr	gibt das Trennzeichen "e.expr"-mal aus
RPT 4Ø	
65Ø2	schaltet auf 65Ø2-Opcodes
65Ø2	
65CØ2	schaltet auf 65CØ2-Opcodes
65CØ2	

6. Übungsbeispiel

1. Wir starten den Assembler mit
-ASM.SYSTEM

2. Wenn das Datum-Menü erscheint, tippen wir „n“ oder „N“, falls wir das alte Datum nicht ändern wollen.

3. Nun gelangen wir in das eigentliche Haupt-Menü. Hier tippen wir „e“ oder „E“ für „Editieren“. Da noch kein Dateiname (Work-File) spezifiziert wurde, springt der Cursor automatisch in die entsprechende Spalte. Hier geben wir „TEST“ ein. Danach gelangen wir in den leeren Editierbildschirm.

4. Wir tippen nun „a“ oder „A“ für „Add“ und geben dann das folgende Beispiel ein:

```
 ORG $1ØØØ
COUT EQU $FDED
 MSB ON ;für String
 LDX #Ø
LOOP LDA STRING,X
 BEQ ENDE
 JSR COUT
 INX
 BNE LOOP
ENDE RTS
STRING ASC 'HALLO'
 HEX 8DØØ
```

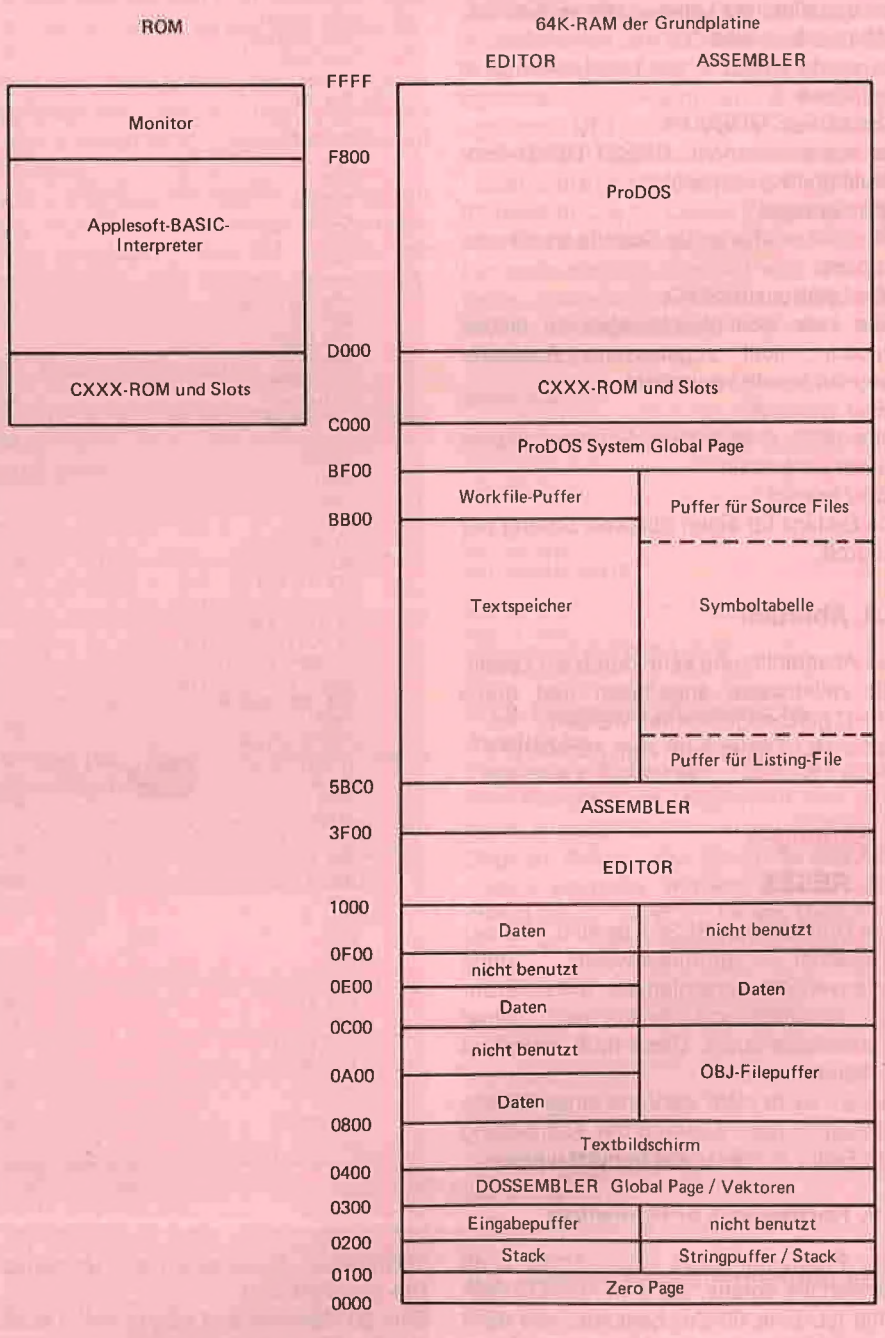
Das obige Beispiel zeigt genau das an, was Sie tippen müssen. Am Bildschirm sehen Sie jedoch, daß automatisch Zeilennummern vergeben werden und im übrigen mit jeder Leertaste automatisch in das nächste Feld tabuliert wird. Am Schluß nach Eingabe der Zeile „ HEX 8D00“ tippen Sie einfach ein nacktes Return. Damit verlassen Sie den Editor.

5. Mit „l“ oder „L“ für „List“ wird der soeben eingegebene Quelltext am Bildschirm gelistet. Nun tippen Sie auf die ESC-Taste und gelangen damit wieder in das Haupt-Menü. Hier geben Sie „s“ oder „S“ für „Save“ ein, womit der Quelltext automatisch unter dem früher angegebenen Namen „TEST“ auf Diskette gespeichert wird.

6. Nun tippen wir auf „a“ oder „A“ für „Assemble“, worauf der Cursor in die Objektcode-Spalte springt und „TEST.OBJO“ ausweist. Es genügt, wenn Sie jetzt einfach auf die Return-Taste drücken. Danach erfolgt die Assemblierung, die Sie am Bildschirm verfolgen können.

7. Wir verlassen nun den DOSSEMBLER über „q“ oder „Q“ für „Quit“ und geben danach als SYS-Datei „BASIC.SYSTEM“ ein. Vom BASIC.SYSTEM aus können wir dann später mit
-TEST.OBJO
unser Übungsprogramm starten, das „HELLO“ am Bildschirm ausgibt.

Speicherverteilung



ATARI ST

ASSEMBLER-PRAXIS AUF ATARI ST

Roland Löhr
„ein Altmeister der Assembleranwendung, Herausgeber des Mikrocomputer-Magazins MICRO MAG, veröffentlicht bei te-wi seine souveräne Darstellung der Assemblerprogrammierung auf ATARI STs.

Erklärt Grundlagen:

Begriffe und Werkzeuge der Assemblerprogrammierung...erforderliche Systemkenntnisse...systembezogene Erläuterung der 68000er Befehlsfunktionen.

Zeigt Anwendungen:

Hantieren mit Assemblern: Aufruf von Assemblern; Steuern ihrer Optionen über Direktiven; Stellungnahme zu realen ATARI-ST-Assemblern.
Arbeiten in der ATARI-ST-Programmierumgebung: Textprogramme zur Programmentwicklung; ein Editor; ein Parser; das Betriebssystem; BIOS-Funktionen; BIOS-Toolbox; GEMDOS Toolkit; das erweiterte XBIOS.
Anwenden des Befehlssatzes in Musterprogrammen für: E/A-Routinen, Rekursionen, dez/bin Rechenarten, Stackverwaltung, Adressverwaltung, Entscheidungen, Schleifenkonstrukte, Unterprogramme, numerierte Traps, Bedienen von Interfacebausteinen, Texterkennung, Textverarbeitung, Tastaturdekodierung, memory dumps, Floppy-Tests/Funktionen, serielle RS232-Datenübertragung usw.

Entwickelt Hilfsprogramme:

BIOS-Toolbox; GEMDOS-Toolkits; ein Editor; ein Parser; Arbeiten mit Toolkits.
Die Programme des Buchs sind auf Diskette vom Autor erhältlich.

Ein Fachtext in klarer Sprache mit leserfreundlichem Druckbild, guter Bilddokumentation und umfangreichen Listings von Musterprogrammen (auf Diskette beim Autor erhältlich).

ca. 300 Seiten, Softcover, DM 59,–

tewi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Weitere te-wi-Bücher

NEU

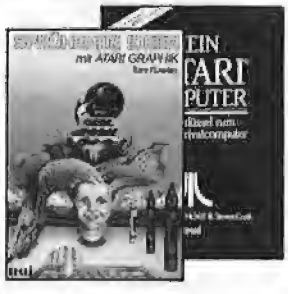
DAS „C"-BUCH
(Herold /Unger)
Ein „C"-Kurs der Industrie. Für sämtliche C-Konstrukte. Über 100 Beispiele. Anspruchsvoll in Text-/Bildmaterial, ca. 500 Seiten, Softcover, DM 79,–

UNIX
(Yates/Thomas) US-Standardwerk der UNIX-Promoterin Yates. Eine sachkundige Übersicht und Einführung in die Anwendung, 550 Seiten, Softcover, DM 79,–

LOGO – Jeder kann programmieren
(Daniel Watt)
Buch des Jahres in den USA.
Best-rezensiert von Pädagogen und deutschen Kultusministerien. Ein bildreicher Führer durch u. a. ATARI's LOGO. Von Papert's Schüler D. Watt.
384 Seiten, A4, DM 59,–

M68000 FAMILIE, 2 Bd,
Hilf/Nausch, ges. 968 Seiten
Einzige Motorola-authentische Darstellung von CPU-68000-Architektur, Programmierung, Systemaufbauten. Behandelt alle 68000-Bausteine sowie 68020, 68881.
Bd 1, Grundlagen + Architektur, 568 Seiten, DM 79,–
Bd 2, Anwendung und Bausteine, 400 Seiten, DM 69,–

UMWELTDYNAMIK
30 Programme für kybernetische Umwelterfahrungen auf allen BASIC-Rechnern. Das Buch enthält beides: Ein Programmsystem zur Simulation eigener Problemformulierungen und 29 kommentierte Modellbeispiele wie Baumsterben, Heizungsbedarf, Nahrungsketten usw. Prospekt anfordern.
Von Hartmut Bossel, 480 Seiten, Softcover, DM 59,–



Mein ATARI Computer
Best-rezensiertes Standardwerk deutscher ATARI-User-Groups. Kompakte ATARI 400-/800-System/Peripheriebeschreibung.
Von Poole/McNiff/Cook, 500 Seiten, Softcover, DM 59,–

Sprühende Ideen mit ATARI-GRAPHIK
Fröhlicher Lehrstoff in Geometrie und Farbenlehre eines amerikanischen Lehrers mit ATARI Graphikmöglichkeiten.
Von Tom Rowley, 224 Seiten, Softcover, DM 49,–

6502 – Programmieren in Assembler
Dieses Buch behandelt ausführlich die Assemblersprachen-Programmierung für den weitverbreiteten Mikroprozessor 6502.
Lance Leventhal, 704 Seiten, Softcover, DM 59,–

fm 4645

Noch im Programm: Einführung in die Mikrocomputer-Technik, DM 66,–
Computer für Kinder, ATARI, DM 29,80

65816/65802-Kompaktkurs für Apple IIgs

Teil 1

von Arne Schäpers

Dieser Kurs, der aus einer allgemeinen Beschreibung des Prozessors mit einigen IIgs-spezifischen Anwendungsbeispielen besteht, muß wegen des großen Umfangs auf zwei Peeker-Hefte verteilt werden. Der Kurs stellt (wie die bereits zur 6502-, Z80- und 68000-Programmierung erschienenen Kurse) keine komplette Einführung in die Programmierung dar. Vom Leser wird erwartet, daß er mit mindestens einem anderen Prozessor gewisse Erfahrungen gesammelt hat und mit grundlegenden Konzepten wie Registern, Speicheradressen und sequentieller Abarbeitung eines Programmes vertraut ist.
Heft 7/85 des Peeker enthält eine wesentlich breiter angelegte Einführung zur Arbeitsweise des 6502-Prozessors, auf der wir hier teilweise aufbauen.

1. Vom 6502 zum 658xx

Um es gleich vorwegzunehmen: Die Prozessoren der Reihe 65SC8xx sind die Erfüllung der Träume eines 6502-Programmierers – sie stellen in (fast) jeder Hinsicht eine konsequente Erweiterung dar.
Beginnen wir mit den gemeinsamen Charakteristika der 65er-Familie, bevor wir die speziellen Eigenschaften der einzelnen Mitglieder erörtern:

● *Speicherorientierung:* Alle Prozessoren der Reihe 65xx verfügen über einen sehr kleinen Satz an Registern. Mehr oder weniger als „Ausgleich" wird ein spezieller Speicherbereich (die „Seite 0") benutzt, dessen Speicherstellen die Funktion von Indexregistern übernehmen können. Der Streit der Experten, welches Konzept das bessere sei – viele Register oder ein spezieller Speicherbereich, über den zusätzliche Adressierungsarten möglich sind – dauert noch an. Das eine Extrem dürfte der TMS 9916 darstellen, der praktisch ohne eigene Register auskommt und nur mit Speicherstellen arbeitet, das andere die 32xx-Reihe von National Semiconductor, bei der der Programmierer mit Registern und dazugehörigen Adressierungsarten förmlich erschlagen wird.
● *Vergleichsweise niedrige Taktfrequenz:* Alle Prozessoren der 65er-Familie verwenden einen zweiphasigen Takt, bei dem in jeder der beiden Phasen Operationen ausgeführt werden – im Gegensatz zu den meisten anderen Prozessoren, die den Systemtakt intern erst ein paarmal herunterteilen. So kommt es, daß der Befehlsdurchsatz eines 65816 bei 4 MHz *über* dem eines mit 8 MHz betriebenen 68000 liegt. Ein Register-zu-Register-Transfer benötigt beim 68000 4 Taktzyklen, beim 65816 sind es nur 2; Schiebeoperationen des Akkus werden beim 65816 in 2 Taktzyklen ausgeführt, der 68000 braucht 6 Zyklen dazu. Daß der 68000 dennoch ein ganzes Stück leistungsfähiger als der 65816 ist, liegt an der Unzahl der verfügbaren Befehle und der internen 16/32-Bit-Architektur.
Ein Vergleich zwischen dem 65816 und dem Z80 ist beinahe unfair (obwohl der Z80 auch über eine Reihe von 16-Bit-Befehlen verfügt) – hier müßte man mit einer Taktfrequenz von ca. 15 MHz arbeiten, um auf dieselbe Rechenleistung zu kommen.
● *Kleiner Befehlssatz:* Der 6502 kennt lediglich 57 verschiedene Befehle mit 13 verschiedenen Adressierungsarten. Seine Assemblerprogrammierung ist deshalb auch für Benutzer relativ leicht zu erlernen, die nicht ihre gesamte Zeit vor einem Bildschirm verbringen wollen. Für den 65816 ist diese Behauptung nicht mehr ganz zutreffend: Die Zahl der Befehle ist auf 91 gewachsen, die der Adressierungsarten auf 24.
● *Starke Asymmetrie:* Nur ein sehr kleiner Teil der Adressierungsarten und Befehle ist uneingeschränkt anwendbar – beispielsweise ist ein Befehl wie *LDX $4000,Y* („Lade X-Register mit...") definiert, *STX $4000,Y* („Speichere X-Register auf...") dagegen nicht. Die nicht definierten Ausnahmen lassen sich aufgrund des kleinen Befehlssatzes relativ einfach im Kopf behalten – in dem Maße, in dem der Befehlssatz bei den neueren Mitgliedern der 65er-Familie gewachsen ist, wurde auch die Asymmetrie zumindest teilweise beseitigt.
Der Hauptgrund für dieses Verhalten liegt in der ausschließlichen Verwendung von Befehlscodes mit *einem* Byte Länge. Um eine vollständige Symmetrie (Stichwort: Orthogonalität) zu erreichen, hätte man Mehrbyte-Befehlscodes verwenden müssen.
Waren beim 6502 noch rund die Hälfte aller möglichen Befehlscodes „reserviert", so ist beim 65816 lediglich ein einziger Code „für Erweiterungen" freigehalten. Wie zu hören war, ist ein künftiger „651632" bereits über die Anfänge der Planung hinaus – er wird vollständig zum 65816 kompatibel sein; sämtliche neuen Befehle dieses „651632" werden mit diesem freigehaltenen Code beginnen, also zwei Byte umfassen.

1.1. Der 6502

Das „Ur-Exemplar" der 65er-Familie kam ungefähr 1977 auf den Mark und wird im Apple II bzw. II+ sowie dem Acorn („BBC-Computer") verwendet. Der 6502 ist für 1 und 2 MHz Taktfrequenz erhältlich, verfügt über einen Datenbus mit 8 Bit Breite (d.h. liest/schreibt mit jeder Operation ein Byte) und einen Adreßraum von 64K. Vier Schlagworte kennzeichnen diesen Prozessor:
● *Reine 8-Bit-Maschine:* Sämtliche Register außer dem Programmzähler („PC") sind 8 Bit breit. Es gibt weder Operationen, die mehr als 8 Bits auf einmal behandeln, noch eine Möglichkeit, mehrere Register zu einem 16-Bit-Wort zu kombinieren (wie z.B. beim Z80).
● *Akkumulatororientiert:* Der 6502 hat nur ein einziges Register, in dem Rechenoperationen

(Addition, Subtraktion, Schiebeaktionen usw.) ausgeführt werden können.

● *Ein-Adreß-Maschine:* Alle Datentransfers müssen über ein Register führen. Um z.B. eine Speicherstelle auf den Wert 0 zu setzen, muß zuerst ein Register mit diesem Wert geladen werden, danach wird der Inhalt des Registers auf die entsprechende Adresse geschrieben (vgl. 65C02). Die einzige Ausnahme stellen Rotations- und Schiebebefehle dar, die auch ohne den Umweg über ein Register auf Speicherstellen angewendet werden können.

● *Fixierter Stack:* Der „Stapelspeicher" des 6502 belegt grundsätzlich den Adreßbereich von $0100 bis $01FF und kann damit maximal 256 Bytes aufnehmen.

1.2. Der 65C02

Der Nachfolger des 6502 ist seit 1984 erhältlich und im Apple IIc sowie den neueren Ausgaben des IIe eingebaut. Seine Taktfrequenz beträgt bis 4 MHz, in den Apple-Geräten wird er mit 1 MHz betrieben. Er unterscheidet sich vom 6502 in mehrfacher Hinsicht:

● Durch einen anderen Herstellungsprozeß (das „C" im Namen steht für „CMOS") wird nur noch ein Bruchteil der elektrischen Leistung benötigt. Der Prozessor erwärmt sich auch bei mehrstündigem Betrieb nur unwesentlich – im Gegensatz zu seinem Vorläufer, der in Insiderkreisen als „Eierkocher" bespöttelt wird.

● Das Konzept der Akkumulatororientierung wurde gelockert. Durch eine Reihe neuer Befehle (PHX, PLX, PHY, PLY..) sind Datentransfers zwischen den Indexregistern und dem Stack möglich. Beim 6502 ist diese Operation nur über den Akku möglich, d.h. über den Umweg „Indexregister zum Akku – Akku auf den Stack".

● Man kann den 65C02 nur noch bedingt als Ein-Adreß-Maschine bezeichnen. Durch neue Befehle (STZ = STore Zero, TSB/TRB = Test, Set/ Reset Bits) sind erweiterte Operationen mit dem Inhalt von Speicherstellen möglich, ohne daß dabei ein Register benötigt wird. Eine echte Zwei-Adreß-Operation wie „lade Speicherstelle X mit dem Inhalt von Speicherstelle Y" ist allerdings nicht vorhanden.

Der 65C02 ist „aufwärtskompatibel" zum 6502, d.h. sämtliche Programme, die für den 6502 geschrieben wurden, laufen unverändert auch mit einem 65C02 (bis auf ein paar sehr exotische Ausnahmen). Der 65C02 „versteht" sämtliche Befehle des 6502 und führt sie in gleicher Weise aus – nur kennt er noch ein paar zusätzliche Befehle, die beim 6502 nicht definiert sind. Weiterhin ist der 65C02 „pinkompatibel": Man kann einen 6502 aus seiner Fassung herausnehmen, einen 65C02 einstecken und danach mit dem neuen Prozessor weiterarbeiten, als wäre nichts geschehen. (Für die Elektronik-Spezialisten unter den Lesern: Wegen leicht veränderter Bus-Zugriffszeiten stimmt das nicht immer – speziell bei sehr alten Ausgaben des II+ (vor Rev. 7B) sollte man dabei auf (seltene) Abstürze gefaßt sein [1]).

Abgesehen von den neuen Befehlen und der moderneren Herstellungsverfahren ist der 65C02 gegenüber dem 6502 unverändert geblieben. Es handelt sich nach wie vor um einen reinen 8-Bit-Prozessor mit 64K Adreßraum, der Stack ist auch hier auf den Bereich $0100..$01FF fixiert.

1.3. Der 65SC816

Dieser Prozessor wird im folgenden meist nur als „65816" bezeichnet und ist die 8/16-Bit-Version des 65C02. Er ist seit Mitte 1986 auf dem Markt und wird im Apple IIgs sowie im Nachfolgemodell des Acorn von BBC verwendet. Seine derzeitige Taktfrequenz beträgt bis 4 MHz, Erhöhungen auf 6 und 8 MHz sind geplant. Im Apple IIgs wird er mit 2,8 MHz betrieben. Der 65816 ist aufwärtskompatibel zum 65C02 und läßt sich per Programm zwischen zwei Betriebsarten (8 und 16 Bits) umschalten. Er verfügt nach wie vor über einen Datenbus mit 8 Bits – der Adreßbus umfaßt aber 24 Bits (Adreßraum: 16M), die internen Register sind 16 Bit breit. Hauptsächlich wegen des erweiterten Adreßbusses ist der 65816 *nicht* pinkompatibel zum 65(C)02.

Im 8-Bit-Modus („Emulation Mode") verhält er sich wie ein 65C02, d.h. sämtliche Register sind 8 Bit breit, der Stack ist auf den Bereich $0100..$01FF fixiert, der verfügbare Adreßraum beträgt 64K (genauere Informationen s. 4.1.2).

Im 16-Bit-Modus („Native Mode") gibt es eine ganze Reihe von Erweiterungen:

● Der Akkumulator („A") und die Indexregister („X", „Y") sind unabhängig voneinander zwischen 8 und 16 Bit Breite umschaltbar.

● Dem Programmzähler („PC") wird ein Programmbank-Register („PB") mit acht Bit Breite vorangestellt, der Prozessor gibt damit jeweils eine 24-Bit-Adresse zum Lesen von Befehlen aus.

● Der Adressierung von Daten wird ein Datenbank-Register („DB") mit acht Bit Breite vorangestellt, der Prozessor gibt damit jeweils eine 24-Bit-Adresse zum Lesen und Schreiben von Daten aus. Das Datenbank-Register ist unabhängig vom Programmbank-Register.

● Sämtliche Adressierungen der Seite 0 („Zeropage") werden über ein Direct-Register („D") ausgeführt. Dieses Register hat eine Breite von 16 Bits und kann beliebig verändert werden. Damit sind bis zu 256 verschiedene „Zeropages" möglich. Das D-Register arbeitet unabhängig von der Programm- und Datenbank.

● Der Stackpointer („SP") hat eine Breite von 16 Bits, der Stack ist somit nicht mehr auf den Bereich $0100..$01FF fixiert und kann maximal 64K aufnehmen;

● Der Prozessor verfügt über zahlreiche neue Befehle zur Manipulation der Register PB, DB und D sowie über eine Reihe neuer Adressierungsarten, die ihn fast ideal für die Implementation von Hochsprachen machen (s. Abschnitt 6).

1.4. Der 65SC802

Die „8-Bit-Version" des GTE-65SC816 ist voraussichtlich ab Ende dieses Jahres erhältlich (bei anderen Produzenten bereits früher). Dieser Prozessor ist softwarekompatibel zum 65SC816 und pinkompatibel zum 65(C)02. Er enthält denselben Registersatz wie der 65SC816, kennt dieselben Befehle und läßt sich genauso zwischen „Emulation Mode" und „Native Mode" umschalten. Der einzige Unterschied besteht darin, daß die Inhalte der Register PB und DB keine (elektronische) Wirkung haben – die entsprechenden Adreßleitungen sind nicht aus dem Chip herausgeführt. Obwohl der Prozessor intern mit 24-Bit-Adressen arbeitet, beträgt der effektive Adreßraum damit 64K.

Ein Prototyp des 65SC802, den mir Herr Earwaker von der Firma GTE (dem Hersteller der 65SC8xx-Reihe) freundlicherweise überlassen hat, befindet sich seit nunmehr zwei Monaten in einem IIe – im „Emulation Mode" laufen selbst extrem zeitkritische Programme wie SUPER-QUICK völlig störungsfrei.

Wer Interesse an Experimenten mit den neuen Befehlen der 65SC8xx-Prozessoren hat und sich nicht gleich einen Apple IIgs zulegen will, sollte sich den 65SC802 und dazu einen der folgenden Assembler besorgen:

ProDOS ORCA/M, Version 4.0 von Byte Works – sehr komplex, arbeitet mit einzelnen Modulen, lokalen Labels und einer Makrobibliothek, ist entsprechend langsam und für sehr große Projekte die einzige Wahl;

Merlin Pro von Roger Wagner – für Projekte mittlerer Größe, recht schnell, unterscheidet sich in Bedienung und Verhalten kaum von den bisher erschienenen Merlin-Versionen;

SC-Assembler (ProDOS-Version) von S-C Software Corporation – zum Experimentieren das ideale Programm; vollständig RAM-resident und deshalb extrem schnell, der erzeugte Objektcode kann *vom Editor aus* aufgerufen werden. Für mittlere und größere Programmprojekte allerdings nur bedingt geeignet.

2. Aufteilung des Adreßraumes

Bild 1 zeigt die Aufteilung des Speichers aus der Sicht des 65816-Programmierers, wobei eine „Erblast" deutlich zu erkennen sein sollte: Bedingt durch die Forderung nach Software-Kompatibilität zum 65(C)02 besteht der adressierbare Speicher nicht aus einer durchgehenden („linearen") Folge von Adressen, er ist statt dessen in 256 Bänke mit jeweils 64K aufgeteilt. Jede Bank besteht ihrerseits aus 256 Seiten („Pages") mit jeweils 256 Bytes.

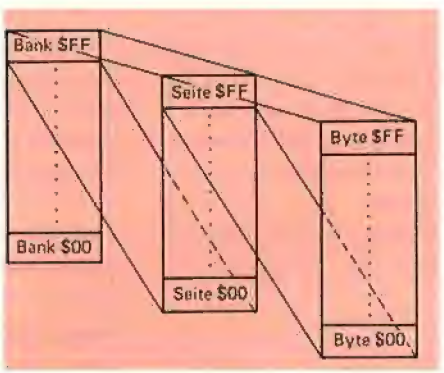


Bild 1: Aufteilung des Adreßraumes

Ohne allzusehr auf die folgenden Abschnitte vorgreifen zu wollen: Eine lineare Adressierung würde grundsätzlich drei Bytes als Operanden benötigen – um z.B. den Akku mit dem Inhalt der Speicherstelle $17A412 zu laden, wäre ein

Befehl erforderlich, der vier Byte Programmspeicherplatz belegt:

```
LDA $17A412 ; 3-Byte-Adresse
```

Diese Adressierungsart („absolut lang") kennt der 65816 *auch* – im Normalfall werden aber nur zwei anstelle von drei Bytes für eine Adreßangabe benötigt. Beispiel: Wenn das Register DB den Inhalt $17 hat, dann führt der Befehl

```
LDA $A412     ; 2-Byte-Adresse
```

zum selben Ergebnis, wobei zusätzlich noch einige Taktzyklen eingespart werden. Das Programm ist also nicht nur kürzer, sondern auch schneller.
Für Maschinenprogrammierer dürfte diese Unterteilung des Speichers in den seltensten Fällen ein Problem darstellen – für Hochsprachen-Compiler sieht das schon etwas anders aus. Man darf gespannt sein, ob die demnächst für den Apple IIgs erscheinenden Programmiersprachen Pascal und C die Größe eines Programms auf 64K begrenzen (wie es z.B. beim Turbo-Pascal für den IBM PC der Fall ist).
Wir werden uns im weiteren für „lange" Adressen an die Schreibweise halten, die von Apple, Inc. in den Handbüchern zum IIgs benutzt wird. Die Bank-Adresse (ein Byte) ist durch einen Schrägstrich von der Adresse *innerhalb* der Bank (zwei Byte) getrennt. Die folgenden Notationsformen bezeichnen also dieselbe Adresse:

```
$17/A412      ; Bank $17, Adr. $A412
$17A412       ; Adresse $17A412
```

2.1. 16-Bit-Operanden

Da der 65816 nur über einen Datenbus mit 8 Bit Breite verfügt, führt das Laden eines 16 Bit breiten Registers zu zwei aufeinanderfolgenden Speicherzugriffen. Ein Befehl wie „LDA $A412" liest also die Speicherstellen $17/A412 und $17/A413 hintereinander, wenn der Akku auf 16 Bit Breite gesetzt ist.
Der Zeitbedarf einer solchen Operation liegt erheblich über dem eines „echten" 16-Bit-Prozessors, der beide Bytes gleichzeitig liest bzw. schreibt. Abgesehen von den niedrigeren Kosten im Vergleich zu einem vollwertigen 16-Bit-System gibt es noch einen kleinen Vorteil anzumerken: Probleme mit dem „alignment" von Adressen wie beim 8086 oder dem 68000 gibt es nicht. Bei diesen Prozessoren ergibt sich ein unterschiedlicher Zeitbedarf bzw. ein Fehler, wenn der niederwertige Teil eines 16- bzw. 32-Bit-Operanden nicht auf einer Speicherstelle mit geradzahliger Adresse liegt.
(Eine Kuriosität am Rande: Das Ansprechen mehrwertig dekodierter „Toggles" in der Elektronik des Computers ist im 16-Bit-Modus praktisch unmöglich. Das beste Beispiel dafür ist der Lautsprecher des Apple, dessen Polarität bei einem Zugriff auf den Adreßbereich $C030..$C03F umgeschaltet wird. Ein Befehl wie „LDA $C030" adressiert im 16-Bit-Modus zuerst die Speicherstelle $C030, danach die Speicherstelle $C031 – der Lautsprecher wird also im Verlauf von zwei Mikrosekunden zweimal umgeschaltet und gibt folglicherweise überhaupt keinen Ton von sich.)

2.2. Reihenfolge der Speicherung

Alle Daten, die mehr als ein Byte Speicherplatz belegen, werden vom Prozessor grundsätzlich so gespeichert, daß das niedrigstwertige Byte auf der niedrigsten Speicheradresse steht. Beispiele:

```
xxxx- AD BB AA      LDA $AABB
```

Dieser Befehl lädt den Akku mit dem Inhalt der Speicherstelle (DB)/$AABB. Der Opcode des Befehls ($AD) steht auf der Adresse xxxx, der niederwertige Teil der Adresse ($BB) auf der Adresse xxxx+1, der höherwertige ($AA) auf der Adresse xxxx+2. Analoges gilt für 16-Bit-Operationen wie z.B. das Laden des Akkumulators mit einem festen Wert („immediate"):

```
xxxx- A9 2Ø 17      LDA #$172Ø
```

Hier wird der Akku mit dem Wert $1720 geladen, wobei der niederwertige Teil ($20) auf der Adresse xxxx+1, der höherwertige Teil ($17) auf der Adresse xxxx+2 steht. Diese Art der Speicherung (Low-Byte first) findet sich bereits beim 65(C)02. Neu hinzugekommen ist beim 65816 die Behandlung von „langen" Adressen:

```
xxxx- AF BB AA 2Ø   LDA $2ØAABB
```

Dieser Befehl lädt den Akku mit dem Inhalt der Speicherstelle $20/AABB, die Adresse ist ebenfalls im „umgekehrter" Reihenfolge gespeichert.
Das Konzept der umgekehrten Speicherung von Adressen wird für *alle* Speicheroperationen konsequent benutzt – es gilt für Variablen, Festwerte, Pointer und „lange" Adressen sowie für die Reihenfolge der Speicherung auf dem Stack. (Eine Returnadresse wird so gespeichert, daß ihr niederwertiger Teil auf der niedrigeren Adresse zu stehen kommt.)

3. Die Register des 65816

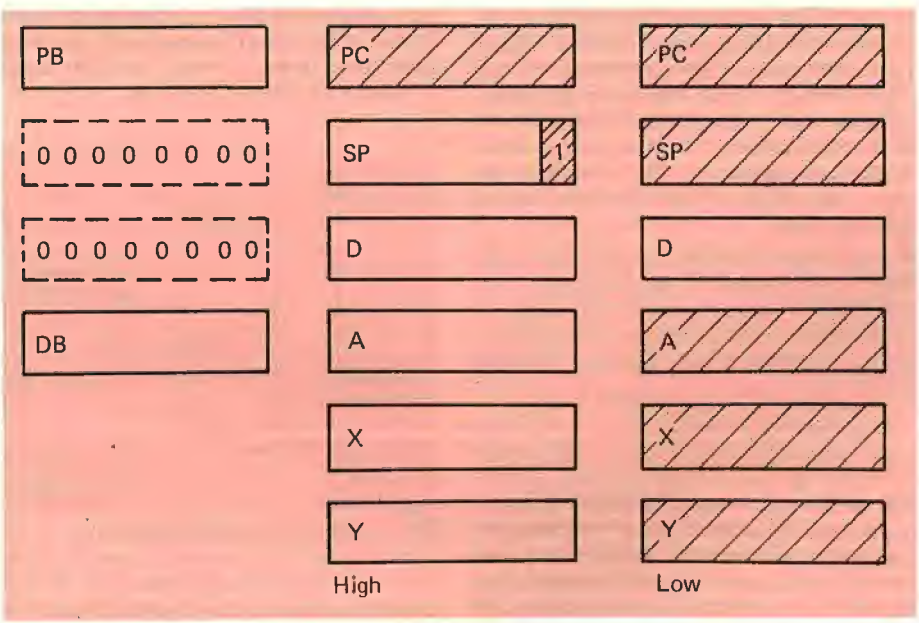
Bild 2 zeigt den vollständigen Registersatz des 65816. Bereits beim 65(C)02 vorhandene Register bzw. Flaggenbits sind schraffiert dargestellt. Alles, was hier und im folgenden über den 65816 gesagt wird, gilt auch für den 65802 – Ausnahmen sind jeweils explizit als solche gekennzeichnet.

3.1. Programmbank-Register (PB) und Programmzähler (PC)

Der 65816 bildet die (24-Bit-)Adresse des nächsten zu lesenden Befehls aus dem momentanen Stand des Programmzählers (untere 16 Bits) und des Programmbank-Registers (obere 8 Bits).
Beispiel: Ist PB = $20 und PC = $140A, wird der nächste Befehl von der Adresse $20/140A gelesen. Der PC wird für jedes gelesene Byte um eins erhöht, ein „Überlauf" in PB findet nicht statt, wenn der PC über $FFFF hinaus erhöht wird. Der folgende Befehl wird *nie* ausgeführt, wenn er sich auf der gezeigten Adresse befindet:

```
$2Ø/FFFE- AD BB AA  LDA $AABB
```

Hier liest der Prozessor den Opcode für „LDA absolut" von der Speicherstelle $20/FFFE, den ersten Teil der Adresse ($BB) von der Speicherstelle $20/FFFF und das nächste Byte von $20/0000.
Sprünge („JMP") und Aufrufe von Unterprogrammen („JSR") finden grundsätzlich innerhalb der Bank statt, auf die das Programmbank-Register zeigt. Das Überschreiten einer Programmbank durch relative Sprünge ist ebenfalls *nicht* möglich.
Es gibt nur zwei Möglichkeiten, den Inhalt von PB zu verändern, wenn man einmal von Interrupts (s. Abschnitt 5) absieht:



*) Die Inhalte der Register PB und DB bleiben beim 65802 ohne Wirkung auf die effektiv erzeugte Adresse.

Bild 2: Register des 65816

● Den Befehl JML („JMP long"). Der Operand (d.h. das Sprungziel) hat hier eine Länge von drei Bytes. Mit dem Inhalt des höchstwertigen Bytes wird PB neu gesetzt, der alte Inhalt von PB geht dabei verloren. Beispiel: Wenn PB = $20 und PC = $1A32, dann haben diese beiden Register nach Ausführung des Befehls

```
$2Ø/1A32-    JML $4Ø8516
```

den folgenden Inhalt: PB = $40, PC = $8516. Der nächste Befehl wird also von der Adresse $40/8516 gelesen.

● Den Befehl JSL („JSR long"). Der Operand hat ebenfalls eine Länge von drei Bytes. Vor der Ausführung werden der alte Inhalt des PC *und* der von PB auf den Stack gebracht (insgesamt 3 Bytes), danach werden PB und PC wie bei JML mit den Werten des Operanden gesetzt. Beispiel: Wenn PB = $20 und PC = $1A32, dann haben diese beiden Register nach Ausführung des Befehls

```
$2Ø/1A32-    JSL $4Ø8516
```

den folgenden Inhalt: PB = $40, PC = $8516. Der nächste Befehl wird also von der Adresse $40/8516 gelesen. Auf dem Stack befinden sich drei Bytes als Rücksprungadresse, nämlich die Werte $35, $1A, $20. Der Befehl RTL („Return long") beendet das Unterprogramm, indem er diese drei Bytes wieder vom Stack holt und in PB/PC einsetzt. (Für Spezialisten: Der altbekannte „Fehler" des 6502, nämlich die Speicherung einer um eins zu niedrigen Returnadresse, ist auch für „lange" Unterprogrammaufrufe erhalten geblieben.)
Mit dem Befehl *PHK* kann der momentane Inhalt des Programmbank-Registers auf den Stack gebracht werden – ein „Herunterholen" ist dagegen nur über ein RTL möglich.

3.2. Datenbank-Register (DB)

Der 65816 bildet eine (24-Bit-)Datenadresse, indem er einer gegebenen 16-Bit-Adresse den Inhalt des Datenbank-Registers voranstellt. Beispiel: Wenn DB den Inhalt $12 hat, führt der Befehl

```
LDA  $8Ø4Ø
```

zum Laden des Akkumulators mit dem Inhalt der Speicherstelle $12/8040. (Wenn der Akku auf 16 Bit Breite gesetzt ist, wird zusätzlich die Speicherstelle $12/8041 gelesen.)
Das Datenbank-Register ist vom Stand des Programmbank-Registers unabhängig – selbstmodifizierender Code (d.h. das Herumstochern im eigenen Programm) ist also nur dann möglich, wenn beide Register denselben Wert haben.
Die Erzeugung von Datenadressen unterscheidet sich in einem wesentlichen Detail von der Erzeugung von Programmadressen: Die Überschreitung einer momentan gesetzten Datenbank ist jederzeit möglich. Das folgende Beispiel kommt leider ohne einen Vorgriff auf die „indizierte Adressierung" nicht aus (s. 6.1.6): Wenn DB den Wert $20 hat und das X-Register den Wert $4412 enthält, dann liest der Befehl

```
LDA  $2ØØØ,X
```

den Inhalt der Speicherstelle $20/6412 ($2000 plus $4412 = $6412). Unter den gleichen Bedingungen adressiert der Befehl

```
LDA  $FØØØ,X
```

die Speicherstelle $21/3412 ($F000 plus $4412 = $13412) – der Überlauf wird also berücksichtigt. Der Inhalt von DB wird durch diese Operation nicht verändert.
Dieses Verhalten gilt für alle „indizierten" Adressierungen, ein *„Unterlauf"* (d.h. die Adressierung einer Bank, die unterhalb des momentanen Standes von DB liegt), ist dagegen nicht möglich.

Der 65816 kennt zwei Möglichkeiten, bei der Adressierung von Daten den Inhalt von DB zu ignorieren:
● „Absolut lang", d.h. durch die Angabe einer vollständigen Adresse (3 Bytes). Beispiel:

```
LDA  $22A4B2
```

Dieser Befehl lädt den Akku mit dem Inhalt der Speicherstelle $22/A4B2. Der momentane Inhalt von DB wird ignoriert, DB wird durch diese Operation nicht verändert.
● „Seite 0", d.h. durch die Angabe einer Adresse, die nur aus einem Byte besteht. Hier geschieht die Adressierung über das Direct-Register und bezieht sich immer auf Bank 00 (s. nächster Abschnitt).
Unter Umständen ergeben sich dadurch Programmierfehler, die äußerst schwierig zu finden sind. Beispiel:

```
LDA $Ø1ØØ       ; adressiert DB/Ø1ØØ
LDA $ØØFF       ; adressiert ØØ/ØØFF
```

Das Datenbank-Register kann mit dem Befehl PHB auf den Stack gebracht und mit PLB wieder heruntergeholt werden. Einen direkten Befehl wie „Lade DB mit <Wert>/<Registerinhalt>" gibt es nicht.

3.3. Direct-Register (D)

Dieses Register hat eine Breite von 16 Bits und bestimmt die Startadresse der „Direct Page" (Seite 0 oder „Zeropage") innerhalb der Bank 00. Der 65(C)02 kennt nur eine Nullseite, nämlich den Speicherbereich von $0000 bis $00FF, der 65816 kann dagegen bis zu 256 verschiedene „Nullseiten" benutzen. Das Setzen des D-Registers auf den Wert $0000 definiert den Speicherbereich $00/0000..$00/00FF als Seite 0 und entspricht damit dem Verhalten des 65(C)02. Wenn das D-Register auf den Wert $0100 gesetzt wird, belegt die Seite 0 den Speicherbereich $00/0100..$00/01FF usw.

Ein als „Direct Page" definierter Speicherbereich hat die folgenden Eigenschaften und Grenzen:
● Für die Adressierung einer Speicherstelle wird nur ein einziges Byte benötigt, die restliche Adresse fügt der Prozessor automatisch hinzu. Beispiel: Wenn D den Wert $0000 hat, dann adressiert der Befehl

```
LDA $8F
```

die Speicherstelle $00/008F. Wenn D den Wert $0900 hat, dann adressiert derselbe Befehl die Speicherstelle $00/098F.
● Da das D-Register eine Breite von 16 Bits hat, ist der Programmierer nicht an Seitengrenzen gebunden. Wenn D den Wert $2344 hat, dann adressiert der Befehl „LDA $8F" die Speicherstelle $00/23D3. Um die Sache nicht unnötig zu komplizieren, werden wir im weiteren davon ausgehen, daß die Seite 0 immer auf einer Seitengrenze (d.h. auf $xx00) beginnt. (Falls Sie ein Liebhaber von völlig undurchschaubarem Code sein sollten: Was ergibt die Befehlsfolge LDX #$2340 / TSC / TCD / STX $01 / RTS ?)
● Speicherstellen der Seite 0 können als „Pointer" benutzt werden, d.h. die Adresse einer Speicherstelle enthalten (s. 6.1.8). Beispiel: Wenn D den Wert $0300 hat, die Speicherstelle $00/0320 den Wert $16 und die Speicherstelle $00/0321 den Wert $80, dann „zeigen" diese beiden Speicherstellen auf die Adresse $8016. Der Befehl

```
LDA  ($2Ø),Y
```

lädt den Inhalt der Speicherstelle DB/8016 in den Akku, wenn Y den Wert $00 hat. Für Y = $01 wird DB/8017 adressiert, für Y = $02 DB/8018 usw. Die Abschnitte 6.1.8, 6.2.1 und 6.2.3 beschäftigen sich ausführlicher mit dieser Adressierungsart und ihren Erweiterungen.
● Beim 65(C)02 kann mit einem „Nullseiten-Befehl" nicht über die Nullseite hinaus adressiert werden – beim 65816 dagegen schon. Beispiel: Wenn D den Wert $0000 hat und das X-Register den Wert $40 (8-Bit-Modus) enthält, dann adressiert der Befehl

```
LDA $EØ,X
```

auf einem 65(C)02 (bzw. auf einem 65816 im „Emulation Mode") die Speicherstelle $(00/)0020 ($E0 plus $40 = $020, der Überlauf wird ignoriert). Im „Native Mode" wird dagegen die Speicherstelle $00/0120 adressiert, d.h. der Überlauf wird berücksichtigt. Das gilt auch für Indexregister mit 16 Bit Breite: Wenn D = $0000 und X = $2000, dann adressiert der oben gezeigte Befehl die Speicherstelle $00/$20E0.
● Adressierungen der Seite 0 finden grundsätzlich in der Bank 00 statt – diese Bank läßt sich durch indizierte Adressierung nicht überschreiten (vgl. 3.2). Wenn D = $9000 und X = $4000, dann adressiert der Befehl

```
LDA $84,X
```

nicht etwa die Speicherstelle $01/3084, sondern $00/3084 – *dieser* Überlauf wird nicht berücksichtigt.

Programmiertricks wie z.B. die sich selbst modifizierende Routine CHRGET in Applesoft dürften der Vergangenheit angehören. Es wäre zwar möglich, ein Unterprogramm zu schreiben, das innerhalb der momentan gesetzten Seite 0 abläuft – es müßte aber mit einem JSL aufgerufen werden und mit einem RTL enden (sehr unökonomisch).
Durch die Tatsache, daß sowohl die Seite 0 als auch der Stack innerhalb der Bank 00 frei be-

Festplattenlösung als Bausatz für jedermann

Seit einiger Zeit sieht man in der Fachpresse einschlägige Anzeigen: Megacore, Megaboard und MDB mit unterschiedlichen Speicherkapazitäten. Das Kernstück all dieser zur Verfügung stehenden Möglichkeiten ist immer MEGA-BOARD, der Harddisk-Controller mit Software. Peeker hat für Sie beim Hersteller, der Firma Frank & Britting GmbH, preisgünstige Sonderkonditionen für die Produktreihe „Mobile Datenbox" ausgehandelt. Die Resonanz auf die Angebote war gut. Es hat sich gezeigt, daß viele Apple-Anwender von den neuen Möglichkeiten, die diese externe Speichererweiterung bietet, begeistert sind, aber den finanziellen Aufwand scheuen.

Peeker hat sich deshalb wieder mit Frank & Britting in Verbindung gesetzt. Durch das Entgegenkommen des Herstellers ist es nun möglich, auch einen preiswerten Harddisk-Bausatz zu erwerben. Er enthält den Controller, den Kabelsatz, die Software und eine 10-Megabyte-Festplatte (ohne Netzteil). Diesen Bausatz (Bausatz 1) können Sie wie die MDB beim Hüthig-Software-Service inkl. DB-Meister für DM 1482,– beziehen.

Wenn Sie über andere Platten-Kapazitäten verfügen (bis 64MB) oder selbst eine Festplatte kaufen möchten, können Sie den Controller mit Software und Kabelsatz auch getrennt erwerben. Harddisks gibt es gebraucht recht günstig auf dem Markt. Der Fachhandel bietet standardisierte Netzteile und Gehäuse.
Der Controller mit dem speziellen Flachbandkabel und der Software kann direkt beim Entwickler und Hersteller Frank & Britting für DM 898,– bezogen werden (Bausatz 2, ohne DB-Meister). Die Firma Frank & Britting entwickelt seit Jahren Controller für Harddisk-Subsysteme, unter anderem auch MEGA-BOARD für Apple II, II+, IIe und den IIgs. Alle Produkte werden von Frank & Britting selbst hergestellt und durchlaufen bis zur Auslieferung mehrere Funktionstests. Die Qualitätsanforderungen an die Produkte sind sehr hoch: Ein Megaboard z.B. wird, nachdem es von der Produktionsabteilung kommt, erst nach einstündigem Dauertest auf speziellen Rechnersystemen freigegeben. Danach durchläuft es noch Tests am Apple, wo möglichst alle ungünstigen Systemvoraussetzungen simuliert werden. Somit wird gewährleistet, daß nur Produkte ausgeliefert werden, die 100%ig in Ordnung sind.

Aufbau eines kompletten Harddisk-Subsystems:

1. Controller
2. Software
3. Spezielles Flachbandkabel (40adrig, 1,5 Meter lang)
4. Power-Kabel mit genormten 4-Pol-HD-Anschlußstecker
5. Ausführliches Handbuch
6. Harddisk mit einer ST506- oder ST412-Schnittstelle (Standard-Schnittstelle von Festplatten im Kapazitätsbereich von 5 MB bis 80 MB, mit 34-Pol- und 20-Pol-Stekkerleiste)
7. Netzteil (5 Volt/mindestens 2 Ampere, 12 Volt/mindestens 3 Ampere)

Bild 1: Lieferumfang Frank & Britting

Der Lieferumfang von MEGA-BOARD umfaßt die Produkte unter Punkt 1 bis 5 (s. Bild 1). Die Verbindung der einzelnen Komponenten ist sehr einfach (s. Bild 2). Der Controller wird in einen freien Slot gesteckt und mit dem 40-Pol-Stecker des Flachbandkabels verbunden. An der anderen Seite des Kabels befinden sich ein 34-Pol- und ein 20-Pol-Stecker. Beide werden einfach auf die ST506-Schnittstelle der Harddisk gesteckt. Dann wird die vorgefertigte und entsprechend bezeichnete 5-Volt-Leitung des Flachbandkabels am Netzteil angeschlossen oder an der Festplatte angelötet (für die externe Spannungsversorgung des Controllers). Das vorgefertigte Power-Kabel für die Festplatte wird mit dem verpolgeschützten 4-Pol-Spannungsnormstecker auf die Harddisk gesteckt (s. Bild 3). Entsprechend der Steckeraufschrift werden dann die Kabel mit dem Netzteil verbunden.

Bild 2: Gesamte Soft- und Hardware für den Selbstbau

Das Netzteil sollte einem guten Qualitätsstandard entsprechen. Harddisk-Laufwer-

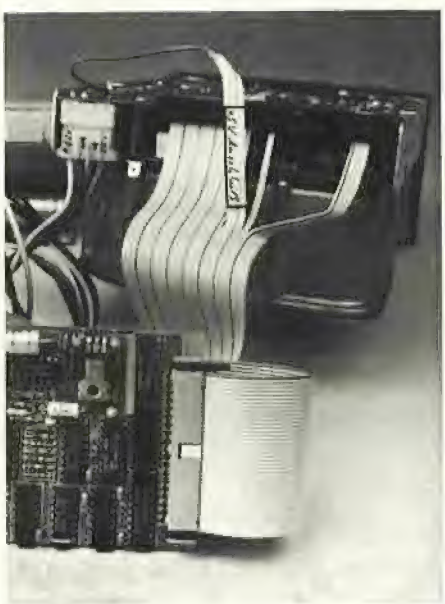
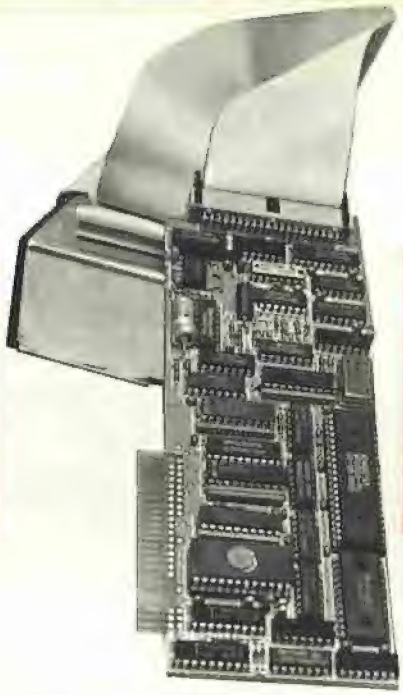


Bild 3: Die Steckpositionen im Detail

er-Leser ·



Sonderangebot für Peeker-Leser

nur gültig bis 31.3.87!

ke reagieren empfindlich auf Störungen und Unregelmäßigkeiten der Spannungsversorgung. Die Harddisk braucht zwei Spannungsversorgungen mit +5 Volt und +12 Volt, die Spannungen haben gemeinsame Masse. Beide Masseleitungen sollten getrennt bis zum Netzteil geführt werden. Die Ströme sind recht unterschiedlich, präzise Werte sind ggf. den Datenblättern des Laufwerkes zu entnehmen. Als Standardwerte gelten bei +5 Volt (inkl. 0,9 Ampere für den Controller, der vom externen Netzteil versorgt wird) etwa 1,8 Ampere, bei +12 Volt etwa 1,5 Ampere. Allerdings ist bei der Dimensionierung des Netzteils zu beachten, daß beim „Hochlaufen" der Harddisk der 12-Volt-Teil für 10 Sekunden mit etwa 2 Ampere belastet wird. Deshalb empfiehlt es sich, eine Leistungsreserve zu kalkulieren und mit 5V/2A bzw. 12V/3A zu rechnen.

Bild 4: Mit wenigen Handgriffen zusammengestecktes, betriebsfertiges System

In verschiedenen Fachzeitschriften, Hobby- und Computerläden werden preiswerte und leistungsfähige Netzteile angeboten. Bei Apple-kompatiblen Rechnern im IBM-Gehäuse beispielsweise ist in den meisten Fällen das Netzteil für die zusätzliche Last einer Harddisk ausreichend dimensioniert. Gehäuse für die Harddisk sind ohne Schwierigkeiten zu bekommen. Man sollte hierbei auf die Möglichkeit ausreichender Belüftung achten, entweder durch Schlitze bzw. Löcher im Gehäuse oder durch einen eingebauten Lüfter. Eine Harddisk kann in der Regel bei Betriebstemperaturen zwischen 5 und 55 Grad Celsius arbeiten. Die obere Grenze der zulässigen Betriebstemperatur wird im geschlossenen Gehäuse ohne entsprechende Belüftungsmaßnahmen, gerade bei warmer Umgebungsluft wie im Sommer, leicht erreicht. Die Festplatte kann irgendwo am Arbeitsplatz untergebracht werden, denn das Flachbandkabel darf bis 5 Meter(!) lang sein.

Harddisk-Parameter

Die Festplatten unterscheiden sich untereinander durch die Anzahl der Köpfe (HEAD), Anzahl der Zylinder (CYL), durch verschiedene Zugriffsmodi (FAST SEEK) und verschiedene Schreibströme (WRITE PRECOMP, WP). Frank & Britting hat bewußt vermieden, diese Parameter über Hardware-Brücken, Mäuseklaviere etc. einzustellen. Sämtliche laufwerkspezifischen Parameter können per Software eingestellt werden. In der Tabelle finden sich einige Beispiele für auf dem Markt befindliche Festplattenlaufwerke.

Bei dem Initialisieren der Platte werden diese Werte vom Initialisierungsprogramm erfragt. Danach wird formatiert, und das Harddisk- Subsystem ist betriebsbereit.

In manchen Datenblättern ist der Wert „FAST SEEK" nicht zu finden. In diesem Fall gilt folgende Regel: Wenn der Wert der durchschnittlichen Zugriffszeit (Datenblatt) kleiner als 100ms ist, wird FAST SEEK mit „Y" beantwortet.

Beispiele Harddisk-Parameter

Laufwerk formatiert	MByte	HEAD	CYL	WP ab CYL	FS
Rodime RO 352	10.00	4	306	256	Y
Rodime RO 204E	41.94	8	640	0	Y
Lapine Titan 3532	20.05	4	612	0	Y
Seagate ST 412	10.00	4	306	0	Y
Segeate ST 506	5.00	4	153	128	N
BASF 6188	11.80	4	360	0	Y
Microscience HH725	10.00	4	612	0	Y
CDC 9415-528	22.30	4	697	0	Y
Micropolis 1323	33.60	4	1024	0	Y
Miniscribe 3425	21.40	4	615	0	Y
Tandon TM 252	10.00	4	306	0	Y

Wie wird bestellt?

Sie senden Ihre Bestellung an den Hüthig-Software-Service. Sie erhalten dann von der Firma Frank & Britting eine Vorausrechnung, nach deren Überweisung Ihnen von dort Bausatz 1 oder Bausatz 2 geliefert wird. Wenn Sie Bausatz 1 bestellen, erhalten Sie gleichzeitig vom Hüthig-Software-Service das Programm DB-Meister (2 Disketten und ein Handbuch) in der für die MDB bereits angepaßten Version. Nach einer geringfügigen Änderung im Hello-Programm können Sie diese Neuversion des DB-Meisters übrigens auch zusätzlich auf 35-Spur-Laufwerken einsetzen.

Zur Bestellung können Sie eine der im Peeker eingehefteten Bestellkarten verwenden. Stichwort:

Bausatz 1: Megaboard-Controller, Kabelsatz, Software, 10MB-Festplatte, DB-Meister.....Sonderangebot DM 1482,–

Bausatz 2: Megaboard-Controller, Flachbandkabel, Software.....Sonderangebot DM 898,–

Hüthig PUBLIKATION

Hüthig Software Service

Im Weiher 10 · 6900 Heidelberg 1

weglich sind, kann eine neue (katastrophale) Art von Programmierfehler auftreten, die es beim 65(C)02 aufgrund der fixierten Speicherbereiche überhaupt nicht gibt – nämlich den Überlauf des Stacks in die momentan benutzte Seite 0 hinein.

Das D-Register kann sowohl über den Befehl PHD auf den Stack gebracht als auch mit PLD wieder zurückgeladen werden. Einen direkten Ladebefehl wie („Lade D mit <Wert>") gibt es nicht, dafür aber die Befehle TCD (Akku-Inhalt nach D) und TDC (D-Inhalt zum Akku). Der Buchstabe „C" steht hier für die Tatsache, daß der Akku bei diesen beiden Befehlen immer als 16-Bit-Register behandelt wird – unabhängig davon, auf welche Breite er momentan gesetzt ist.

3.4. Stackpointer (SP)

Wie bereits erwähnt, benutzt der 65(C)02 den fixen Speicherbereich von $0100 bis $01FF als Stack. Der Stackpointer hat eine Breite von 8 Bits und zeigt (im Gegensatz zu den meisten anderen Prozessoren) nicht auf die zuletzt belegte Speicherstelle, sondern auf den jeweils nächsten freien Speicherplatz.
Der Stack „wächst" in Richtung absteigender Adressen – das Speichern von *n* Bytes auf dem Stack setzt SP um *n* herab. **Bild 3** gibt ein Beispiel für den 65(C)02 (Registerbreite: 8 Bits), in dem der Akku-Inhalt ($20) zuerst auf den Stack gebracht und danach wieder heruntergeholt wird.

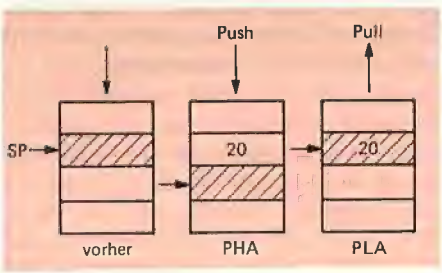


Bild 3: Stackpointer beim 6502

Der einzige Unterschied zum 65(C)02 besteht darin, daß der Stackpointer des 65816 eine Breite von 16 Bits hat und der Stack damit nicht mehr auf eine Speicherseite (d.h 256 Bytes) begrenzt ist – er kann mit einer beliebigen Adresse innerhalb der Bank 00 beginnen.

Stack und Stackpointer haben beim 65816 die folgenden Eigenschaften und Grenzen:

- Als Speicherbereich wird *grundsätzlich* die Bank 00 benutzt – unabhängig von den Inhalten der Register PB, DB und D.
- Der Stackpointer bewegt sich bei Über- und Unterlauf im Kreis. Eine Erniedrigung über $0000 hinaus (zuviele „Push"-Operationen) setzt ihn auf den Wert $FFFF und erniedrigt ihn von da aus weiter, eine Erhöhung über $FFFF hinaus (zuviele „Pull"-Operationen) setzt ihn auf den Wert $0000 und erhöht ihn von da aus weiter.
- Adressierungen des Stacks können nie über die Bank 00 hinausgehen, selbst dann nicht, wenn mit Indizierung gearbeitet wird (s. 6.2.5). Beispiel: Wenn SP = $FFF0, dann adressiert der Befehl

```
LDA $40,S     ; SP plus "Offset" $40
```

nicht die Speicherstelle $01/0030, sondern $00/0030.
Es existieren weder direkte Befehle, mit denen man den momentanen Wert des Stackpointers auf den Stack bringen bzw. ihn herunterholen kann, noch Befehle wie „Lade SP mit <Wert>". (Falls Ihnen der Gedanke daran merkwürdig vorkommen sollte: Mit dem Z80 geht so etwas ohne Schwierigkeiten). Für den Transfer zwischen einem Register und dem Stackpointer gibt es dagegen gleich vier Möglichkeiten: TSC und TCS (Stackpointer → 16-Bit-Akku und umgekehrt) sowie TSX und TXS (Stackpointer → X-Register und umgekehrt). Der letzte dieser vier Befehle hat einen ganz speziellen „Haken" (s. Abschnitt 4.2).

3.5. Akku und Indexregister (A, X und Y)

Der Akkumulator (A) ist das zentrale (und einzige) Rechenregister des Prozessors – sämtliche logischen (AND, ORA, EOR) und arithmetischen Operationen (Addieren, Subtrahieren) lassen sich nur über ihn ausführen.
Rotations- und Schiebebefehle (ROL, ROR, ASL, LSR) sind dagegen sowohl über den Akku als auch direkt auf Speicherstellen anwendbar.
Für Operationen mit diesem Register stehen die meisten Adressierungsarten zur Verfügung (23 von 24 Adressierungsarten).
Der Akku des 65(C)02 ist grundsätzlich 8 Bit breit, der des 65816 läßt sich (getrennt von den Indexregistern) zwischen 8 und 16 Bits umschalten (s. 3.6.3).
Wenn der Akku des 65816 auf 8 Bit Breite gesetzt ist, bleiben die höherwertigen 8 Bits in den Tiefen des Prozessors verborgen. Diesen Platz kann man unter dem Registernamen B als Zwischenspeicher benutzen, wobei nur eine einzige Operation definiert ist:

```
XBA       ; Tausch AL – AH
```

Hier handelt es sich um einen echten Tausch: Der vorherige Inhalt des 8-Bit-Akkus befindet sich danach im Register „B", der des „B"-Registers im Akku. Die Flags N und Z werden entsprechend dem neuen Akku-Inhalt gesetzt.
Da das „B"-Register (im Gegensatz zu den höherwertigen Teilen der Indexregister X und Y) durch Umschalten der Breite des Akkus nicht gelöscht wird, ist dieser Befehl eine wesentlich schnellere Alternative zur Speicherung auf den Stack. Der Befehl XBA funktioniert auch dann, wenn der Akku auf 16 Bit Breite gesetzt ist – in diesem Fall wird das höherwertige Byte mit dem niederwertigen ausgetauscht. Das folgende Programmfragment könnte eine vierstellige Hexzahl ausgeben:

```
LDA #$2476      ; Akku <- $2476
XBA             ; Akku: $7624
JSR xxx         ; Ausgabe von $24
XBA             ; Akku: $2476
JSR xxx         ; Ausgabe von $76
```

Die konventionelle Alternative dazu sieht wesentlich umständlicher aus:

```
LDA #$2476
PHA             ; Zwischenspeichern
LSR A           ; höherwertigen Teil
LSR A           ; nach links verschieben
...             ; 8mal: Akku <- $0076
JSR xxx         ; Ausgabe von $24
PLA             ; Original zurück
AND #$00FF      ; bleibt $0076
JSR xxx         ; Ausgabe von $76
```

Die Indexregister des Prozessors (X und Y) sind nicht für Rechenoperationen vorgesehen, sondern zur indizierten Adressierung von Daten. Bevor Sie über diese Erklärung den Kopf schütteln, folgt ein Beispiel: Der Befehl

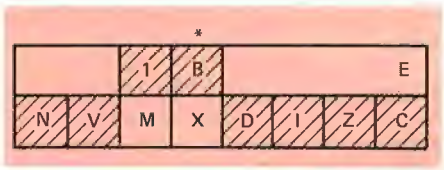
```
LDA $2000,X
```

lädt den Akku mit dem Inhalt der Speicherstelle $2000, wenn das X-Register den Wert 0 hat, für X = 1 wird $2001 adressiert, für X = 2 $2002 usw. (für den 65816: DB/2000, DB/2001, DB/2002...).
Folgerichtig sind arithmetische Operationen mit den Indexregistern nur sehr beschränkt möglich. Man kann ihren Inhalt um eins erhöhen (INX, INY), um eins erniedrigen (DEX, DEY) sowie Vergleiche ausführen (CPX, CPY = „Compare X" bzw. „Compare Y"). Additionen oder Bitmanipulationen mit den Inhalten der Indexregister sind nicht definiert.
Die Indexregister des 65(C)02 haben grundsätzlich eine Breite von 8 Bits – mit dem zuvor gezeigten Beispiel lassen sich also Speicherstellen im Bereich von $2000 bis $20FF (d.h. eine „Speicherseite") adressieren.
Die Indexregister des 65816 lassen sich getrennt vom Akkumulator zwischen 8 und 16 Bit umschalten (s. 3.6.3). Der Befehl „LDA $2000,X" hat im letzteren Fall eine „Reichweite" von DB/2000 bis DB+1/1FFF. (Um ganz korrekt zu sein: Wenn der Akku ebenfalls auf 16 Bits gesetzt ist und X den Inhalt $FFFF hat, werden die Speicherstellen DB+1/$1FFF und DB+1/$2000 angesprochen).
Der 65(C)02 kennt bereits zahlreiche Adressierungsarten und Transfermöglichkeiten für die beiden Indexregister, die beim 65816 noch erweitert worden sind. Die erfreulichsten Neuerungen bestehen aus einer direkten Transfermöglichkeit zwischen X und Y über die Befehle TXY und TYX sowie den (bereits auf dem 65C02 implementierten) direkten Stackoperationen (PHX, PHY, PLX, PLY).

3.6. Das Prozessorstatusregister (P)

Dieses Register besteht aus einer Reihe einzelner Bits („Flags"), die voneinander getrennt behandelt werden, seine „Breite" (8 Bits für sämtliche Prozessoren der 65er-Familie) ist nur im Zusammenhang mit Stackoperationen interessant.
Die Bits M und X sind beim 65(C)02 nicht definiert und haben immer den Wert „1", sind also immer gesetzt. Dasselbe ist der Fall, wenn sich der 65816 im „Emulation Mode" befindet. **Bild 4** gibt die Reihenfolge und Bedeutung der einzelnen Bits noch einmal wieder.

Die einzelnen Bits des P-Registers werden durch die Ergebnisse von Rechenoperationen gesetzt. Der Prozessor kennt eine Reihe von Sprungbefehlen, die abhängig vom Stand der



*) 65(C)02 sowie im „Emulation Mode" des 658xx: Wird als „0" auf den Stack geschrieben, wenn durch IRQ unterbrochen wird (s. Abschnitt 5), hat ansonsten immer den Wert „1".

Bild 4: Status-Register

einzelnen Bits ausgeführt oder ignoriert werden („bedingte Sprünge"). Die nächsten Abschnitte enthalten zahlreiche Beispiele dazu.
Achtung: Im Gegensatz zu den meisten anderen Prozessoren werden die Bits Z, V und N auch durch Register-Ladeoperationen gesetzt! Vorteil: Die z.B. vom Z80 her gewohnte Befehlsfolge „OR A,00" kann man sich in den meisten Fällen sparen. Nachteil: Ein Ablauf wie der folgende funktioniert meistens nicht:

```
<Prüfung>           ; setzt Flags
<Ladeoperation>     ; mit Wert A
<bedingter Sprung>  ; zu <weiter>
<Ladeoperation>     ; mit Wert B
<weiter> ...
```

Auf einem Z80 oder einem 8088 nimmt dieses Programmfragment eine Prüfung vor und lädt ein Register abhängig vom Ergebnis der Prüfung entweder mit dem Wert A oder dem Wert B.
Auf einem 65(C)02 oder einem 65816 werden die Flags *durch die Ladeoperation mit dem Wert A* neu gesetzt, der darauffolgende Sprung basiert also nicht mehr auf dem Ergebnis der Prüfung, sondern auf dem soeben geladenen Wert A. Um das gewünschte Ergebnis zu erreichen, muß die Routine folgendermaßen formuliert werden:

```
<Ladeoperation>     ; mit Wert A
<Prüfung>           ; setzt Flags
<bedingter Sprung>  ; zu <weiter>
<Ladeoperation>     ; mit Wert B
<weiter>...
```

Der 65(C)02 kennt nur eine einzige Adressierungsart des P-Registers, nämlich die Befehle PHP („Push P-Register auf den Stack") und PLP („Pull P-Register"), wobei jeweils ein Byte auf dem Stack belegt bzw. wieder freigegeben wird. Beim 65816 sind die Befehle SEP („Set P-Bits") und REP („Reset P-Bits") hinzugekommen, mit denen einzelne Bits direkt gesetzt bzw. gelöscht werden können.

3.6.1. Das N-Flag („Negativ-Flag")

Es wird gesetzt, wenn das Ergebnis einer Operation im Zweierkomplement einen negativen Wert darstellen würde. („Zweierkomplement": Die Werte $00 bis $7F bzw. $0000 bis $7FFF werden als positive Zahlen von dezimal 0 bis 127 bzw. 0 bis 32767 definiert. Werte von $80 bis $FF bzw. $8000 bis $FFFF werden als negative Zahlen interpretiert, die Bereiche gehen von dezimal -128 ($80) bis -1 ($FF) bzw. von -32768 ($8000) bis -1 ($FFFF). Eine negative Zahl wird dadurch gekennzeichnet, daß das höchstwertige Bit eines Operanden gesetzt ist ($7F binär: 0111 1111, $80 binär: $1000 0000)).
Das N-Flag wird durch vier Arten von Operationen gesetzt:

- Ladevorgänge: „LDA $A000" setzt N, wenn der Inhalt von $A000 größer $80 bzw. der Inhalt von $A000/$A001 größer $8000 ist, ansonsten wird es gelöscht;
- Rechen- und Schiebeoperationen setzen N, wenn die folgende Bedingung zutrifft, ansonsten wird das Flag gelöscht:
  – Addition: Ergebnis größer $7F bzw. $7FFF;
  – Subtraktion: Ergebnis kleiner 0 (d.h. $FF..$80 bzw. $FFFF..$8000);
  – Linksschieben oder -rotieren: Ergebnis hat höchstes Bit gesetzt;
  – Rechtsrotieren: Ergebnis hat höchstes Bit gesetzt;
  – Vergleich: Vergleichsoperand ist größer.
- Erhöhen und Erniedrigen von Registern/Speicherstellen: Wenn das Ergebnis im Bereich $80..$FF bzw. $8000 bis $FFFF liegt, wird N gesetzt, ansonsten gelöscht.
- Den Befehl BIT: Dieser Befehl kopiert das höchstwertige Bit des adressierten Operanden auf das N-Flag, ohne den Inhalt des Akkus zu verändern. Wenn der Operand größer als $7F bzw. $7FFF ist, wird das N-Flag gesetzt, ansonsten wird es gelöscht.

Betrachten wir beispielsweise eine Befehlsfolge wie:

```
LDX $AØØØ       ; X <- Inhalt von $AØØØ
DEX
```

Der Befehl DEX erniedrigt den Inhalt von X um eins. Wenn wir davon ausgehen, daß X auf acht Bit Breite gesetzt ist, ergeben sich die folgenden Möglichkeiten:
– wenn $A000 einen Wert im Bereich $81..$FF enthält, bleibt das N-Flag gesetzt, weil X auch nach der Operation einen Wert größer/gleich $80 enthält;
– wenn $A000 den Wert $80 enthält, dann hat X nun den Inhalt $7F: Das N-Flag wird durch die zweite Operation gelöscht;
– wenn $A000 einen Wert zwischen $01 und $7F enthält, bleibt das N-Flag unverändert gelöscht: X enthält einen Wert zwischen $00 und $7E;
– wenn $A000 den Wert $00 enthält, wird das N-Flag durch die zweite Operation gesetzt: X enthält nun den Wert $FF.
Mit dem Befehl SEP #$80 kann das N-Flag explizit gesetzt, mit REP #$80 explizit gelöscht werden. Der Befehl BMI („Branch on Minus") führt zu einem Sprung, wenn das N-Flag gesetzt ist; wenn es gelöscht ist, wird er ignoriert. Der Befehl BPL („Branch on Plus") verhält sich umgekehrt – er wird nur dann ausgeführt, wenn das N-Flag gelöscht ist.

3.6.2. Das V-Flag („oVerflow" = Überlauf)

Dieses Flag dient der Erkennung eines Zweierkomplement-Überlaufs. Es wird nur durch zwei Arten von Operationen beeinflußt:

- Durch Addition und Subtraktion: Wenn sich während einer derartigen Operation das Vorzeichen (also das N-Bit) ändert, dann wird das V-Flag gesetzt – aber nur dann, wenn der dazuaddierte bzw. subtrahierte Wert kleiner als $80 bzw. $8000 ist. In allen anderen Fällen wird es gelöscht, also wenn
  – ein Wert addiert oder subtrahiert wird, der einen Vorzeichenwechsel erzwingt (größer/gleich $80 bzw. größer/gleich $8000) oder
  – sich während der Operation der Stand des N-Flags nicht ändert.
- Durch den Befehl BIT: Dieser Befehl kopiert das zweithöchste Bit des adressierten Operanden auf das V-Flag.

Ein einmal gesetzter Zustand des V-Flags ist außerordentlich langlebig – er wird weder durch Vergleiche, Schiebeoperationen noch durch Erhöhungen/Erniedrigungen von Registern/Speicherstellen verändert.
Mit dem Befehl CLV (ersatzweise REP #$40) kann das V-Flag explizit gelöscht, mit SEV (oder SEP #$40) kann es explizit gesetzt werden. Der Sprungbefehl BVS wird nur dann ausgeführt, wenn das V-Flag gesetzt ist, der Sprungbefehl BVC wird nur bei gelöschtem V-Flag ausgeführt. Aufgrund seiner „Langlebigkeit" wird dieses Flag recht gerne zweckentfremdet, um z.B. eine bestimmte Bedingung zeitweilig zwischenzuspeichern.

3.6.3. Die Flags M und X („Memory" und „indeX")

Diese beiden Bits haben beim 65(C)02 sowie im „Emulation Mode" des 65816 grundsätzlich den Wert „1", sind also gesetzt. Im „Native Mode" des 65816 bestimmen sie die Breite des Akkumulators (M = 1: 8 Bit, M = 0: 16 Bit) sowie die der Indexregister (X = 1: X- und Y-Register 8 Bit, X = 0: X- und Y-Register 16 Bit). Verzweigungen, die vom Stand dieser beiden Bits abhängig sind, kennt der 65816 nicht – es existiert lediglich die Möglichkeit, die Bits mit den Befehlen REP und SEP zu löschen bzw. zu setzen:

- SEP #$20 setzt M (und damit den Akku auf 8 Bit Breite), REP #$20 löscht dieses Flag. Der Inhalt des Akkus wird durch diese Befehle nicht beeinflußt – nach einem SEP #$20 wird der höherwertige Teil lediglich „versteckt";
- SEP #$10 setzt X (und damit die Indexregister auf 8 Bits), REP #$10 löscht das Flag X und setzt 16 Bit breite Indexregister. Der Befehl SEP #$10 führt eine automatische Löschung des höherwertigen Teils der Indexregister aus (s.a. 4.2);
- SEP #$30 ist identisch mit der Befehlsfolge SEP #$20/SEP #$10 und setzt A, X und Y auf 8 Bits; REP #$30 entspricht REP #$20/REP #$10 und setzt A, X und Y auf 16 Bits.

Die Breite der jeweiligen Register ist bei Operationen mit Speicherstellen ausschlaggebend dafür, ob eine oder zwei Adressen angesprochen werden. Beispiele:

```
SEP #$3Ø        ; A, X und Y: 8 Bit
LDA $AØØØ       ; lädt $AØØØ
CMP $BØØØ       ; vergleicht mit $BØØØ
STX $1FFF       ; X-Inhalt -> $1FFF
```

Dieselbe Befehlsfolge, mit REP #$30 eingeleitet, würde A mit dem Inhalt der Speicherstellen $A000/$A001 laden, den Akku-Inhalt mit den Speicherstellen $B000/$B001 vergleichen und den Inhalt von X in $1FFF/2000 ablegen.

Mit den (teilweise recht delikaten) Problemen, die sich bei Datentransfers zwischen Registern unterschiedlicher Breite ergeben, beschäftigt sich der gesamte Abschnitt 4, in dem auch die Umschaltung zwischen „Emulation Mode" und „Native Mode" behandelt wird.

3.6.4. Das D-Bit („Decimal")

Über dieses Bit kann der Prozessor zwischen „normaler" Rechenweise und („packed") BCD-Arithmetik umgeschaltet werden. Im ersten Fall rechnet der Akku mit Werten im Bereich von 0 bis 255 (bzw. -128 bis +127), im zweiten Fall werden jeweils vier Bits zur Darstellung einer Dezimalzahl benutzt, der Rechenbereich erstreckt sich damit von 00 bis 99. Für den Betrieb mit 16 Bits ergeben sich die Bereiche 0..65535 (bzw. -32768..+32767) sowie 0000 bis 9999 (geht beim 68000 nicht).
So gut wie alle Prozessoren verfügen über eine derartige Umschaltmöglichkeit, die allerdings in den seltensten Fällen genutzt wird (Turbo BCD von Borland ist eines der wenigen kommerziellen Programme, die dem D-Bit – hier dem Äquivalent des 8086/88 – zu seiner Existenzberechtigung verhelfen).
Der einzige Unterschied zwischen dem 65(C)02 und dem 65816 besteht darin, daß das D-Bit bei letzterem durch ein Reset automatisch gelöscht wird (s. Abschnitt 5). Als Apple-Entwickler hat man aber offensichtlich solche Ängste vor gesetzten D-Flags, daß auch die Reset-Routine im Monitor des IIgs noch mit einem CLD beginnt.
Der Befehl CLD (ersatzweise REP #$08) löscht das D-Flag und setzt den „normalen" Modus, der Befehl SED (bzw. SEP #$08) setzt das D-Flag und schaltet auf BCD-Arithmetik um. Sprungbefehle, die sich auf den Zustand des D-Flags beziehen, gibt es nicht.

3.6.5. Das I-Flag (Interruptsperre)

Wird zum Sperren und Entsperren von IRQ-Anforderungen (Interrupt request; hardwaremäßiger, sperrbarer Interrupt) verwendet und hat bei sämtlichen Mitgliedern der 65er-Familie dieselbe Funktion. SEI (ersatzweise SEP #$04) *sperrt* IRQ-Anforderungen, CLI (oder REP #$04) läßt Interrupts zu. Ein durch den Befehl BRK ausgelöster „Software-"Interrupt wird durch den Zustand des I-Flags nicht beeinflußt. Näheres zur Funktion in Abschnitt 5.

3.6.6. Das Z-Flag („Zero" = Null)

Dieses Flag wird gesetzt, wenn eine Operation den Wert $00 ergibt, ansonsten wird es zurückgesetzt. Unterschiede zwischen 65(C)02 und 65816 gibt es nicht, wenn man von der Möglichkeit des direkten Setzens über die Befehle REP und SEP absieht (s.u.). Ähnlich dem N-Flag wird das Z-Flag durch eine Vielzahl von Operationen verändert:

- Ladevorgänge: Wenn ein Register mit dem Wert $00 (bzw. $0000) beladen wird, setzt diese Operation das Z-Flag. Ist der Wert ungleich $00, wird Z gelöscht;
- Addition und Subtraktion: Wenn das Ergebnis den Wert $00 (oder $0000) hat, wird das Z-Flag gesetzt. Achtung: Das ist auch bei einem Über- oder Unterlauf der Fall: Im 8-Bit-Modus ergibt die Befehlsfolge LDA #$90 / CLC / ADC #$70 das Ergebnis $100, d.h. der Akku enthält danach den Wert $00.
- Schieben und Rotation: Wenn der verbleibende Operand den Wert $00 (oder $0000) hat, wird das Z-Flag gesetzt, ansonsten wird es gelöscht. Beispiel: Das Rechtsschieben einer Speicherstelle mit dem Inhalt $01 setzt diese Speicherstelle auf den Wert $00 und setzt damit das Z-Flag. Analoges gilt für das Linksschieben einer Speicherstelle oder des Akkus mit dem Wert $80 (im 8-Bit-Modus).
- Erhöhung und Erniedrigung: Z wird abhängig davon gesetzt, ob das Ergebnis der Operation den Wert $00 hat oder nicht. Zur Abwechslung ein Beispiel im 16-Bit-Modus:

```
LDX #$0001    ; <> $00: löscht Z
DEX           ; X=0000: setzt Z
DEX           ; X=$FFFF: löscht Z
```

- Vergleiche: der Prozessor führt jeden Vergleich „intern" als Subtraktion aus – nur daß der numerische Wert des Ergebnisses in keinem Register festgehalten wird. Ein Vergleich setzt damit das Z-Flag, wenn beide Operanden denselben Wert haben (die Subtraktion ergibt den Wert $00).
- Den Befehl BIT: abgesehen von einer direkten Kopie der beiden höchstwertigen Bits des adressierten Operanden in die Flags N und V führt dieser Befehl ein bitweises AND zusammen mit dem Inhalt des Akkus aus und setzt mit dem Resultat das Z-Flag. Folge: Wenn der Akku den Inhalt $00 (oder $0000) hat, ergibt ein BIT-Befehl immer ein gesetztes Z-Flag. Hat der Akku dagegen den Inhalt $FFFF (alle Bits gesetzt), dann wird Z nur gesetzt, wenn der adressierte Operand den Wert $00 hat.

Abgesehen von den Befehlen SEP #$02 und REP #$02 kennt der 65816 keine direkten Befehle zur Beeinflussung des Z-Flags. Die zu diesem Flag gehörenden bedingten Sprünge dürften wohl neben den Befehlen LDA und STA zu den am häufigsten benutzten gehören: BEQ („Branch on EQual") führt einen Sprung aus, wenn Z gesetzt ist und wird ansonsten ignoriert; BNE („Branch on Not Equal") wird nur bei gelöschtem Z-Flag ausgeführt.

3.6.7. Das C-Flag („Carry" = Übertrag)

Das C-Flag wird zur Erkennung von Über- bzw. Unterlauf bei vorzeichenloser Arithmetik (vgl. V-Flag) sowie für Vergleiche und Schiebeoperationen verwendet. (Vorzeichenlose Arithmetik: Interpretation der Werte $00..$FF als positive Zahlen im Bereich von 0 bis 255, vgl. 3.6.1).
Das Carry funktioniert quasi als neuntes (bzw. siebzehntes) Bit des Akkumulators, sein Verhalten ist bei allen Mitgliedern der 65er-Familie dasselbe (bis auf eine zusätzliche Funktion beim 65816, die in Abschnitt 4 beschrieben ist). Die folgenden Arten von Operationen beeinflussen den Zustand des Carry *nicht*:

- Laden, Speichern und Transfer von Registerinhalten, wobei das Zurückholen des P-Registers vom Stack natürlich eine Ausnahme darstellt;
- Erhöhung oder Erniedrigung von Speicherstellen und Registern;
- Bitweise logische Operationen (AND, ORA, EOR).

Additionen und Subtraktionen benutzen das Carry als Erweiterung des Akkumulators. So ergibt z.B. die bereits in 3.6.6. gezeigte Befehlsfolge LDA #$90 / CLC / ADC #$70 das Ergebnis $100 – der Akku enthält danach den Wert $00, das Carry ist gesetzt. Eine Addition, deren Ergebnis die Breite des Akkus übersteigt, muß das Carry als Zwischenspeicher benutzen. Als Beispiel die Addition zweier 32-Bit-Zahlen:

```
LDA $A000     ; $A000/$A001
CLC
ADC $B000     ; $B000/$B001
STA $C000     ; -> $C000/$C001
LDA $A002     ; $A002/$A003
ADC $B002     ; $B002/$B003
STA $C002     ; -> $C002/$C003
BCS FEHLER    ; größer $FFFFFFFF!
```

Vor der ersten Addition muß das Carry gelöscht werden (der 65816 kennt keinen Befehl wie „ADD"). Ergibt sich bei dieser Operation ein Überlauf, dann wird das Carry gesetzt: Zur nächsten Addition wird automatisch der Wert 1 hinzuaddiert. (Eine Befehlsfolge wie LDA #$10 / SEC / ADC #$20 ergibt also den Wert $31 – und nicht $30.) Wenn sich durch die mit gesetztem Carry ausgeführte Addition nicht ein weiterer Überlauf ergibt, ist das Carry am Ende der gesamten Operation gelöscht.

Analoges gilt für Subtraktionen – nur daß das Carry in diesem Fall als „borrow" (= „geborgt") verwendet wird. Subtraktionen werden ebenfalls grundsätzlich unter Berücksichtigung des Carry ausgeführt, ein Befehl wie „SUB" ist nicht implementiert. Ein gesetztes Carry vor einer Subtraktion bedeutet: „nichts geborgt", bei einem gelöschten Carry wird dagegen zusätzlich der Wert 1 abgezogen. Ein Beispiel im 8-Bit-Modus:

```
LDA #$20      ; Akku <- $20
SEC           ; kein "borrow"
SBC #$21      ; Akku: $FF
LDA #$10      ; höherwertiger Teil
SBC #$08      ; ergibt $07 (nicht $08)
```

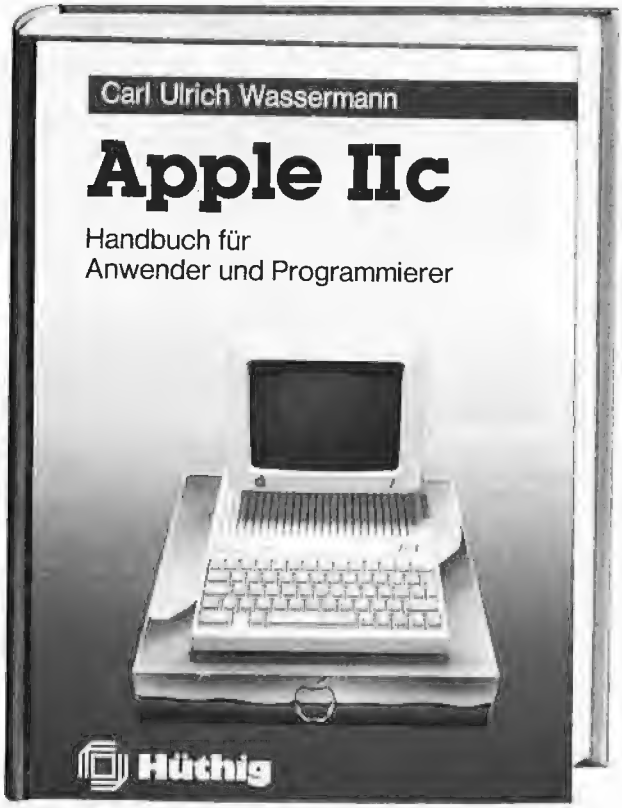
Durch den Unterlauf nach der ersten Subtraktion ist das Carry gelöscht worden, die zweite Subtraktion ergibt deshalb automatisch ein um eins erniedrigtes Ergebnis. Da in unserem Beispiel in der zweiten Subtraktion kein weiterer Unterlauf auftritt, ist das Carry am Ende der gesamten Operation wieder gesetzt.

Wie bereits des öfteren erwähnt, wird ein Vergleich zweier Operanden („Compare") über eine interne Subtraktion ausgeführt. Neben den Flags Z und N wird dadurch auch das Carry beeinflußt: Vor der Ausführung des „Compare"-Befehls wird es (implizit) gesetzt, danach wird der adressierte Operand (ebenfalls intern) von dem entsprechenden Register abgezogen. Das Carry wird dem Ergebnis entsprechend gesetzt. Beispiel:

```
LDA #$60      ; Akku <- $60
CMP $A000
```

Abhängig vom Inhalt der Speicherstelle $A000 ergeben sich drei Möglichkeiten für die Werte der Flags C, Z und N:

FACHBUCHTIP Hüthig



Apple IIc

Handbuch für Anwender und Programmierer

von Carl-Ulrich Wassermann

1985, 324 S., zahlr. Abb., kart.,
DM 35,—
ISBN 3-7785-1157-2

Wenn Sie die Leistungsfähigkeit ihres Apple IIc bisher noch nicht ausschöpfen konnten, brauchen Sie dieses Buch.

Leicht verständlich, trotzdem ausführlich wird die Sprache Applesoft BASIC dargestellt. Eine Vielzahl von Programmen, die speziell auf den Apple IIc zugeschnitten wurden, zeigen die Wirkung der einzelnen Befehle bis zu Programmiertechniken für Diskettenzugriff, Maus und andere Anwendungen. Apple IIc Maschinensprache, Programmeingabe und -kontrolle mit Monitorbefehlen in PASCAL und FORTH geben dem Leser eine Basis für die Programmierung des Apple IIc in anderen Sprachen.

Die Konfigurierung der seriellen Ports und weitere spezifische Beschreibungen des Computers ermöglichen die gezielte Kontrolle über die Maschine.

Mehr als 90 PEEK-, POKE-und Maschinenprogrammadressen, Speicheraufteilung und weitere Tabellen sind für den Programmierer eine konzentrierte, wichtige Grundlage.

BESTELLCOUPON

Bitte ausfüllen und an Hüthig Vertriebsservice, Postfach 102869 · 6900 Heidelberg schicken.

Buchtitel

Name

Straße

Ort

Unterschrift

– $A000 enthält einen Wert, der *größer* als der Inhalt des Akkus (d.h. größer als $60) ist: Die intern ausgeführte Subtraktion ergibt damit einen Unterlauf: C ist gelöscht, N ist gesetzt. Da das Ergebnis nicht $00 ist, wird Z ebenfalls gelöscht;
– $A000 enthält den Wert $60: Die Subtraktion ergibt keinen Unterlauf, C ist nach der Operation gesetzt, N ist gelöscht. Da das interne Ergebnis den Wert $00 hat, ist das Z-Flag gesetzt;
– $A000 enthält einen Wert, der *kleiner* als der Inhalt des Akkus ist: Die interne Subtraktion ergibt einen positiven Wert, C ist gesetzt, N und Z sind gelöscht.

Für Schiebe- und Rotationsbefehle wird das Carry ebenfalls als zusätzliches Bit verwendet – und zwar sowohl bei Operationen mit dem Akku als auch bei Operationen mit Speicherstellen. **Bild 5** gibt eine grafische Darstellung der Anwendung aller vier möglichen Befehle mit der Speicheradresse $A000 als Operand, wobei von einem gelöschten M-Flag ausgegangen wird. Wenn M gesetzt ist, finden alle gezeigten Operationen ausschließlich mit der Speicherstelle $A000 statt, der Inhalt von $A001 bleibt unberührt.
Ob ein Schiebebefehl auf ein oder zwei Byte angewendet wird, ist allein durch die Breite des Akkus festgelegt – die Breite der Indexregister spielt in diesem Zusammenhang keine Rolle.

Das Carry kann mit dem Befehl SEC (ersatzweise SEP #$01) explizit gesetzt und mit CLC (oder REP #$01) explizit gelöscht werden. Der bedingte Sprungbefehl BCS („Branch on Carry Set") wird nur ausgeführt, wenn das Carry gesetzt ist, und ansonsten ignoriert; der Befehl BCC („Branch on Carry Clear") bewirkt nur bei gelöschtem Carry eine Verzweigung.

4. Umschaltungen und Probleme

Dem aufmerksamen Leser dürfte aufgefallen sein, daß in den vorhergehenden Abschnitten des öfteren die Rede von der Umschaltung zwischen „Emulation Mode" und „Native Mode" war – nur das „Wie" ist nach wie vor ungeklärt. Der Grund: Das zur Umschaltung gehörige Flag „E" ist beim 65816 als neuntes Bit des P-Registers definiert und nur über einen speziellen Befehl erreichbar. Die Umschaltung von „Emulation" auf „Native" geschieht über die folgende Befehlssequenz:

```
CLC         ; löscht Carry
XCE         ; Tausch E – C, löscht E
```

Nach Ausführung dieser Befehle enthält das Carry den vorherigen Stand von E – wenn es gesetzt ist, war der Prozessor vorher im „Emulation Mode"; ist es gelöscht, hat sich der Prozessor bereits in seinem „Urzustand" befunden.
Die Umschaltung von „Native" auf „Emulation" geschieht auf ähnliche Weise:

```
SEC         ; setzt Carry
XCE         ; Tausch E – C, setzt E
```

Der Befehl XCE stellt die einzige Möglichkeit dar, an den Stand des E-Flags heranzukommen. Durch den Tausch mit dem Carry läßt sich allerdings ein Problem recht elegant lösen, das ansonsten viel Kopfzerbrechen bereiten würde – nämlich Unterprogramme, die in einem bestimmten Zustand „gefahren" werden müssen. Ein Beispiel dazu: Die I/O-Routinen des IIgs arbeiten intern grundsätzlich im „Native Mode". Damit sie sowohl von 16- als auch von 8-Bit-Programmen benutzt werden können, wird der „Aufrufzustand" des Prozessors zwischengespeichert:

```
CLC         ; Carry löschen
XCE         ; löscht E
PHP         ; P auf den Stack
REP #$3Ø    ; M=Ø, X=Ø: 16 Bit A,X,Y
JSR ...     ; Eingabe/Ausgabe
PLP         ; P vom Stack
XCE         ; Carry -> E-Flag
CLC
RTS
```

Der erste XCE-Befehl setzt den „Native Mode", der vorherige Stand von E wird dabei im Carry festgehalten. Das darauffolgende PHP bringt das P-Register (also auch das Carry-Flag) auf den Stack. Nach Ausführung der Routine wird das P-Register wieder vom Stack geholt, wobei der ursprüngliche Zustand der Flags M und X automatisch wiederhergestellt wird. Der zweite XCE-Befehl tauscht Carry und E miteinander aus und stellt so den vorherigen Zustand von E wieder her.

4.1. Umschaltmöglichkeiten

Aus dem Stand des Carry und dem Stand von E *vor* einem XCE-Befehl lassen sich insgesamt vier verschiedene Möglichkeiten („Permutationen") ableiten:

– C ist gesetzt, E ist gesetzt. Der Prozessor befindet sich vor dem Austauschbefehl im „Emulation Mode" und bleibt es danach ebenfalls. Der XCE-Befehl hat in diesem Fall nicht die geringste Wirkung.
– C ist gelöscht, E ebenfalls. Der Prozessor befindet sich vor dem Austauschbefehl im „Native Mode" und bleibt es danach ebenfalls. Der XCE-Befehl bleibt auch hier ohne jede sichtbare Wirkung.
Die anderen beiden Möglichkeiten (4.1.1. und 4.1.2.) sind leider nicht ganz so einfach:

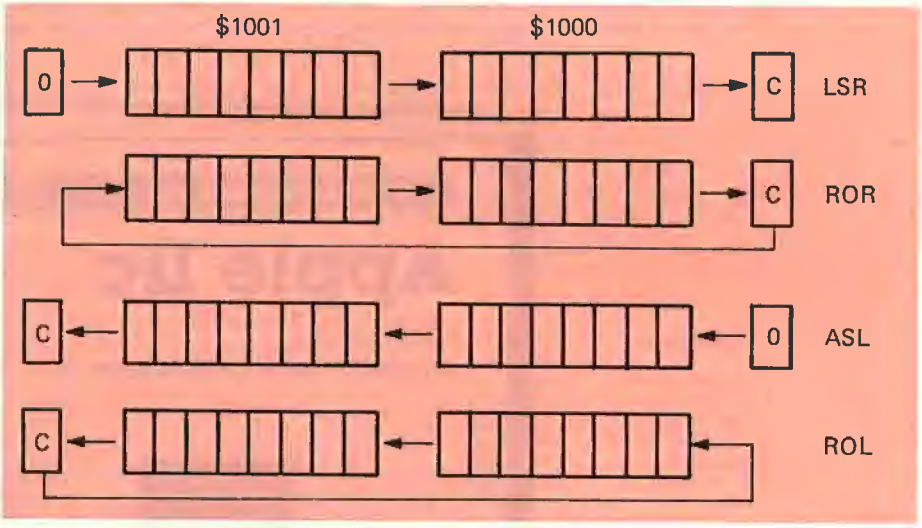


Bild 5: Schiebe- und Rotierbefehle

4.1.1. Von „Emulation Mode" auf „Native Mode"

Diese Umschaltung hat folgende Wirkungen:
- Die Flags M und X werden entsperrt. Mit einem folgenden REP-Befehl können der Akku und/oder die Indexregister auf 16 Bit Breite umgeschaltet werden. Achtung: Das Löschen des E-Flags selbst bewirkt keine Veränderung der Registerbreite!
- Das höherwertige Byte des Stackpointers wird entsperrt. SP kann mit einem folgenden Befehl auf einen anderen Bereich als $0100..$01FF gesetzt werden. Die Löschung des E-Flags selbst hat keinen Einfluß auf den Inhalt von SP.
- Der Prozessor greift von nun an für IRQ, BRK und andere Unterbrechungsanforderungen auf andere Vektoren zu (s. Abschnitt 5).

Wo bleiben die Register PB, DB und D? Der nächste Absatz hat dafür eine verblüffende Antwort parat.

4.1.2. Von „Native Mode" auf „Emulation Mode"

Diese Umschaltung hat folgende Wirkungen:

- Die Flags M und X werden auf den Wert „1" gesetzt und gesperrt, Akku und Indexregister sind dadurch auf 8 Bit Breite festgelegt. Folgende REP-Befehle für diese beiden Bits bleiben ohne Wirkung.
- Das höherwertige Byte des Stackpointers („SH") wird auf den Wert $01 gesetzt und gesperrt, der Stack somit auf den Bereich $0100..$01FF festgelegt. *Der vorherige Inhalt von SH geht verloren.*
- Der Prozessor greift von nun an für IRQ-, BRK- und andere Unterbrechungsanforderungen auf die Vektoradressen des 6502 zu (s. Abschnitt 5).

FACHBUCHTIP

Hüthig

Apple Assembler

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

Begleitdiskette DM 28,—
ISBN 3-7785-1048-7

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

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

Im zweiten Teil werden alle Adressen des Monitors zusammengestellt, die für Assembler-Programmierer von Nutzen sein können. Darüber hinaus findet der Leser Unterroutinen für hexadezimale Addition/Subtraktion/Multiplikation/Division, Binär-Hex-ASCII-Umwandlung usw.

Der dritte Teil befaßt sich mit der Speicherverwaltung der Language Card und der IIe-64K-Karte und enthält Move-Programme zum Verschieben von Daten in die und aus der Language Card sowie der 64K-Karte.

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

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

Apple-Assembler lernen

Band 1: Einführung in die Assembler-Programmierung des 6502/65C02

von Jürgen Kehrel
1985, 234 S., kart., DM 38,—
ISBN 3-7785-1151-3

Begleitdiskette: DM 44,—
ISBN 3-7785-1243-9

Das zweibändige Werk „Apple Assembler lernen" ist ein kompletter Kurs in der Assemblerprogrammierung des 6502 und des 65C02 auf dem Apple II, der nicht da aufhört, wo andere Einführungen auf „weiterführende Literatur" verweisen.

Starkes Gewicht wird auf die praktische Anwendung gelegt. Deshalb gehört zum Kurs ein vollwertiger 2-Pass Assembler, der als einer von wenigen die erweiterten Befehle der neuen IIc und IIe Prozessoren 65C02 verarbeitet und der auch lange Programme von mehr als 1 000 Zeilen in wenigen Sekunden übersetzt. Ein interaktiver Debugger und Simulator hilft Ihnen, eigene und fremde Maschinenprogramme zu verstehen. Mit seinen vielfältigen und mächtigen Möglichkeiten läßt er Sie hinter die Kulissen Ihres Rechners schauen. Sie können „sehen", was abläuft, Ihre Vorstellungskraft wird angeregt und nicht nur einfach Ihr Gedächtnis strapaziert.

In 35 überschaubaren Lektionen lernt der Leser sämtliche Maschinenbefehle des Apple und die wichtigen Grundalgorithmen.

Apple-Assembler lernen

Band 2: Nutzung besonderer Apple-Eigenschaften

von Jürgen Kehrel
1986, 281 S., kart., DM 38,—
ISBN 3-7785-1170-X

Begleitdiskette: DM 44,—
ISBN 3-7785-1244-7

Der zweite Band stellt Programme und Unterroutinen vor, um fast alle grundlegenden Probleme auf dem Apple in Maschinensprache zu lösen: Dazu gehören Ein- und Ausgabeoperationen wie z. B. formatierte Bildschirmausgabe in 40 und 80 Z/Z., Lores- und Hires-Grafik, Hires-Schrift durch Bit-Grafik, Hires-Fenster incl. Fensterscroll, Ansprache von Diskettenlaufwerken entweder unter DOS und ProDOS oder auch ganz direkt Nibble für Nibble am Beispiel eines schnellen Kopierprogramms, das in einem Durchgang formatiert und Daten schreibt.

Zugriffe auf die Stringverwaltung und die Fließkommaarithmetik von Applesoft werden ebenso behandelt wie elegante Musik- und Toneffekte. Dabei wird ausgiebig von ROM-Routinen Gebrauch gemacht, die jeweils im Zusammenhang erklärt werden. Sie lernen viel über das Innenleben Ihres Apple und wie Sie BASIC-Programme mit Maschinensprache verbessern können.

NEU

Apple-CP/M: Assembler-Programmierung

für Einsteiger, Fortgeschrittene und 6502-Kenner

von Heinrich Kersten
1986, 246 S., DM 48,—
ISBN 3-7785-1379-6

Begleitdiskette: DM 44,—
ISBN 3-7785-1380-X

Das Buch beschreibt die Assembler-Programmierung im Rahmen des CP/M-Betriebssystems. Es wurde vorrangig für die Besitzer von Apple II-Geräten konzipiert. Der überwiegende Teil des Buches ist aber geräteunabhängig und somit auch für die Benutzer anderer CP/M-tüchtiger Fabrikate (mit 8080/Z80-Prozessor) von Interesse.

Einsteiger erhalten im ersten Abschnitt einen Intensiv-Kurs (auch in der DDT-Anwendung). Für Kenner der 6502-Programmierung werden viele Querverweise vorgestellt. Der Z80-Befehlsvorrat wird im Detail erläutert. Es folgen wichtige Basisprogramme (Konvertierungen, Binär- und BCD-Arithmetik), umfassende Diskussionen der Debugger, Assembler und Linker sowie des CP/M-Betriebssystems (Speicherverwaltung, BDOS und BIOS-Aufrufe, Bildschirmfunktionen). Den Abschluß bilden lauffähige Beispielprogramme und Utilities und ein umfangreicher Tabellen-Anhang.

BESTELLCOUPON

Bitte ausfüllen und an Ihren Buchhändler oder an den Hüthig Vertriebsservice, Postfach 10 28 69 · 6900 Heidelberg schicken.

Buchtitel

Name

Straße

Ort

Unterschrift

● Die höherwertigen Bytes der Indexregister X und Y werden auf den Wert $00 gesetzt und gesperrt, ihr vorheriger Inhalt geht verloren, das höherwertige Byte des Akkus bleibt dagegen unverändert.

Die Register PB (Programmbank), DB (Datenbank) und D (Direct Page) bleiben nach wie vor aktiv.
Damit ein für den 65(C)02 geschriebenes Programm korrekt abläuft, müssen diese Register allesamt *vor der Umschaltung* auf den Wert $00 (bzw. $0000) gesetzt werden.
Um es genau zu nehmen: Dieses Setzen kann auch nach der Umschaltung erfolgen – auch im „Emulation Mode" stehen sämtliche Befehle des 65816 weiterhin zur Verfügung. Die Ausnahmen sind:

– Ein Löschen der Flags M und X ist nicht möglich, ein Befehl wie REP #$30 bleibt ohne Wirkung.
– Ein Versetzen des Stacks auf einen anderen Bereich als $0100..$01FF ist nicht möglich, der höherwertige Teil von SP bleibt immer auf dem Wert $01.
– Indizierte Adressierungen der Seite 0 sprechen immer den durch das D-Register festgelegten Bereich an, bleiben also auf eine Speicherseite beschränkt (vgl. 3.3). Dasselbe gilt für indizierte Adressierungen über den Stackpointer.

Das große Problem bei einer Umschaltung in dieser Richtung besteht im „rücksichtslosen" Setzen von SP. Ein Unterprogramm wie das folgende führt mit absoluter Sicherheit zu einem Systemabsturz, wenn der Stackpointer zum Zeitpunkt des Aufrufs auf einen anderen Wert als $01xx gesetzt ist:

```
SEC             ; -> Emulation Mode: SP
XCE             ; wird auf $Ølxx gesetzt
<...>
<...>
CLC             ; zurück auf "Native":
XCE             ; SP bleibt auf $Ølxx!
RTS             ; -> Absturz
```

Um diesen Absturz zu verhindern, muß der Stand von SP vor der Umschaltung zwischengespeichert und hinterher wieder zurückgeladen werden. Eine Umschaltung von 16 auf 8 Bits muß also zumindest die folgenden Elemente enthalten:

```
TSC             ; Akku <- Stackpointer
STA xxxx        ; Stackpointer retten
LDA #$Ø1FF
TCS             ; Stack auf $Ø1FF
SEC
XCE             ; "Emulation Mode"
PHP             ; auf $Ø1FF-Stack
<...>
<...>
PLP             ; vom $Ø1FF-Stack
XCE             ; -> "Native Mode"
STA yyyy        ; Funktionsergebnis
LDA xxxx        ; 16-Bit-Stack wieder
TCS             ; zurück in SP
RTS             ; via 16-Bit-Stack
```

Dieses Programmfragment ist in mehrfacher Hinsicht unvollständig – es zerstört den momentanen Inhalt des Akkus sowie die höherwertigen Hälften der Indexregister und läßt außerdem DB, PB und D völlig unberücksichtigt.

Der Apple IIgs kann mit fast sämtlichen Zusatzkarten betrieben werden, die auch in einem IIe funktionieren. Da die „Firmware" dieser Zusatzkarten für den 65(C)02 geschrieben ist, findet vorher eine recht aufwendige Umschaltung statt, die sämtliche der hier fehlenden Elemente enthält. Apple, Inc. hat zu diesem Zweck ein festes „Protokoll" definiert, dessen genaue Abfolge allerdings zum gegenwärtigen Zeitpunkt noch nicht endgültig feststeht. Wir werden es deshalb in einer späteren Ausgabe des Peeker beschreiben.

4.2. Register unterschiedlicher Breite

Bei einem Befehl wie „LDA <Speicheradresse>" ist die Sachlage klar: Je nachdem, ob der Akku auf 8 oder auf 16 Bit Breite gesetzt ist, werden eine bzw. zwei Speicherstellen angesprochen. Analoges gilt für Transfers zwischen den Indexregistern und Speicheradressen – hier ist die Anzahl der angesprochenen Speicherstellen vom Stand des X-Flags abhängig.
Etwas schwieriger wird es mit der Wortbreite schon, wenn es sich um Speicheroperationen ohne direkte Beteiligung der Register handelt, d.h. Befehle wie ROR oder STZ. Die Entwickler des 65816 haben sich hier ebenfalls an der Breite des Akkus orientiert: Wenn der Akku auf 16 Bits gesetzt ist, arbeiten Schiebebefehle und Rotationen mit zwei aufeinanderfolgenden Speicheradressen, der Befehl STZ setzt zwei aufeinanderfolgende Speicherstellen auf den Wert $00. Hat der Akku eine Breite von 8 Bits, dann wird jeweils nur mit einem Byte bzw. einer Speicheradresse gearbeitet.
Was passiert, wenn z.B. der Akku auf 8, die Indexregister auf 16 Bits gesetzt sind und der Befehl

```
TXA             ; Inhalt von X -> Akku
```

gegeben wird? Per Konvention ist festgelegt, daß sich bei Transfers, die von einem zu einem anderen Register stattfinden, die Anzahl der Bytes immer nach der Breite des *Zielregisters* richtet. Ein Beispiel:

```
REP #$3Ø        ; Akku, X und Y: 16 Bits
LDA #$2167      ; Akku <- $2167
SEP #$2Ø        ; Akku auf 8 Bits
LDX #$A232      ; X-Register <- $A232
TXA             ; Inhalt von X -> Akku
REP #$2Ø        ; Akku auf 16 Bits
```

Der Akku enthält nach diesen Operationen den Wert $2132 – der Befehl TXA hat nur das niederwertige Byte verändert. In die umgekehrte Richtung funktioniert das allerdings nicht:

```
REP #$3Ø        ; Akku, X und Y: 16 Bits
LDX #$2167      ; X-Register <- $2167
SEP #$1Ø        ; X und Y auf 8 Bits
LDA #$A232      ; Akku <- $A232
TAX             ; Inhalt vom Akku -> X
REP #$1Ø        ; X, Y wieder auf 16 Bits
```

Das X-Register enthält nach diesen Operationen den Wert $0032, weil der Befehl SEP #$10 seinen höherwertigen Teil gelöscht hat.

Neben dem Datentransfer zwischen dem Akku und den Indexregistern sind drei weitere Register-zu-Register-Operationen möglich:
1) Akku → Stackpointer („TCS") und Stackpointer → Akku („TSC"): Wie durch das „C" angedeutet, ignoriert dieser Befehl die momentan gesetzte Breite des Akkus, es werden immer 16 Bits übertragen.
2) Akku → Direct-Register („TCD") und Direct-Register → Akku („TDC"): Hier gilt dasselbe, die gesetzte Breite des Akkus wird ignoriert.
3) X-Register → Stackpointer („TXS") und Stackpointer → X-Register („TSX"): Diesen Befehl hätten die Entwickler des 65816 wohl am liebsten wieder gestrichen – leider ist er beim 65(C)02 die einzige Möglichkeit, mit dem Stand von SP zu arbeiten. Wenn das X-Register auf 8 Bit Breite gesetzt ist, dann hat sein höherwertiger Teil immer den Wert $00, der Befehl TSX liefert in diesem Fall also nur den niederwertigen Teil von SP. Ein Transfer in die umgekehrte Richtung hat ein (meist) katastrophales Ergebnis: Da der Stackpointer im „Native Mode" immer 16 Bit breit ist, wird er durch den Befehl TSX auf den Wert $00xx gesetzt. Dieses Verhalten ist der einzige Grund, warum ein für den 65(C)02 geschriebenes Programm im „Native Mode" auch dann nicht funktionieren kann, wenn A, X und Y auf 8 Bit Breite gesetzt sind. Die häufig verwendete Befehlsfolge

```
LDX #$FF    ; X-Register <- $FF
TXS         ; Inhalt von X -> SP
```

sollte eigentlich den Stackpointer auf die höchste Position (d.h. auf $01FF) setzen – im „Native Mode" kommt dabei der Wert $00FF heraus.

5. Reset, BRK, IRQ, NMI usw.

Die Prozessoren der 65er-Familie reagieren auf eine Unterbrechungsanforderung folgendermaßen:
– Speichern des PC (bzw. BP/PC) und des P-Registers auf dem Stack (außer bei Reset);
– Lesen von zwei Bytes aus dem Speicherbereich ($00/)FFFx;
– Einsetzen der gelesenen Werte in den PC.

Ein Beispiel: Wenn die Speicheradressen $FFFC/D die Werte $62 $FA enthalten, dann bewirkt ein Reset-Impuls einen Sprung zur Adresse $FA62 – von dieser Adresse liest der Prozessor den nächsten Befehl.
Aus diesem Verhalten ergibt sich eine grundsätzliche Forderung an den elektronischen Aufbau einer 65xx-Maschine: Der Bereich ($00/)FFFx muß zumindest beim Start durch ein ROM besetzt sein. Ältere „Apple-Hasen" werden sich noch mit Schrecken an die Folgen eines Reset bei aktivierter „Language Card" (und undefinierten Reset-Vektoren) erinnern.
Der 65(C)02 kennt nur drei verschiedene Vektoren, der 65816 dagegen sechs (eigentlich sind es sogar neun). Die Vektoren des 65816 werden ebenfalls aus der Speicherbank 00 gelesen, sämtliche Behandlungsroutinen müssen zumindest in der Bank 00 beginnen. Fangen wir mit dem wichtigsten an:

5.1. Reset

Sowohl der 65(C)02 als auch der 658xx lesen nach einem Reset-Impuls die Speicheradressen $(00/)FFFC/D als Vektor und setzen den gelesenen Wert in den PC ein. Die einzige zusätzliche Operation, die der 65(C)02 bei einem Reset ausführt, ist ein SEI, also das Sper-

ren von IRQ-Anforderungen. Alle anderen Flags (inklusive dem D-Bit) bleiben unverändert, eine Speicherung von P und PC auf dem Stack findet nicht statt.

Der 658xx führt bei einem Reset eine ganze Reihe zusätzlicher Schritte aus:
- durch Setzen des E-Bits wird auf „Emulation Mode" umgeschaltet. Folge: Die Bits M und X werden ebenfalls gesetzt, die höherwertigen Bytes von X- und Y-Registern erhalten den Wert $00, der höherwertige Teil des Stackpointers den Wert $01. Die niederwertigen Teile von SP, X und Y sowie der gesamte Inhalt des Akkus (AH und AL) bleiben dagegen unverändert.
- Im P-Register wird das D-Bit gelöscht, IRQ-Anforderungen werden (wie beim 65(C)02) gesperrt. Die Flags N, V, Z und C bleiben unverändert bzw. undefiniert.
- Das Direct-Register (D) erhält den Wert $0000.
- Programm- und Datenbankregister (PB und DB) werden auf $00 gesetzt.

So kompliziert es damit wird, ein 65816-Programm „resetfest" zu machen – wenn diese Schritte nicht ausgeführt würden, könnte ein 65802 in einer 65(C)02-Maschine wie dem IIe nicht funktionieren. (Die Reset-Routine im Monitorprogramm des IIe enthält selbstverständlich keine Befehle, die „Emulation Mode" setzen oder dafür sorgen, daß auch wirklich der Speicherbereich $0000..$00FF als „Seite 0" benutzt wird.)

5.2. BRK und IRQ

Ein Hardware-Interrupt wird durch ein Signal am IRQ-Eingang des Prozessors erzeugt und hat nur dann eine Wirkung, wenn das I-Flag („IRQ-Sperrbit") im P-Register gelöscht ist.
Auf den Befehl BRK reagiert der Prozessor mit einem Software-Interrupt. Diese Unterbrechung kann nicht durch ein gesetztes IRQ-Sperrbit maskiert werden, sie wird immer ausgeführt.
Beide Arten von Interrupts haben bei einem 65(C)02 oder einem 658xx im „Emulation Mode" dieselbe Wirkung:

– Der momentane Stand des PC (bei BRK: PC + 2) wird auf den Stack gebracht, zuerst das höherwertige, dann das niederwertige Byte.
– Das P-Register wird auf den Stack gebracht. Wenn die Unterbrechung durch einen IRQ erfolgte, wird anstelle des Bit 4 eine „0" geschrieben, ansonsten eine „1". (Das Bit 4 des P-Registers ist beim 65(C)02 im „Normalbetrieb" nicht definiert).
– Weitere IRQs werden durch Setzen des I-Flags gesperrt.
– Sprung zu der Adresse, die von den Speicherstellen $(00/)FFFE/F gelesen wird.

Achtung: Das Register PB wird auf den Wert $00 gesetzt. Ein Programm, das im „Emulation Mode" außerhalb der Bank 00 läuft, darf also nicht durch einen IRQ oder BRK unterbrochen werden!
Die Routine zur Behandlung von IRQ/BRK muß anhand des „gepushten" P-Registers selbständig zwischen IRQ und BRK unterscheiden, also:

```
STA $xx         ; Akku retten
PLA             ; P-Register!
PHA             ; wieder zurück
AND #$10        ; Bit 4 gesetzt?
BEQ IRQSET      ; -> ist IRQ
<..>            ; -> ist BRK
```

Der Befehl RTI („Return from Interrupt") holt (im Gegensatz zu einem RTS) sowohl den PC als auch das P-Register wieder vom Stack.

Im „Native Mode" verhält sich der 658xx etwas anders – eine Unterscheidung innerhalb der Behandlungsroutine zwischen IRQ und BRK ist nicht mehr nötig, weil zwei verschiedene Vektoren benutzt werden.

– Ein BRK liest die Adressen $00/FFE6/7 als Vektor und hat ein Argument, ist also ein Zwei-Byte-Befehl („BRK $xx"). Das zweite Byte wird lediglich gelesen und hat keine weitere Funktion als das „Durchnumerieren" gesetzter Unterbrechungspunkte. Solange bei der Programmierung ausschließlich mit absoluten Adressen gearbeitet wird, erscheint diese Möglichkeit etwas arg luxuriös, schließlich kann man über die „Break-Adresse" feststellen, an welcher Stelle des Programms der BRK stattgefunden hat. Ein System wie der IIgs bestimmt dagegen die Adreßlage eines Programms erst unmittelbar vor seinem Start – ohne die Möglichkeit numerierter Breakpoints wäre das „Debugging" deshalb mehr als kompliziert.
– ein IRQ liest dagegen die Adressen $00/FFEE/F.

Der „Trick" mit dem Bit 4 des P-Registers ist hier nicht mehr nötig – er wäre auch gar nicht mehr möglich, weil mit diesem Bit der momentane Stand des X-Flags festgehalten wird. Bit 5 des gespeicherten P-Registers enthält das M-Flag.
Zusätzlich zu PC und P wird das Programmbank-Register auf dem Stack gespeichert, so daß sich die folgende Belegung des Stacks nach einem IRQ oder BRK ergibt:

```
          PB        Programmbank-Register
        *)PCH       PC, höherwertiger Teil
        *)PCL       PC, niederwertiger Teil
          P         Flags
SP    ->
```

*) s. Abschnitt 5.2.1
Das PB-Register wird auf den Wert $00 gesetzt, die Behandlungsroutine muß also innerhalb der Bank 00 beginnen.
Der Befehl RTI liest im „Native Mode" vier Bytes vom Stack (entspricht also im Prinzip der Befehlsfolge PLP/RTL) und stellt somit nicht nur die Register P und PC, sondern auch den vorherigen Inhalt des Programmbank-Registers wieder her.
An dieser Stelle sei noch eine Merkwürdigkeit erwähnt (vgl. 3.1.): Alle Prozessoren der 65er-Familie speichern bei einem JSR- bzw. JSL-Befehl eine Returnadresse auf dem Stack, deren Wert um eins zu niedrig ist. Dieser „Fehler" wird dadurch korrigiert, daß ein RTS bzw. RTL den vom Stack zurückgelesenen Wert um eins erhöht, bevor er in den PC eingesetzt wird. Eigentlich wäre dieses Verhalten kein sonderliches Problem – wenn der Prozessor nicht bei einem IRQ die „richtige" Adresse speichern würde: Der Befehl RTI nimmt folglich *keine* Erhöhung der Returnadresse vor. Eine Routine, die via IRQ aufgerufen wird, darf also nicht mit einem RTS oder RTL enden – ansonsten überspringt der Prozessor ein Byte in dem Programm, das durch den IRQ unterbrochen wurde.

5.3 NMI

Diese Unterbrechung wird durch einen Impuls am NMI-Eingang des Prozessors ausgelöst. Sie ist nicht sperrbar („Non Maskable Interrupt") und läuft je nach Prozessortyp folgendermaßen ab:
– 65(C)02 bzw. „Emulation Mode": PC und P (unverändert) auf den Stack, Lesen der Speicheradressen ($00/)FFFA/B als Vektor und Sprung zur entsprechenden Adresse. Der Befehl RTI holt das P-Register sowie den PC wieder vom Stack und setzt das unterbrochene Programm fort.
– 658xx im „Native Mode": PB/PC und P (in derselben Reihenfolge wie bei einem IRQ/BRK) auf den Stack, Lesen der Speicheradressen $00/FFEA/B und Sprung zur entsprechenden Adresse. Der Befehl RTI holt das P-Register sowie PB/PC wieder vom Stack und setzt das unterbrochene Programm fort.
Das PB-Register des 658xx wird in beiden Fällen auf den Wert $00 gesetzt. Ein in „Emulation Mode" laufendes Programm darf sich also nicht außerhalb der Bank 00 befinden – nach dem RTI geht es sonst in Bank 00 weiter.

5.4. COP und ABORT

Diese beiden Unterbrechungsmöglichkeiten sind nur bei den 658xx-Prozessoren definiert, sie sind sowohl im „Native Mode" als auch im „Emulation Mode" verfügbar. Ein ABORT wird durch ein extern zugeführtes Signal ausgelöst und unterbricht den Prozessor mit *sofortiger* Wirkung. Hauptsächlicher Verwendungszweck: Unterbrechung bei elektronischen Fehlern (für 68000-Fans: BERR-Signal).
Ein COP wird durch den Befehl COP („Coprozessor") ausgelöst und hat ein Argument (COP #$xx), entspricht also in etwa dem TRAP-Befehl des 68000.
Beide Unterbrechungsmöglichkeiten sind nicht maskierbar und haben aus der Sicht des Programmierers denselben Ablauf wie ein NMI: Im „Emulation Mode" werden PC und P-Register, im „Native Mode" zusätzlich das PB-Register auf den Stack gebracht. COP liest die Adressen $00/FFE4/5 als Vektor, ABORT benutzt $00/FFE8/9. Ein RTI im „Emulation Mode" holt drei, ein RTI im „Native Mode" dagegen vier Bytes vom Stack und setzt so das unterbrochene Programm fort. Auch hier gilt, daß sich ein im „Emulation Mode" laufendes Programm nicht außerhalb der Bank 00 befinden darf, weil sonst der Inhalt von PB verändert wird.

Hinweis:
Im Teil 2 dieses Kompaktkurses erläutert A. Schäpers die Adressierungsarten des 65(C)02 (der 65816 kennt im „Emulation Mode" und im „Native Mode" sämtliche Adressierungen des 65(C)02) und die Adressierungsarten, die der 65816 zusätzlich erhalten hat. Stackoperationen und Blocktransfers werden ausführlich beschrieben.

PEEKER Börse

Biete Hardware

Apple II europlus
mehrere Laufwerke, 2FD-Controller, Z80, 80Z, 64K Ram, Metall-Gehäuse, gr. Tastatur und viel SW. Preis: DM 1.500,–.
Telefon 0 76 21 / 8 81 68

Softcard IIe (80Z., Z80 mit CP/M und 64K) DM 600,–; Extended 80 Zeichen Karte mit 64K DM 280,–; beides Originale;
Tel. 0 21 51 / 30 21 77 oder 02 41 / 3 67 74 Andreas Grün.

Z80H, AP22, 8Mhz, komplett DM 380,–. Erphi Controller DM 120,–. AE-RAMworks III 1MB DM 850,–. Tel. ab 18.30 h.
Telefon 0 25 07 / 14 90.

Wir bieten Applied engineering: Ramworks 512K IIe 800,–, Z-RAM 999,–, Ramfactor 512K IIe/II+ 850,–, Transwarp 3,6 × schneller 825,–. Bitter Labortechnik, Große Bleiche 27, 3050 Steinhude, Tel. 0 50 33 / 15 22 ©

APPLE II+; 2LW; Bernstein-Monitor; 80Z; RS232; Parallel-Schnittst., Druckerinterf., Arith.-Proz. 32Bit; Softw: Wordst. DBASE; TurboPascal; Assembler; ProDOS, u.v. mehr VB DM 1400,–.
Telefon 0 74 61 / 86 72

Verkaufe: Apple IIc + 2.LW+Imagew. + Bildsch. + Mouse+Appleworks + Software +Bücher+Peeker+Peekerdisk + Uhr IIc DM 3975,–.
Tel. 02 35 / 4 44 90

APPLE IIc + Z-RAM 512K mit CP/M 4.0 + Monitor + Imagewriter + alle Betr. Syst. DOS/PRO-DOS + Textverar. + Pascal + Peeker + Sammeldisk. DM 3600,–
Tel. 0 68 87 / 15 84

Original Apple Interface IEEE488 mit Handb. DM 490,–.
Tel. 0 51 21 / 4 66 95

Apple IIe-komp. neue ROMS, Z80 Maus 80Z + 64K, Drucker +Epson-Interface, 35-80 Track-LW+Controller, Lüfter, Eprommer, Literatur Preis: VB 2900,–.
Tel. 0 73 32 / 61 29 ab 17 h

Apple II mit AT Look, 2 Laufwerke Monitor bernstein sep. Tastatur Epson MX80 Drucker, Software, VB 2200,–.
Tel. 0 97 21 / 8 94 77 nach 16 Uhr.

APPLE IIe, Alphakey, ImageWriter POLAROID PALETTE, Zubehör, preiswert.
Telefon 0 71 43 / 9 18 36.

APPLE IIc kompl. mit APPLEWORKS u.a. Prog. 1900,–. BROTHER HR15 5m Kabel 900,–.
Telefon 02 03 / 76 74 99.

Apple IIe, 128KB, N80e, 2×Disk II, Sanyo DM 8212, Matrixdrucker NEC, VB DM 2750,–.
Tel. 0 22 04 / 6 28 62.

Premium-Softcard IIe, Z-80 6 MHZ DM 499,– komplett.
Tel. 0 92 52 / 10 11

Gewerbliche Anzeigen sind mit © gekennzeichnet.

Apple IIc, mit Monitor, 2. Laufwerk, u. CP/M Card, sowie umfangr. Literatur u.v.a. zu verkaufen. Preis VB. Tel. 089 / 4 30 09 85

Biete Software

APPLE BUSINESS GRAPHICS + auf Wunsch Druckeranpassung GEMINI STAR 10X DM 380,–. Tel. 0 61 07 / 6 13 25

IIe+c: CAD paint: exakte Zeichnungen m. Beschriftung (2 Stärken), 40 Funktionen, 80 fertige Symbole für Wohnungspläne + elektr. Schalt. Info DM 2,–. Briefm., Disk: DM 99,– Scheck. Lohmann, M.-Müller-Ring 7, 6500 Mainz

TURBO Pascal Turtlegrafik-Paket für Apple II, kennt alle Befehle der UCSD-Unit, DM 89,– von Fa. Jochen Tucht, Rudolf-Albrecht-Straße 52, 3052 Bad Nenndorf. ©

Apple II: DFÜ-Kermit, Pascal satt, Public Domain in DOS u. CP/M. Je Volume DM 15,–. Bahnhofssimulation, Sprachen (S.A.L.), Schulprogr. Gratisinfo: Fa. Waltraud Muhle, Waldwinkel 3, 2105 Seevetal 3 ©

*** Disketten ***
*5 1/4", 48 tpi, DM 0,99, 2D *
*3 1/2", 135 tpi, DM 3,19, 1 DD *
*auch andere, bes. Garantie *
*Allg. Austro-AG, Ringstraße 10 *
*D-8057 Eching, T. 08133/6116 *
©

Profi. CAD-Prog. CASCADE I für Apple IIe preisgünstig abzugeben. Preis VH. Tel. 0 42 03 / 21 60 n. 18.00 Uhr.

Topsoftware für Apple II: Sporttabellenverwaltung: nur 40,– PFS-Programme: 300,–, BALLblazer: 100,–, Superbase: 300,–. Info (0,80 DM) bei: W. Rittmeyer, Wehrbruchweg 30, 4060 Viersen 1 ©

AppleWorks-Praktikum, Lern- und Nachschlagewerk, über 250 Seiten mit vielen Abbildungen 49,–; mit Diskette 65,–; Templates für Datenbank mit Diskette je 15,–; Hans-Jürgen Kühne, Neisserstr. 2, 3008 Garbsen 9

****** ******
MAGNAT Das beste Apple-Pascal-Disk-Bearbeitungsprogramm! Disk+Manual: 34.90 Info per 50 Pf-Rückumschlag.
Röhl, Hornerstr. 9, 2800 Bremen

Finanzbuchhaltung. Frei wählbarer Kontenrahmen, Summen- und Saldenliste, Journal, Kontoauszüge autom. UST-Verbuchung ... DM 300,– (II+) BM-Software, Poetenweg 44, 4800 Bielefeld 1 ©

Neu! Kyan Includefile-Tool II- sehr schnelle Hires-Routinen und Turtle-Grafik.
DM 50,– (Schein/V-Scheck): Info gegen Rückporto bei: R. Winzen, Daimlerstr. 70, 4040 Neuss 1

Gewerbliche Anzeigen sind mit © gekennzeichnet.

MULTISCRIBE 180,–. MACROWORKS 90,–, THINKTANK 350,–, CHESSMASTER 2000 80,–. Alle Original. Tel. 0 22 26 / 30 08.

Suche Software

ProDOS-Treiber für SATURN 128K
RAM-DISK J. Flacher, Kornweg 2, CH-8405 Winterthur, Tel. 0 14 35 / 43 16 od. 052 / 28 23 92

Suche Privatliquidation Arztp. für Apple IIe, 128 K 2. Laufw. Kopie oder Original.
Tel. 0 56 01 / 8 67 47

Fundgrube, Textbearbeitung in Maschinenprogrammen 38,–, Relocator 38,–, Einfügen-Entfernen von Programmteile 52,–, Copy-File für Erphi 78,–. Info gegen Freiumschlag: Manfred Herrmann, Südwest Korso 11a, 1000 Berlin 41

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 ©

Microsoft Softcard II, Z80B, CPM 2. Z8, Handbücher DM 700,–, Apple II Mouse Mousepaint DM 350,–, NEC8023BN Matrixdrucker DM 700,–, Peeker 01/84-12/86 gegen Gebot. T. 0 22 04 / 6 28 62

Familienforschung mit Computer! Zeitschrift + Computergenealogie (3. Jg.) bietet Hilfe und schafft Kontakte. Pro Jahr 4 Hefte DM 12,–. K. Thomas, PF 13 44, 5778 Meschede

Welche Lehrer (Gymnas.) sind am Tausch von Unterrichtsmaterial zu Biologie, Chemie, Physik im AppleWorks-Format (oder AWriter) interessiert? Chiffre-Nr. **P1009**

Suche Kontakt zu MS-Cobol 80 Ugern und MS-Cobol 80 Programme im Source-Code.
L. Elsenberg, Tel. 0 28 37 / 84 53.

Verkaufe Peeker 1/85-12/86 und BASIC/Manuals 1/2.
T. 05 11 / 46 69 48

Verkaufe Peeker 1/84-12/86 + Sammeldisk Nr. 1-19 + Turtle Graphics-Library + Softbreaker VB DM 250,–. Tel. 0 26 33 / 9 61 47 nach 18 Uhr.

Wer hilft? Wie können Textfiles schneller bearbeitet werden? – (Apple II+, 2 × 128K-Saturn als Pseudo-Disk, Erphi-Contr. mit 2 × (2 × 80 Track), Div-DOS oder evtl. gepachtes DOS?).
Steffen Neugebauer, Schönwaldstr. 19, 8721 Hesselbach.

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 GmbH, Postfach 10 28 69, 6900 Heidelberg 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

Für Ihre Unterlagen

Folgende Bücher bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Versandbuchhandlung
Postfach 10 28 69
6900 Heidelberg 1

Für Ihre Unterlagen

Folgende Disketten
und Programme bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Softwareabteilung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Abo-Karte

Ja, ich möchte **Peeker** abonnieren.
Liefern Sie mir **Peeker** ab Ausgabe . . . . . zum Jahresbezugspreis von z.Zt. DM 75,– (Inland) inkl. MwSt. Die Lieferung erfolgt frei Haus. Porto, Verpackung und Zustellgebühren übernimmt der Verlag. Der Jahresbezugspreis für das Ausland beträgt z.Zt. DM 75,– plus DM 20,– Versandspesen.

X _____

Datum 1. Unterschrift

Bitte lesen!

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

X _____

Datum 2. Unterschrift

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

Wir können nur Bestellungen mit zwei Unterschriften bearbeiten.

PEEKER

Buch-Karte

Bitte senden Sie mir gegen Rechnung folgende Bücher:

- ☐ Bühler, Applesoft-BASIC, 3-7785-1094-0, DM 38,–
- ☐ Eggerich, dBase II, Bd. 1, 3-7785-1147-5, DM 39,80
- ☐ Eggerich, dBase II, Bd. 2, 3-7785-0987-X, DM 39,80
- ☐ Eggerich, dBase II, Bd. 3, 3-7785-0988-8, DM 39,80
- ☐ Gabriel, Applewriter, 3-7785-1234-X, DM 35,–
- ☐ Hagenmüller, Microsoft-BASIC, Bd. 1, 3-7785-1038-X, DM 38,–
- ☐ Juhnke/Redlin, Apple Pascal, Bd. 1, 3-7785-1246-3, DM 42,–
- ☐ Kehrel, Apple Assembler lernen, Bd. 1, 3-7785-1151-3, DM 38,–
- ☐ Kehrel, Apple Assembler lernen, Bd. 2, 3-7785-1170-X, DM 38,–
- ☐ Schäpers, ProDOS Analyse, 3-7785-1134-3, DM 68,–
- ☐ Schäpers, Bewegte Apple-Graphik, 3-7785-1150-5, DM 58,–
- ☐ Stiehl, Apple DOS 3.3, 3-7785-1297-8, DM 28,–
- ☐ Stiehl, Apple ProDOS, Bd. 1, 3-7785-1098-3, DM 28,–
- ☐ Stiehl, Apple ProDOS, Bd. 2, 3-7785-1036-3, DM 30,–
- ☐ Stiehl, Apple Assembler, 3-7785-1047-9, DM 34,–
- ☐ Wassermann, Apple IIc Handbuch, 3-7785-1157-2, DM 35,–

_____

Datum Unterschrift

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,–
- ☐ Apple DOS 3.3, Begleitdisk., DM 28,–
- ☐ ProDOS, Band 1, Begleitdisk., DM 28,–
- ☐ ProDOS, Band 2, Begleitdisk., DM 28,–
- ☐ Apple Assembler, Begleitdisk., DM 28,–
- ☐ ProDOS-Editor 1.0, Programm, DM 98,–
- ☐ MMU 2.0, Programm, DM 98,–
- ☐ INPUT 2.0, Programm, DM 98,–
- ☐ DB-Meister, Programm, DM 290,–
- ☐ Superquick, Programm, DM 48,–
- ☐ Turtle Graphics, Programm, DM 98,–
- ☐ Disk 40, Programm, DM 48,–
- ☐ Kyan-Pascal 2.0, Programm, DM 170,–
- ☐ Fast-Writer, DOS 3.3, DM 128,–
- ☐ Fast-Writer, ProDOS, DM 128,–
- ☐ Double-Hires-Tools für Applesoft, DM 28,–
- ☐ Double-Hires-Tools für Kyan, DM 28,–
- ☐ Kyan-Toolkit Nr._____, DM_____

_____

Datum Unterschrift

Abo-Karte

Name

Firma

Straße

PLZ/Ort

Ich wünsche jährliche Berechnung durch:
☐ Verlagsrechnung ☐ Abbuchung von meinem Bank- bzw. Postscheckkonto

Bank/PschA

Bankleitzahl

Kto.-Nr.

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

Bitte freimachen

POSTKARTE

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

Bitte freimachen

POSTKARTE

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

Bitte freimachen

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

DISK 40

Disketten-Organisationsprogramm für DOS-3.3-35-40 Spuren

von Hermann Seibold und Dipl.-Ing. Udo Marin, 1986, Programmdiskette mit Anleitung, DM 48,–

Durch eine einfach zu bedienende Menüführung können DOS-3.3-Disketten umfangreich bearbeitet oder kopiert werden.
– Tabellarische Ausgabe der Diskettenbelegung
– Ordnen des Catalogs
– „Undelete"n von versehentlich gelöschten Dateien
– Vergleichen von Disketten, Dateien oder DOS-Spuren
– Kopieren von Disketten, Dateien oder DOS-Spuren
– Formatieren von Daten-Disketten
– Erweitern auf 40 Spuren bei bestehenden 35-Spur-Disketten
– Ändern des Boot-Programms
– File-Editor zum Editieren von Disketten-Dateien
– Komfortabler Sektor-Editor für Hex- und ASCII-Darstellung
–VTOC-Editor, z. B. zur Freigabe der DOS-Spuren

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

Grafikausdruck anschaulich erklärt

Mit Apple- und Atari-Algorithmen für den 24-Nadel-Drucker LQ-800

von Ulrich Stiehl

Im Peeker sind bereits mehrere Hires-Grafik-Druckprogramme erschienen, von denen das äußerst flexible „Superdump"-Programm (Heft 6/85) für den Epson FX80 und den Imagewriter am häufigsten eingesetzt wird. Viele Leser hatten allerdings Schwierigkeiten, Superdump für andere Drucker oder andere Interface-Karten anzupassen, weil wir niemals in einer vereinfachten und anschaulichen Form erklärt haben, wie Grafikdruckprogramme funktionieren. Dies soll mit dem vorliegenden Beitrag nachgeholt werden, der einen kurzen und hoffentlich durchsichtigen Algorithmus für den Atari ST und den Apple II beschreibt. Zwar drucken wir auch für jeden der zwei Rechner ein Beispielprogramm ab, doch liegt das Schwergewicht dieses Aufsatzes auf der Beschreibung der Grundlagen des Algorithmus und nicht auf der Beschreibung der konkreten Programme, da hierzu GFABASIC- bzw. Assemblerkenntnisse vorausgesetzt werden müßten.
Als Drucker wurde der zur Hannover-Messe 1986 vorgestellte LQ-800 von Epson gewählt, der mit seiner 24-Nadel-Matrix Grafiken mit einem 70er Raster ausdrucken kann. Dies sind rund 5.000 Bildpunkte pro qcm!

1. Grafikbildschirm

Punktzeilen und Punktspalten

Den Grafikbildschirm können wir uns als ein „kartesisches Rechteck" vorstellen, dessen Ursprung (Origo) sich allerdings beim Apple und Atari im Gegensatz zum normalen kartesischen Koordinatensystem links oben befindet. Dabei nennen wir die Parallelen zur waagrechten X-Achse *Grafikzeilen* und die Parallelen zur senkrechten Y-Achse *Grafikspalten*. Der Apple-Bildschirm hat danach 280 Spalten (numeriert von 0-279) und 192 Zeilen (numeriert von 0-191); vgl. **Abb. 1a**. Analog gelten für den Atari 640 Spalten und 400 Zeilen; vgl. **Abb. 1b**. Damit beschreiben wir nur die jeweils verständlichste Grafikauflösung beider Rechner und ignorieren in diesem Beitrag die Double-Hires des Apple (560 x 192) sowie den gesamten Komplex der Farbgrafik.

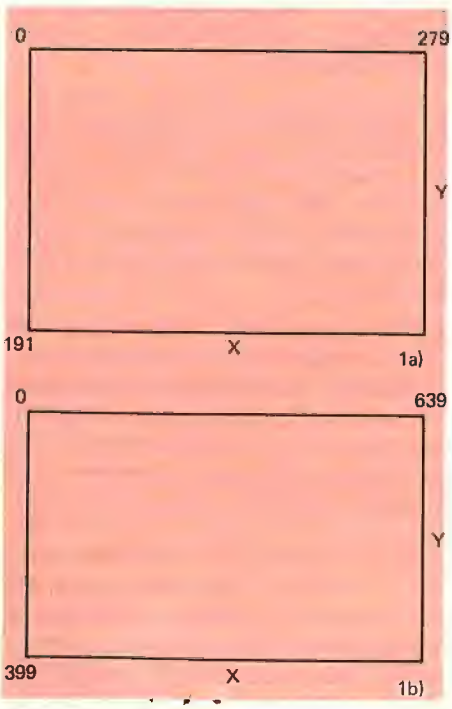


Abb. 1: Grafikbildschirm-Koordinatensystem beim Apple (1a) sowie beim Atari (1b)

Da sich die Spalten und Zeilen aus einzelnen Bildpunkten = Rasterpunkten = Pixeln aufbauen, können wir den Grafikbildschirm auch als eine *Punktmatrix* auffassen, die sich aus einer bestimmten Anzahl von Punktzeilen oder Punktspalten zusammensetzt, z.B. beim Atari aus 400 Punktzeilen mit jeweils 640 Bildpunkten oder umgekehrt 640 Punktspalten mit jeweils 400 Bildpunkten.

Konstante und variable Punktgröße

In der Computergrafik sind Rasterpunkte entweder kleine Vollkreise (Apple, IBM) oder kleine Quadrate (Atari, Macintosh), so daß idealerweise zwei nebeneinanderliegende Punktquadrate zu einem Doppelquadrat verschmelzen. Dabei entstehen Grautöne bei einer Grafik dadurch, daß bei einem entsprechenden Betrachtungsabstand die schwarzen (beim Apple weißen) Rasterpunkte optisch mit dem weißen (beim Apple schwarzen) Hintergrund zu einem Grauton verschmelzen.
In der drucktechnischen Reproduktionsphotographie zur Herstellung von Abbildungen in Büchern usw. werden die echten Grautöne von Photographien ebenfalls durch unechte Grautöne = Rasterpunkte simuliert, wie man mittels einer Lupe mit mindestens 10facher Vergrößerung („Fadenzähler") leicht nachprüfen kann. Dies

sollten Sie jetzt auch tun, wobei Sie insbesondere die Übergänge zwischen hellen und dunklen Bildstellen unter die Lupe nehmen sollten, weil Sie dort den Punktgrößenzuwachs am besten beobachten können.
Der *Grauwert* (= auf optischer Täuschung beruhender Schwärzungsumfang) eines reprographischen Rasterfilms hängt jedoch nicht nur von der Anzahl der Rasterpunkte pro cm ab (z.B. 70er Raster = 70 Punkte pro cm oder 70 mal 70 = 4900 Punkte pro qcm), sondern insbesondere von der Größe der Rasterpunkte selbst, wobei ein einzelner Rasterpunkt von einem mikroskopisch kleinen Pünktchen bis zu einem vollen Punktkreis anwachsen kann. Bei „digitalen" Computern ist dieses „analoge" Prinzip, das zu einer erheblich besseren Bildqualität führt, zur Zeit kaum denkbar, denn der Grafikbildschirm setzt sich beim Apple aus immerhin 280 x 192 = 53.760 Punkten und beim Atari bereits aus 640 x 400 = 256.000 Punkten zusammen. Gäbe es nämlich für jeden einzelnen Punkt noch beispielsweise 100 verschiedene Punktgrößen vom winzigen Pünktchen bis zum vollen Punktquadrat, dann müßten geradezu gigantische Grafikspeicher vorhanden sein. Das in der Reproduktionsphotographie zur Realisierung von Grauwerten angewandte Verfahren variabler Punktgrößen wird man deshalb insbesondere bei Mikrocomputern vorab nicht realisieren können. Um einen einzelnen Rasterpunkt zu vergrößern, muß man vier gleichgroße Rasterpunkte im Karree anordnen, worauf wir später noch eingehen werden.

2. Bildpunkt und Nadelpunkt

Der Druckkopf eines Nadel- oder Matrixdruckers enthält eine bestimmte Anzahl von senkrecht oder vertikal zur Druckzeile angeordneten Nadeln, die von Elektromagneten durch das Farbband auf das Papier gedrückt werden. Da der Druckkopf über die Nadeln mit dem Papier in Berührung gelangt, spricht man auch vom Impact-Verfahren im Gegensatz zum berührungslosen Non-Impact-Verfahren bei Farbstrahldruckern usw. Im übrigen geht die Terminologie ziemlich durcheinander, da bereits der Begriff des Matrix*druckers* falsch gewählt ist, denn der Druck im drucktechnischen Sinne setzt einen eigens zu diesem Zweck angefertigten *Druckträger* (z.B. Offsetplatte im Flachdruck, Kupferzylinder im Tiefdruck usw.) voraus, mit dessen Hilfe eine Vielzahl gleicher *Abdrucke* erzeugt werden kann, während umgekehrt ein Matrix„drucker" das *Druckbild* immer wieder neu generieren muß.

Das Druckbild kann bei einem Matrixdrucker auf zweifache Weise generiert werden:

Selbstgesteuerter Bildaufbau: Diese Form des Bildaufbaus ist typisch für den Ausdruck von Texten. Wenn wir beispielsweise über den Mikrocomputer den ASCII-Code für den Buchstaben „A" an den Drucker senden, so baut er das Buchstabenbild von „A" aus einer bestimmte Anzahl vertikaler Nadelspalten *selbst* auf, vereinfacht etwa folgendermaßen:

Der Programmierer muß sich also hier keine Gedanken über die Punktmatrix des Buchstabens „A" machen. Anders würden die Dinge liegen, wenn man eigene Zeichen definieren wollte, deren Matrix dann im RAM-Speicher des Druckers abgelegt werden müßte. *Nach* der Neudefinition der Matrix des Buchstabens „A" würde es jedoch wiederum genügen, daß man den ASCII-Code von „A" an den Drucker sendet und nicht dessen Punktinformationen.

Computergesteuerter Bildaufbau: Wenn nun jedoch Grafiken ausgedruckt werden sollen, so muß dem Matrixdrucker mitgeteilt werden, welche der Nadeln beim Aufbau der jeweiligen Nadelspalte aktiviert werden sollen, denn Anweisungen in der Art von „Print Mona Lisa" reichen leider nicht aus, weil die Zahl der (möglichen) Grafikbilder im Gegensatz zur Zahl der (üblichen) Buchstabenbilder schier unendlich groß ist.

Wir halten zunächst fest, daß die gesamte Druckgrafik aus einer bestimmten Anzahl von Druckzeilen besteht, die sich ihrerseits aus einer bestimmten Anzahl von Nadelspalten zusammensetzen, wobei die Nadeln einer Spalte selektiv aktiviert werden können.

Während herkömmliche Matrixdrucker oft nur über 8 Nadeln verfügten, befinden sich im Druckkopf des LQ-800-Epsondruckers (LQ = Letter Quality) 24 vertikal angeordnete Nadeln. Die Druckqualität ist deshalb bei Texten beeindruckend, wie man dem Ausdruck des Standardzeichensatzes entnehmen kann. Aber auch für den Grafikausdruck ergeben sich Qualitätsvorteile, weil in 24-Nadel-Zeilen gedruckt werden kann, wobei der vertikale und horizontale Nadelabstand 1/180 Zoll = ca. 0,14mm beträgt. Dies entspricht etwa einem 70er Raster (= 70 Nadelpunkte pro cm → 0,14 * 70 = ca. 10mm), der beispielsweise im Offsetdruck nur bei Farbvorlagen (z.B. Peeker-Titelseiten) benutzt wird. Dabei kann eine Zeile theoretisch aus 1440 diskreten Nadelspalten bestehen (1/180 x 1440 = 8 Zoll oder ca. 20,3cm), die jedoch praktisch nicht mehr unterscheidbar sind.

Daß die LQ-800-Druckqualität trotzdem immer noch von einer Offset-Druckqualität himmelweit entfernt ist, ist auf folgende Faktoren zurückzuführen:

– Die Papieroberfläche der normalen Endlosdruckpapiere ist für einen 70er Raster viel zu rauh. Außerdem fehlt es an der nötigen Dimensionsstabilität bzw. Zugfestigkeit des Papiers, obwohl Grafiken nur von links nach rechts, d.h. uni- statt bidirektional, ausgedruckt werden.
– Das Stoffarbband ist für derart feine Nadeln völlig ungeeignet. Man müßte eigentlich ein Einmalfarbband (Karbonfarbband) verwenden.
– Die Nadelpunktgröße ist ähnlich wie die Bildschirmpunktgröße nicht variabel mit der Folge, daß einerseits dunklere Bildstellen zu dunkel und hellere Stellen zu hell werden, womit die Grauskala nicht mehr stimmt.
Trotz alledem bietet der LQ-800 die bestmögliche Druckqualität, die mit heutigen Matrixdruckern erzielbar ist.

3. Makrozeilen

Die 24 vertikalen Nadeln des LQ-800 können wir von 0 (oberste Nadel) bis 23 (unterste Nadel) durchnumerieren. Da der Grafikbildschirm beim Apple aus 192 Punktzeilen (0-191) und beim Atari aus 400 Punktzeilen (0-399) besteht, könnte man theoretisch nur jeweils die Nadel 0 bei einem Vorschub von 1/180 Zoll (= Größe *einer* Nadel) benutzen und dann beim Apple die 192 und beim Atari die 400 Grafikpunktzeilen als entsprechende Nadeldruckzeilen auf den LQ-800 herauslassen. Tatsächlich gibt es von der Firma M.Gehret Useware in Grönenbach ein Atari-LQ-800-Dump-Programm für DM 100,-, das u.a. nach diesem Verfahren arbeitet. Der Ausdruck des Atari-Bildschirms dauert dann sage und schreibe über 7min und liefert zudem ein verschmiertes Bild. Dies ist natürlich Bauernfängerei, denn ein 100-DM-Programm für einen 24-Nadel-Drucker muß alle 24 Nadeln gleichzeitig benutzen können, denn sonst brauchte man keinen teuren 24-Nadel-Drucker zu kaufen.

Unser Grafikdruckprogramm verwendet selbstverständlich alle 24 Nadeln. Wir können deshalb die 192 Apple-Punktzeilen in exakt 8 Makrozeilen, numeriert von 0 bis 7, mit je 24 Punktzeilen aufteilen (8 x 24 = 192); vgl. **Abb 2a**. Das analoge Rechenexempel für den Atari mit seinen 400

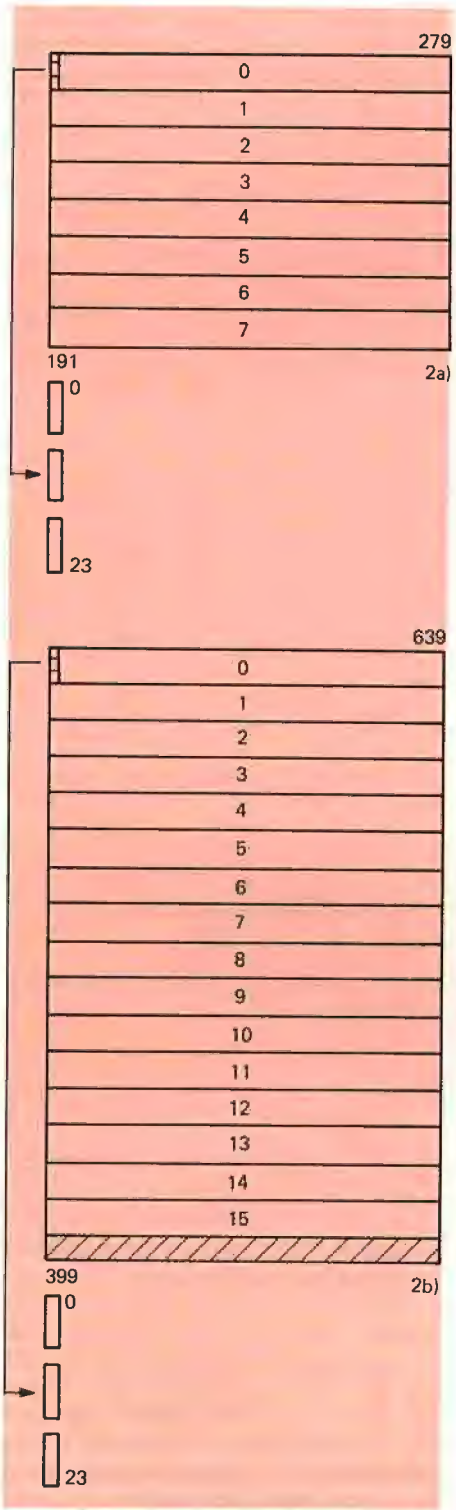


Abb. 2: 24-Punkt-Grafikbildschirm-Makrozeilen beim Apple (2a) sowie beim Atari (2b)

Punktzeilen geht jedoch nicht ganz glatt auf (400 : 24 = 16 komplette 24-Punkt-Makrozeilen und 1 Restmakrozeile mit nur 16 Punktzeilen); vgl. **Abb. 2b**. Somit muß der LQ-800-Druckkopf beim Apple 8mal und beim Atari 17mal bewegt werden, um einen kompletten Grafikausdruck zu erzeugen. Der Algorithmus ist damit für den Apple oder Atari folgender (vgl. Abb. 2a):

Algorithmus
Schritt 1: Setze den Makrozeilenzähler auf Null.
Schritt 2: Setze den Spaltenzähler auf Null.
Schritt 3: Hole die momentane Spalte der momentanen Makrozeile (= 24 vertikale Punkte) aus dem Grafikbildschirm und schicke sie an den Drucker.
Schritt 4: Erhöhe den Spaltenzähler um eins. Falls die 280. (beim Atari 640.) Bildschirmspalte noch nicht erreicht ist, gehe zurück zu Schritt 3.
Schritt 5: Sende einen Zeilenvorschub im Abstand von 24 Nadeln an den Drucker.
Schritt 6: Erhöhe den Makrozeilenzähler um eins. Falls die 8. (beim Atari 17.) Makrozeile noch nicht erreicht ist, gehe zurück zu Schritt 2.

4. „Liegende Zeilenbytes"

Mit dem obigen Algorithmus haben wir das Grafik-Dump-Problem grundsätzlich gelöst. Wenn wir uns jedoch jetzt an die Implementierung eines konkreten Dump-Programms machen, so stoßen wir auf unerwartete Komplikationen, die darauf zurückzuführen sind, daß die Bildschirmpunkte als „liegende Zeilenbytes" gespeichert werden, während der Drucker „stehende Spaltenbytes" erwartet.
Ein Byte besteht bekanntlich aus 8 Bits, die üblicherweise „liegend" von rechts nach links dargestellt und entsprechend von 0 bis 7 durchnumeriert werden. Ein Bit selbst kann den Wert 0 oder 1 annehmen. Dabei entspricht ein Einserbit einem gesetzten bzw. sichtbaren Punkt und ein Nullerbit einem gelöschten bzw. unsichtbaren Punkt. Ein Bildschirmspeicherbyte enthält somit 8 Bits und damit (theoretisch auch) 8 horizontale Rasterpunkte. Der Speicher des Apple wie auch des Atari ist also dergestalt organisiert, daß die gesetzten Bits eines Bytes für die sichtbaren Punkte einer **horizontalen** Punktfolge = Punktzeile stehen. Betrachten wir hierzu das allererste Byte des Bildschirmspeichers beim Apple und Atari und nehmen zusätzlich an, daß der zweite Bildpunkt von links, also Punkt 1 bei einer Zählung von 0 bis 279 bzw. 399, gesetzt sein soll:

```
Ø1234567 Punktnummern
  .      Punkt
76543210 Bit-Nummern
Ø1ØØØØØØ Atari-Bits
xØØØØØ1Ø Apple-Bits
```

Beim Atari muß Bit 6 gesetzt werden, um Punkt 1 zu erzeugen, während beim Apple Bit 1 gesetzt werden muß, um Punkt 1 zu erzeugen. Daraus folgt: Beim Atari zeigen die linken Bits zum linken Bildschirmrand (vgl. **Abb. 3b**), während beim Apple umgekehrt die rechten Bits zum linken Bildschirmrand zeigen (vgl. **Abb. 3a**). Beim Apple kommt erschwerend hinzu, daß Bit 7 als Farbbit reserviert ist und deshalb auch auf einem Schwarzweißmonitor keinen Rasterpunkt erzeugt. Demgegenüber gibt es beim Atari nur dann reservierte Farbbits, wenn ein Farbmonitor benutzt wird.

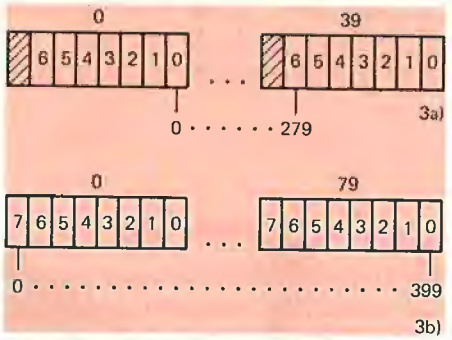


Abb. 3: Grafikbildschirm-Punkt-Reihenfolge der „liegenden Zeilenbytes" beim Apple (3a) sowie beim Atari (3b)

Test für Apple
Um sich von der Richtigkeit der umgedrehten Bitreihenfolge beim Apple zu überzeugen, kann man mit
CALL -151
in den Monitor gehen, nachdem man zuvor mit
HGR
die Grafik eingeschaltet hat. Wenn man nun
2000:02 (= Bitmuster 00000010)
eingibt, wird Punkt 1 von Punktzeile 0 sichtbar, während mit beispielsweise
2000:10 (= Bitmuster 00010000)
ein um drei Stellen nach *rechts* versetzter Punkt generiert wird.

Test für Atari
Das nachfolgende GFABASIC-Programm namens PUNKT.BAS demonstriert einen von links nach rechts wandernden Punkt, der durch eine 32-Bit-Langwort-Maske (Punktspalte 0 bis 31) erzeugt wird. Dazu setzen wir zunächst in Punktspalte 32 einen „Grenzpunkt", dem sich der wandernde Punkte allmählich nähert, wobei zusätzlich die jeweils gepokte Bitmaske als Binärzahl angezeigt wird. Die 32 Langwort-Masken repräsentierten folgende Bit-Muster:

```
L% (Ø) 1ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
L% (1) Ø1ØØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
L% (2) ØØ1ØØØØØØØØØØØØØØØØØØØØØØØØØØØØ
usw.
L%(29) ØØØØØØØØØØØØØØØØØØØØØØØØØØ1ØØ
L%(3Ø) ØØØØØØØØØØØØØØØØØØØØØØØØØØØ1Ø
L%(31) ØØØØØØØØØØØØØØØØØØØØØØØØØØØØ1
```

Mit Lpoke S%, L%(0) wird somit ein Punkt in Spalte 0 und mit Lpoke S%, L%(31) ein Punkt in Spalte 31 gesetzt. S% (= Screen) steht dabei für die Bildschirmadresse der jeweiligen Punktzeile.

```
Rem Wandernder Punkt
S%=Xbios(2)+2Ø*8Ø !Zeile 2Ø
Dim L%(31)
L%(Ø)=-2↑31
Y%=31
For X%=1 To 31
 Dec Y%
 L%(X%)=2↑Y%
Next X%
Lpoke Xbios(2)+2Ø*8Ø+4,L%(Ø) !Spalte 32
For X%=Ø To 31    !Punktspalte Ø bis 31
 L$=Bin$(L%(X%))
 L=Len(L$)
 Print At(1,1);Spc(32-L);L$
 Lpoke S%,L%(X%) !Punkt setzen
 Pause 5Ø
Next X%
```

5. „Stehende Spaltenbytes“

Aus Abb. 3 können wir ersehen, daß sich eine horizontale Punktzeile beim Apple aus 40 Bytes (numeriert 0 bis 39) und beim Atari aus 80 Bytes (numeriert 0 bis 79) = 40 Wörtern (Words) = 20 Langwörtern (Long Words) zusammensetzt.
„Liegende Zeilenbytes“ lassen sich in BASIC oder Assembler leicht peeken. Der Drucker benötigt jedoch keine „liegenden Zeilenbytes“, sondern „stehende Spaltenbytes“, denn die Druckkopfnadeln sind vertikal angeordnet. „Stehende Spaltenbytes“ lassen sich jedoch weder in BASIC peeken noch in Assembler in ein Prozessorregister laden, sondern müssen vielmehr für einen 24-Nadeldrucker aus jeweils 24 Zeilenbytes einzeln extrahiert werden. Dies kostet jedoch sehr viel Prozessorzeit, denn eine Apple-Grafik umfaßt 280 x 192 = 53.760 Bits, und eine Atari-Grafik enthält sogar 256.000 Bits. Typischerweise entfallen deshalb über 90% der Laufzeit eines Grafik-Dump-Programms auf dieses „Herauspulen“ = Extrahieren der Spaltenbits. Wäre der Grafikbildschirm nicht zeilen-, sondern spaltenweise dergestalt organisiert, daß die 8 Bits eines Bytes 8 *senkrechte* Rasterpunkte repräsentieren würden, dann könnten zumindest 1:1-Grafik-Dump-Programme selbst in BASIC effizient geschrieben werden.

Wie gelangt man nun von „liegenden Zeilenbytes“ zu „stehenden Spaltenbytes“? Zunächst müssen wir wissen, daß ein 24-Nadel-Drucker bei einer parallelen Schnittstelle nicht 24 Bits gleichzeitig, sondern 3 x 8 Bits in Form von 3 Bytes *nacheinander* empfängt (1. Byte, 2. Byte, 3. Byte). In **Abb. 4a** sehen wir 3 liegende Bytes, die die 24 gesendeten Bits symbolisieren, während die 3 stehenden Bytes

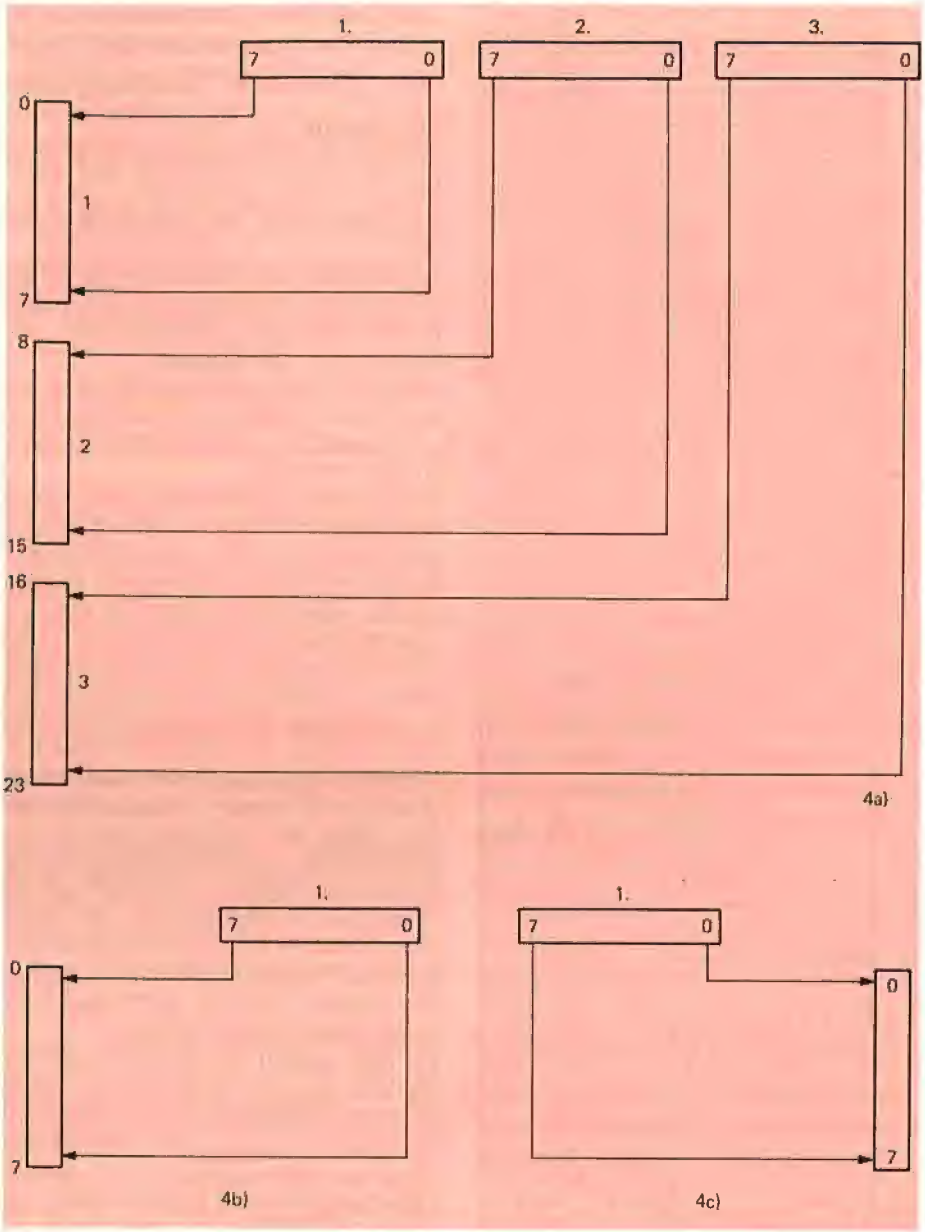


Abb. 4: Zuordnung der Bits der Sendebytes zu den Bits der Nadelspalte-Bytes beim LQ-800 (4a: 24 Nadeln), FX80 (4b: 8 Nadeln) und Imagewriter (4c: 8 Nadeln)

die 24 Nadeln repräsentieren. Bit 7 des 1. gesendeten Bytes steuert also Nadel 0, während Bit 0 des 3. gesendeten Bytes Nadel 23 aktiviert. Diese Zuordnung gilt für den LQ-800. Für den 8-Nadel-Drucker FX80 (vgl. **Abb. 4b**) gilt das gleiche, während beispielsweise beim 8-Nadel-Imagewriter (vgl. **Abb. 4c**) die Bit-Nadel-Zuordnung verdreht ist.

Bleiben wir zunächst aus Gründen der Vereinfachung bei einem 8-Nadel-Drucker vom Typ Epson FX80 und betrachten **Abb. 5a** in Verbindung mit Abb 4b. Um beim Apple mit dem FX80 Spalte 0 von Zeile 0 mit 8 vertikalen Nadeln zu drucken, muß aus 8 untereinanderliegenden Bildschirmzeilen das jeweilige Bit 0 „herausgepult“ und von rechts nach links in ein „liegendes Byte“ geschoben werden, das dann das an den Drucker zu sendende 1. Byte darstellt. Bit 0 von Byte 0 wird dann zum Bit 7 des 1. Bytes, Bit 0 von Byte 1 zum Bit 6 des 1. Bytes usw., bis schließlich Bit 0 von Byte 7 zum Bit 0 des 1. Bytes wird. Dieses dergestalt mit 8 Bits gefüllte 1. Byte (vgl. Abb. 5a) geht dann zum Drucker und steuert dort die Nadeln 0 bis 7 (vgl. Abb 4b).

Wenn wir statt eines 8-Nadel-Druckers den 24-Nadel-LQ-800 verwenden (vgl. **Abb. 5b**), so benötigen wird neben dem 1. Byte noch das 2. und 3. Byte, die nacheinander zum Drucker geschickt werden.

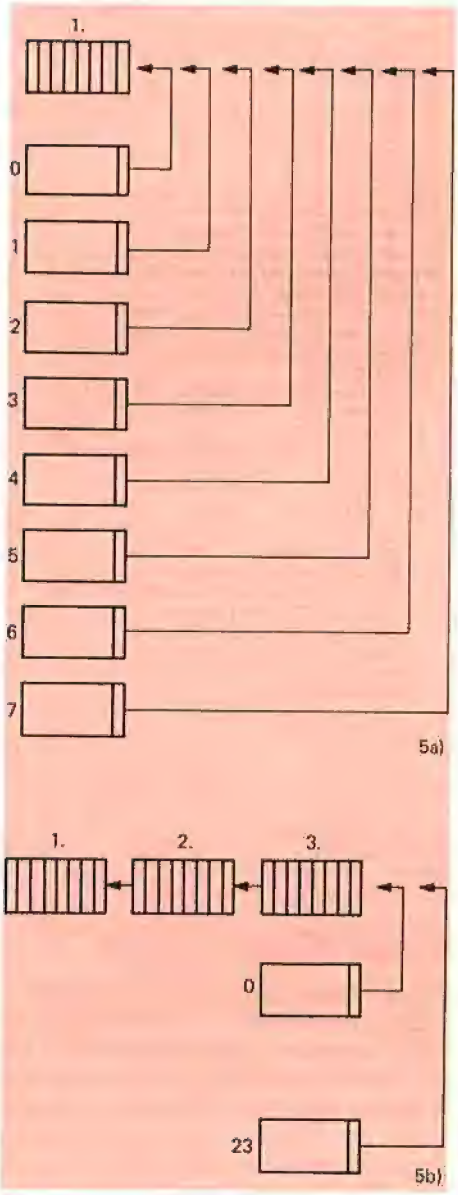
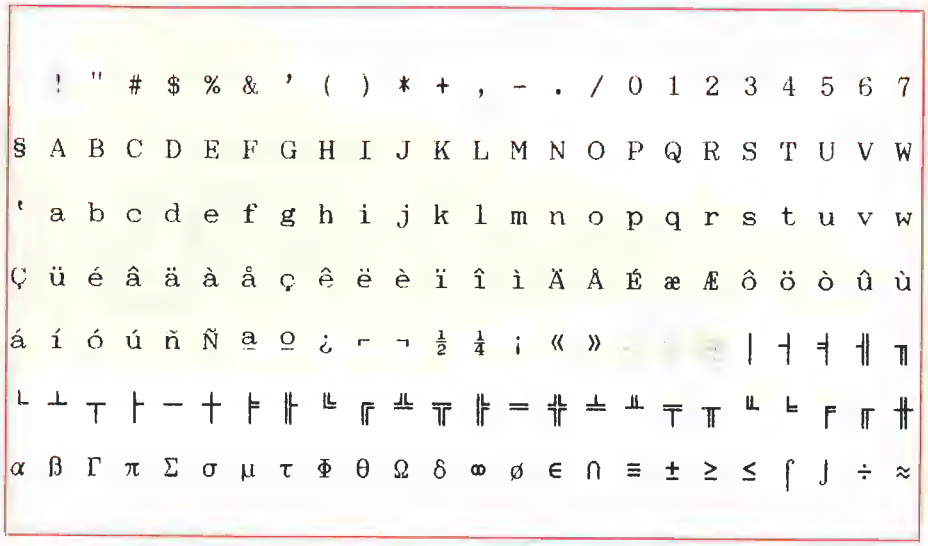


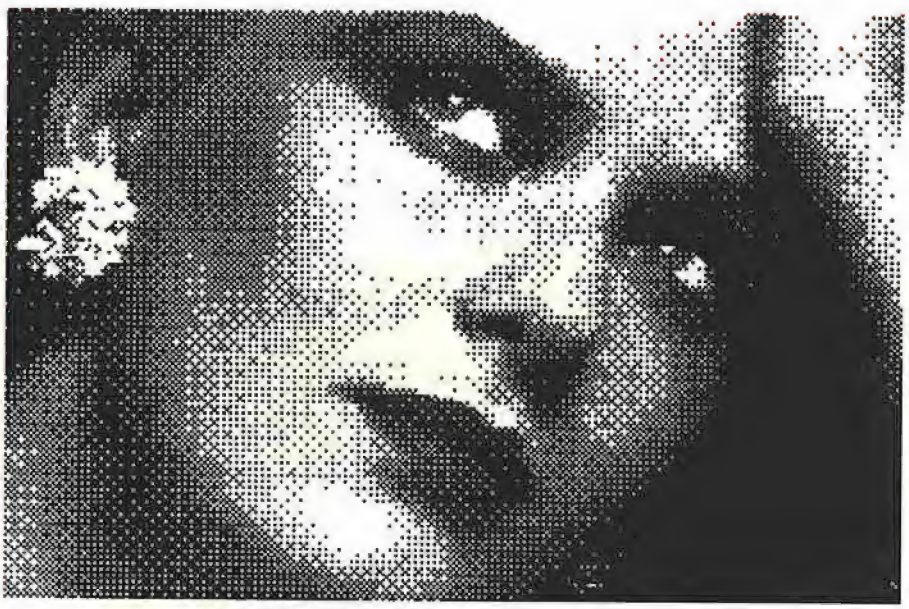
Abb. 5: Hineinschieben der Bits von 8 Grafikbildschirm-Zeilen in 1 Sendebyte (5a: FX80) oder der Bits von 24 Zeilen in 3 Sendebytes (5b: LQ-800) beim Apple. Beim Atari würde man mit dem höchsten statt mit dem niedrigsten Bit beginnen.

Entsprechend müssen wir die dann zum Füllen der 3 Bytes benötigten 24 Bits aus insgesamt 24 untereinanderliegenden Zeilenbytes „herauspulen".
Abb. 5a und 5b beziehen sich auf den Apple-Bildschirm. Beim Atari würden wir mit Bit 7 statt mit Bit 0 beginnen.

Zusammenfassung
Die Einser- und Nullerbits eines an den Drucker geschickten Bytes repräsentieren die zu aktivierenden bzw. nicht zu aktivierenden vertikalen Nadeln. Für 1 Nadel-



Ausdruck a: LQ-800-Zeichensatz 1:1



Ausdruck b: Apple-Grafik 3:1 auf LQ-800

Ausdruck c: Apple-Grafik 1:1 auf LQ-800

Bei einem Grafikausdruck ohne Grauwertverlust (Ausdrucke b und c) wird sofort die zu steile „Gradation" deutlich, d. h. die helleren Stellen („Lichter") sind zu weiß und die dunkleren Stellen („Tiefen") sind zu schwarz. Die Grafikausdrucke sehen deshalb so aus, als wären echte Halbtonvorlagen ohne Verwendung eines Rasters („Magenta-Kontaktrasters") reproduziert worden. Damit dies auch wirklich deutlich wird, wurden in unserer Reproanstalt ausdrücklich sog. Strichaufnahmen gemacht, während wir sonst Grafikausdrucke aufrastern lassen. Im Grunde lassen sich jedoch mit Stoffarbband hergestellte Epson-Ausdrucke weder gerastert noch ungerastert zufriedenstellend reproduzieren, weil die Reprovorlagen keine ausreichende Deckung aufweisen, wie dies etwa bei den in unserem grafischen Atelier hergestellten technischen Zeichnungen der Fall ist.

spalte benötigt ein 8-Nadel-Drucker 1 Byte und ein 24-Nadel-Drucker entsprechend 3 Bytes.
Ein Grafikspeicherbyte repräsentiert beim Apple 7 und beim Atari 8 horizontale Bildpunkte. Um ein „Nadelbyte" für den Matrixdrucker zu erzeugen, müssen folglich die entsprechenden Bits aus 8 Grafikpunktzeilen extrahiert werden. Dies geschieht in Assembler mittels Schiebe- und Rotierbefehlen.

6. Grafikumschaltung

Wenn wir ein Byte mit dem hexadezimalen Wert $55 (= binär %01010101 = dezimal 85) an den Drucker senden, so wird zu unserer Überraschung nicht jede zweite Nadel eines 8-Nadelkopfes aktiviert, sondern vielmehr der Buchstabe „U" ausgegeben, denn der ASCII-Code von „U" ist 85. Wir müssen also dem Drucker zunächst einmal mitteilen, daß wir jetzt Grafik und nicht normale Buchstaben ausdrukken wollen. Für den Atari gilt dann in Verbindung mit dem LQ-800 folgendes:
1. Da der vertikale Nadelabstand 1/180 Zoll beträgt und alle 24 Nadeln benutzt werden sollen (= Makrozeilen, s.o.), muß der Zeilenvorschub 24/180 Zoll betragen. Die ESC-Sequenz hierfür lautet:
ESC „3" 24
oder
Chr$(27) Chr$(51) Chr$(24)
Wir brauchen dann später nur noch nach jeder Makrozeile ein Wagenrücklauf-Zeilenvorschub-Kommando (CR-LF) an den Drucker zu senden.
2. Jede Makrozeile leiten wir zudem zwecks Umschaltung auf den Grafikmodus mit folgender ESC-Sequenz ein:
ESC „*" 39 128 2
oder
Chr$(27) Chr$(42) Chr$(39) Chr$(128) Chr$(2)
Die Zahl 39 steht für einen horizontalen Nadelabstand von 1/180 Zoll. Damit sind vertikaler und horizontaler Punktabstand identisch, und die Grafik wird folglich unverzerrt („ohne Zoom") ausgedruckt.
Die Zahlen 128 (Low Byte) und 2 (High Byte) stehen für 128 + 2 * 256 = 640. Dies ist die Anzahl der Punktspalten beim Atari-Bildschirm.
3. Nachdem die gesamte Grafik mit ihren 17 Makrozeilen ausgedruckt worden ist, schalten wird mit
ESC „2"
auf den normalen Text-Zeilenvorschub zurück (= 30/180 oder 1/6 Zoll). Ein Abstellen des Grafikmodus ist nicht erforderlich, da der LQ-800 automatisch auf Textmodus zurückschaltet, sobald er die für eine Makrozeile erforderlichen Bytes empfangen hat. Aus diesem Grunde muß deshalb auch jede weitere Makrozeile erneut mit der obigen ESC-Sequenz eingeleitet werden.

7. Grafikvergrößerung

Oft möchte man Grafiken nicht 1:1 ausdrucken, weil die Ausdrucke dann recht klein sind (beim Atari ca. 9cm Breite x 5,6cm Höhe, beim Apple entsprechend kleiner), denn der Punktabstand beträgt bekanntlich nur 1/180 Zoll = 0,14mm). Es bieten sich zwei Möglichkeiten an:

1. **Vergrößerung mit Grauwertverlust:** Rechts neben und unter jedem Punkt wird ein zusätzlicher „Leerpunkt" eingefügt. Dadurch wird der Ausdruck jedoch sehr grau, weil die Punktgröße unverändert bleibt. Beispiel:

```
.. -> .  .
..       .
```

2. **Vergrößerung ohne Grauwertverlust:** Jeder Punkt wird nach rechts und nach unten verdoppelt. Es entstehen dann gleichschwarze Ausdrucke wie beim 1:1-Grafik-Dump. Beispiel:

```
.. -> ....
..    ....
      ....
      ....
```

8. Atari-Beispielprogramm

Das folgende GFABASIC-Programm erzeugt einen 1:1-Grafikausdruck. Obwohl es seinen Zweck erfüllt, dient es nur zur Veranschaulichung des im Abschnitt 3 beschriebenen Algorithmus, denn der Ausdruck dauert auf dem LQ-800 ca. 3min., weil GFABASIC wie auch die anderen Hochsprachen über keinen Rotierbefehl verfügt. Die Simulation der Bit-Rotierung nimmt deshalb 90% der Laufzeit der Dump-Prozedur ein. Eine nähere Erläuterung des Programms ist an dieser Stelle nicht sinnvoll, da hierzu GFABASIC-Kenntnisse vorausgesetzt werden müßten.

(Ein 68000-Drucker-Dump-Programm finden Sie auf der Seite 36.)

HGRLQ.BAS

(1:1-Atari-LQ-800-Dump mit Demo)

```
Rem *** Atari-LQ-800-HGR-Dump ***
Color 1
Box 0,0,639,399
Deffill 1,2,1
Fill 1,1
Gosub Screendump
End
Rem ------------------------------
Procedure Screendump
 Local X%,Y%,S%
 S%=Xbios(2)
 Rem ------------------------
 Dim Longmaske%(31)
 Longmaske%(0)=-2^31
 Y%=31
 For X%=1 To 31
  Dec Y%
  Longmaske%(X%)=2^Y%
 Next X%
 Dim Long%(23)
 Rem ------------------------
 Pinit$=Chr$(27)+"3"+Chr$(24)
 Pline$=Chr$(27)+"*"+Chr$(39)
 Pline$=Pline$+Chr$(128)+Chr$(2)
 Pexit$=Chr$(27)+"2"
 Rem ------------------------
 Lprint Pinit$
 For Vpos1%=0 To 408-24 Step 24
  Lprint Pline$;
  For Hpos%=0 To 80-4 Step 4
   Vpos2%=Vpos1%
   For X%=0 To 23
    Long%(X%)=Lpeek(S%+Vpos2%*80+Hpos%)
    Inc Vpos2%
   Next X%
   For Y%=0 To 31
    Long%=0
    For X%=0 To 23
     Rem Die zwei folgenden Zeilen
     Rem simulieren den Rotierbefehl
     If Long%(X%) And Longmaske%(Y%)
      Add Long%,Longmaske%(X%)
     Endif
    Next X%
    Long$=Mkl$(Long%)
    Lprint Left$(Long$,3);
   Next Y%
  Next Hpos%
  Lprint
 Next Vpos1%
 Lprint Pexit$
 Erase Longmaske%()
 Erase Long%()
Return
```

9. Apple-Beispielprogramm

Für den Apple mußte das Druckprogramm LQ800.HGR1 (für 1:1-Grafik-Dump) in 6502-Assembler geschrieben werden. Es ist deshalb entsprechend schnell, und der Ausdruck wäre nach 3 Sekunden erledigt, wenn der LQ-800 genauso flott wäre. Auch hier verzichten wir auf eine Erläuterung des Programms, weil dazu Assemblerkenntnisse erforderlich sind. Im übrigen lehnt es sich exakt an den oben beschriebenen Algorithmus an.
Zusätzlich drucken wir einen Auszug aus dem Programm LQ800.HGR3 ab, das einen 3:1-Ausdruck ohne Grauwertverlust erzeugt.

HGR1.TEST

(Demo für 1:1-Apple-LQ-800-Dump)

```
10 PRINT CHR$ (4);"BLOAD LQ800.HGR1"
15 A = 24576: REM $6000
20 POKE A + 3,32: REM Page 1
25 POKE A + 4,1 * 16: REM Slot 1
30 POKE A + 5,0: REM Positiv
35 HGR : HCOLOR= 7
40 HPLOT 0,0 TO 279,0 TO 279,191
   TO 0,191 TO 0,0
45 HPLOT 0,0 TO 279,191
50 HPLOT 0,191 TO 279,0
55 REM 4 Modes testen
60 PRINT CHR$ (4);"PR#1"
65 FOR X = 1 TO 4: READ Y
70 POKE A + 6,Y: CALL A
75 PRINT : NEXT X
80 PRINT CHR$ (4);"PR#0"
85 TEXT
90 DATA 32,33,38,39
```

LQ800.HGR1

(1:1-Apple-LQ-800-Dump)

```
BSAVE LQ800.HGR1, A$6000, L210

1             ORG  $6000
3   *
4   * 1:1-Ausdruck: LQ800.HGR1
5   *
6             JMP  START
7   *-------------------------------
8   PAGE      HEX  20     ;20=Pg 2
9   SLOT      HEX  10     ;Slot*16
10  XOR       HEX  00     ;FF=negativ
11  MODE      HEX  27     ;20/21/26/27
12  *-------------------------------
13  PUFFER    EQU  $0280      ;24 Bytes
14  HBASL     EQU  $00CE
15  HBASH     EQU  $00CF
16  BYTE3     EQU  $0019
17  BYTE2     EQU  $001A
18  BYTE1     EQU  $001B
19  VPOS1     EQU  $00FA
20  VPOS2     EQU  $00FB
21  HPOS      EQU  $00FC
22  AREG      EQU  $00FD
23  YREG      EQU  $00FE
24  *
25  START     LDA  #0
26            STA  VPOS1
27            LDA  MODE
28            STA  M
29  *-------------------------------
30  LOOP1     LDY  #0         ;Y=0-39
31            STY  HPOS
32  *
33            LDX  #0
34  ESCPRT1   LDA  ESC1,X
35            JSR  PRINT
36            INX
37            CPX  #ESC2-ESC1
38            BNE  ESCPRT1
39  *
40  LOOP2     LDX  #0         ;X=0-23
41            LDA  VPOS1
42            STA  VPOS2
43  *
44  * 24 Vertikalbytes 0-23
45  *
46  LOOP3     LDA  VPOS2      ;$F411f.
47            AND  #%11000000
48            STA  HBASL
49            LSR
50            LSR
51            ORA  HBASL
52            STA  HBASL
53            LDA  VPOS2
54            STA  HBASH
55            ASL
56            ASL
57            ASL
58            ROL  HBASH
59            ASL
60            ROL  HBASH
61            ASL
62            ROR  HBASL
63            LDA  HBASH
64            AND  #%00011111
65            ORA  PAGE
66            STA  HBASH
67  *
68            LDY  HPOS
69            LDA  (HBASL),Y
70            EOR  XOR        ;invertieren
71            STA  PUFFER,X
72            INC  VPOS2
73            INX
74            CPX  #24
75            BNE  LOOP3      ;--->
76  *
77  * 7mal 24 senkrechte Bits drucken
78  *
79  DRUCK1    LDY  #7         ;7 Bits
80  DRUCK2    LDX  #0
81  DRUCK3    LSR  PUFFER,X
82            ROL  BYTE3
83            ROL  BYTE2
84            ROL  BYTE1
85            INX
86            CPX  #24        ;24 Bytes
87            BNE  DRUCK3
88            LDX  #2
89  DRUCK4    LDA  BYTE3,X
90            JSR  PRINT
91            DEX
92            BPL  DRUCK4
93            DEY
94            BNE  DRUCK2
95  *
96  * 40 Horizontalbytes 0-39
97  *
98            INC  HPOS
99            LDY  HPOS
100           CPY  #40
101           BNE  LOOP2      ;--->
102 *
103           LDX  #0
104 ESCPRT2   LDA  ESC2,X
105           JSR  PRINT
106           INX
107           CPX  #ESC3-ESC2
108           BNE  ESCPRT2
109 *
110 * Alle 192 Zeilen erledigt?
111 *
112           CLC
113           LDA  VPOS1
114           ADC  #24
115           STA  VPOS1
116           CMP  #191
117           BCS  ENDE
118           JMP  LOOP1      ;--->
119 ENDE      RTS
120 *-------------------------------
121 * Esc-3 n/180
122 ESC1      HEX  1B33       ;Esc 3
123           HEX  18         ;24/180
124 * Esc-* m ll hh
125           HEX  1B2A       ;Esc *
126 M         HEX  27         ;m
127           DA   280        ;ll hh
128 * CR-LF Esc-2
129 ESC2      HEX  0D0A       ;CR-LF
130           HEX  1B32       ;Esc 2
131 ESC3      NOP
132 *-------------------------------
133 * Zeichen im Akkumulator senden
134 PRINT     STA  AREG
135           STY  YREG
136           LDY  SLOT
137 OFFLINE?  LDA  $C081,Y
138           ASL
139           ASL
140           BPL  OFFEXIT
141 READY?    LDA  $C081,Y
142           AND  #$08
143           BNE  READY?
144           LDA  AREG
145 SEND      STA  $C081,Y
146           STA  $C082,Y
147           STA  $C084,Y
148 OFFEXIT   LDY  YREG
149           LDA  AREG
150           RTS
```

LQ800.HGR3

(3:1-Apple-LQ-800-Dump, Auszug des Listings)

```
1             ORG  $6000
3   *
4   * 3:1-Ausdruck: LQ800.HGR3
    *
77  * 3fach dehnen nach rechts + unten
78  DEHNEN1   LDY  #7         ;7 Bits
79  DEHNEN2   LDX  #0
80  DEHNEN3   LSR  PUFFER,X
81            JSR  VERTIKAL
82            JSR  VERTIKAL
83            JSR  VERTIKAL
84            INX
85            CPX  #8         ;8 Bytes
86            BNE  DEHNEN3
87            JSR  HORIZONT
88            JSR  HORIZONT
89            JSR  HORIZONT
90            DEY
91            BNE  DEHNEN2
92            BEQ  WEITER     ;im Listing
93  *
94  VERTIKAL  PHP
95            ROL  BYTE3      ;3fach
96            ROL  BYTE2      ;vertik.
97            ROL  BYTE1      ;dehnen
98            PLP
99            RTS
100 *
101 HORIZONT   LDX  #2         ;3fach
102 HORIZ1    LDA  BYTE3,X    ;horiz.
103           JSR  PRINT      ;drucken
104           DEX
105           BPL  HORIZ1
106           RTS
```

Kurzhinweise für die Apple-Sammeldisk

Die Sammeldisk enthält die Dateien
HGR1.TEST (Demo-Programm)
T.LQ800.HGR1 (Big-Mac-Quelltext)
LQ800.HGR1 (Objektcode)
für 1:1-Dump ohne Grauwertverlust

HGR3.TEST (Demo-Programm)
T.LQ800.HGR3 (Big-Mac-Quelltext)
LQ800.HGR3 (Objektcode)
für 3:1-Dump ohne Grauwertverlust (Q = Quick)

HGR3Q.TEST (Demo-Programm)
T.LQ800.HGR3Q (Big-Mac-Quelltext)
LQ800.HGR3Q (Objektcode)
für 3:1-Dump mit Grauwertverlust

TEXT.TEST (Demo-Programm)
T.TEXT.LQ800 (Big-Mac-Quelltext)
TEXT.LQ800 (Objektcode)
für den Zeichensatz-Dump, der in diesem Beitrag abgebildet worden ist.

Man beachte, daß die Druckroutinen auf die Parallel-Interface-Karte der Firma Nippon Express, Hamburg, abgestimmt sind, die mit der alten Orange-Grappler- bzw. Star-Micromix-Graphstar-Karte funktionsgleich ist. Für andere Parallel-Karten sind in LQ800.HGR1 (Zeile 134-150) usw. entsprechende Anpassungen erforderlich. Wir haben diese Karte hier deshalb ausgewählt, weil sie beispielsweise bei dem Superdump-Programm keine Berücksichtigung fand und dort entsprechend eingefügt werden kann.

LQG.ASM

```
                        PAGELEN   1000          ;KEIN FF
                        SECTION   ONE           ;FUER GST
;
; LQG.ASM = LQ800-Screendump 1:2
; ==============================
; GST-Assembler; us/1.12.86
;
; LQG.ASM -LIST LQG.LST -BIN LQG.PRG -NOLINK
;
; D0 = Screen-Basis
; D1 = Vpos (0-408)
; D2 = Vpos-Temporary
; Shift-Temporary           Rem LQG.ASM-Test für GFABASIC
; D3 = Hpos  (0-80)         Rem Vorher Testbild erzeugen
; D4 = Koordinate (vgl. A1) Open "I",#1,"LQG.PRG"
;      Sendebyte            L=Lof(#1)
; D5 = Zaehler              L$=Input$(L,#1)
; D6 = Zaehler              Close
; D7 = Shift-Akkumulator    L=Instr(L$,"NqNqNq") ! 3 NOPs
; A0 = Puffer-Anfang        L$=Mid$(L$,L)
;      Esc-Anfang           L=Varptr(L$)
; A1 = Koordinate (vgl. D4) Call L
;
000 4E71                NOP                     ;SIGNATUR
002 4E71                NOP
004 4E71                NOP
                ;
                ; REGISTER RETTEN UND DRUCKER-STATUS PRUEFEN
006 48E7FFC0    START:  MOVEM.L  D0-D7/A0-A1,-(SP)
00A 3F3C0000            MOVE.W   #0,-(SP)       ;0=PARALLEL
00E 3F3C0008            MOVE.W   #8,-(SP)       ;8=BCOSTAT
012 4E4D                TRAP     #13            ;BIOS
014 588F                ADDQ.L   #4,SP          ;STACKKORREKT.
016 4A80                TST.L    D0             ;PRINTER READY
018 6A0000A2            BPL.L    ENDE           ;0=NEIN, -1=JA
                ;
                ; BILDSCHIRM-ANFANGSADRESSE ---> D0
                ;
01C 3F3C0002            MOVE.W   #2,-(SP)       ;D0=SCREENBASE
020 4E4E                TRAP     #14            ;XBIOS
022 548F                ADDQ.L   #2,SP          ;STACKKORREKT.
                ;
                ; DRUCKERPUFFER LEEREN
                ;
024 183C000D            MOVE.B   #$0D,D4        ;RETURN
028 61000098            BSR.L    PRINT
02C 183C000A            MOVE.B   #$0A,D4        ;LINEFEED
030 61000090            BSR.L    PRINT
                ;
                ; INIT MAKROZEILE (= 12 MIKROZEILEN)
                ;
034 7200                MOVEQ    #0,D1          ;VPOS=0
                ;
                ; INIT MAKROSPALTE (= 32 MIKROSPALTEN)
                ;
036 7600        LOOP1:  MOVEQ    #0,D3          ;HPOS=0
                ;
                ; GRAFIKMODUS EINSCHALTEN
                ;
038 41FA00D6            LEA      ESC1S(PC),A0   ;RELOKATIV
03C 7A07                MOVEQ    #8-1,D5        ;8-ZAEHLER
03E 1818        ESC1:   MOVE.B   (A0)+,D4       ;D4=ESC-BYTE
040 61000080            BSR.L    PRINT
044 51CDFFF8            DBF      D5,ESC1        ;->
                ;
                ; MAKROSPALTENABSCHNITT
                ;
048 2401        LOOP2:  MOVE.L   D1,D2          ;D2=VPOS-TEMP
04A 7A0B                MOVEQ    #12-1,D5       ;12-ZAEHLER
04C 41FA0092            LEA      PUF1(PC),A0    ;RELOKATIV
                ;
                ; KOORDINATE = VPOS * 80 + HPOS + SCREENBASIS
                ;
050 7850        LOOP3:  MOVE.L   #80,D4         ;D4=80
052 C8C2                MULU     D2,D4          ;D4=D2*80
054 D883                ADD.L    D3,D4          ;D4=D4+HPOS
056 D880                ADD.L    D0,D4          ;D4=D4+BASIS
058 2244                MOVE.L   D4,A1          ;A1=KOORDINATE
05A 20D1                MOVE.L   (A1),(A0)+     ;(A1) IN PUF1
05C 5242                ADDQ.W   #1,D2          ;VPOS+1
05E 51CDFFF0            DBF      D5,LOOP3       ;---->
                ;
                ; SENDEBYTES FUER DRUCK AUFBEREITEN
                ;
062 7C1F                MOVEQ    #32-1,D6       ;32-ZAEHLER
064 41FA007A    DRUCK1: LEA      PUF1(PC),A0    ;RELOKATIV
068 7A0B                MOVEQ    #12-1,D5       ;12-ZAEHLER
                ;
                ; ZWEIMAL VERTIKAL DEHNEN
                ;
06A 2410        DRUCK2: MOVE.L   (A0),D2        ;LONG-LOAD
06C E392                ROXL.L   #1,D2          ;SHIFTEN
06E 2082                MOVE.L   D2,(A0)        ;LONG-SAVE
070 40E7                MOVE.W   SR,-(SP)       ;EXTEND-SAVE
072 E397                ROXL.L   #1,D7          ;1. SHIFT
074 44DF                MOVE.W   (SP)+,CCR      ;EXTEND-LOAD
076 E397                ROXL.L   #1,D7          ;2. SHIFT
078 5888                ADD.L    #4,A0          ;ZEIGER+4
07A 51CDFFEE            DBF      D5,DRUCK2      ;->
                ;
                ; ZWEIMAL HORIZONTAL DRUCKEN
                ;
07E 7A01                MOVEQ    #2-1,D5        ;2-ZAEHLER
080 2807        DRUCK3: MOVE.L   D7,D4          ;--XX----
082 E08C                LSR.L    #8,D4
084 E08C                LSR.L    #8,D4
086 613A                BSR.S    PRINT
088 2807                MOVE.L   D7,D4          ;----XX--
08A E08C                LSR.L    #8,D4
08C 6134                BSR.S    PRINT
08E 2807                MOVE.L   D7,D4          ;------XX
090 6130                BSR.S    PRINT
092 51CDFFEC            DBF      D5,DRUCK3      ;->
096 51CEFFCC            DBF      D6,DRUCK1      ;--->
                ;
                ; MAKROZEILE FERTIG?
                ;
09A 5803                ADDQ.B   #4,D3          ;HPOS+4
09C 0C030050            CMP.B    #80,D3         ;BEREITS 80?
0A0 66A6                BNE.S    LOOP2          ;----->
                ;
                ; CR-LF SENDEN UND GRAFIK ABSTELLEN
                ;
0A2 41FA0074            LEA      ESC2S(PC),A0   ;RELOKATIV
0A6 7A03                MOVEQ    #4-1,D5        ;4-ZAEHLER
0A8 1818        ESC2:   MOVE.B   (A0)+,D4       ;D4=ESC-BYTE
0AA 6116                BSR.S    PRINT
0AC 51CDFFFA            DBF      D5,ESC2        ;->
                ;
                ; LETZTE MAKROZEILE?
                ;
                ; (MIKROZEILEN 400-407 IMMER LEER???)
                ;
0B0 0641000C            ADD.W    #12,D1         ;VPOS+12
0B4 0C410198            CMP.W    #408,D1        ;BEREITS 408?
0B8 6600FF7C            BNE      LOOP1          ;----->
                ;
                ; REGISTER LADEN UND ENDE
                ;
0BC 4CDF03FF    ENDE:   MOVEM.L  (SP)+,D0-D7/A0-A1
0C0 4E75                RTS
                ;
                ; ALS STAND-ALONE-PROGRAMM STATT RTS
                ;
                ;       MOVE.W   #0,-(SP)       ;0=TERMINATE
                ;       TRAP     #1             ;GEMDOS
                ;
                ; BYTE AN DRUCKER SENDEN
                ;
0C2 48E7FFC0    PRINT:  MOVEM.L  D0-D7/A0-A1,-(SP)
0C6 0284000000FF        AND.L    #$000000FF,D4  ;NUR LOW-BYTE
0CC 3F04                MOVE.W   D4,-(SP)       ;SENDEBYTE
0CE 3F3C0000            MOVE.W   #0,-(SP)       ;0=PARALLEL
0D2 3F3C0003            MOVE.W   #3,-(SP)       ;3=BCONOUT
0D6 4E4D                TRAP     #13            ;BIOS
0D8 5C8F                ADDQ.L   #6,SP          ;STACKKORREKT.
0DA 4CDF03FF            MOVEM.L  (SP)+,D0-D7/A0-A1
0DE 4E75                RTS
                ;
0E0 00000030    PUF1:   DS.L     12             ;12*32 BITS
                ;
                ; ESC-SEQUENZ VORHER (8 BYTES)
                ;
0E4 1B          ESC1S:  DC.B     $1B            ;ESC
0E5 33                  DC.B     $33            ;"3"
0E6 18                  DC.B     $18            ;24/180
0E7 1B                  DC.B     $1B            ;ESC
0E8 2A                  DC.B     $2A            ;"*"
0E9 27                  DC.B     $27            ;MODUS 39
0EA 00                  DC.B     $00            ;LL 1280
0EB 05                  DC.B     $05            ;HH 1280
                ;
                ; ESC-SEQUENZ NACHHER (4 BYTES)
                ;
0EC 0D          ESC2S:  DC.B     $0D            ;RETURN
0ED 0A                  DC.B     $0A            ;LF
0EE 1B                  DC.B     $1B            ;ESC
0EF 32                  DC.B     $32            ;"2"
                        END
```

BUCHHALTUNG mit BUCH

Doppelte Buchführung, die auf Selbständige und Kleinunternehmer zugeschnitten ist:

- 14 Tage frei zur Ansicht!
- einfache und sichere Benutzerführung
- Prüfsiegel »anwenderfreundliche Software«
- Betriebsübersicht, G&V Rechnung, Debitoren und Creditoren, Journal, Kontenblätter, Saldenliste
- automatische Umsatzsteuerverbuchung
- das Programm hat eine Lohnsteueraußenprüfung des Finanzamtes in unserem Hause bestanden

»Ein leistungsfähiges, leicht zu bedienendes Werkzeug...« (Testbericht PC-Soft, 7/85)
»...halten wir dieses Finanzbuchhaltungsprogramm für ganz ausgezeichnet.« (Softwaretestjahrbuch '85)

660,— DM für Apple II + IIe/IIc und IBM-PC/XT/AT, Commodore PC 10 etc.

Informationen bei

RÖNTGEN SOFTWARE
Simpert-Krämer-Str. 44,
8909 Edelstetten,
Tel. 08283/1463

Erster aaa „little big" Ableger

für Apple II/+/e, Basis 108 und compatible

Sicher. – Wir haben uns bei der Entwicklung u. Fertigstellung unserer **Mega Ramcard II**® etwas Zeit gelassen, damit Sie jetzt auch Zeit gewinnen. Und das hat sich gelohnt!
Die Tests, aber auch die mitgelieferten Ramdisktreiber-Programme für folgende Betriebssysteme, beweisen es:

- Apple DOS 3.3
- Diversi DOS 2-C/4-C
- Apple ProDOS 1.0.1, 1.1.1
- Apple Pascal 1.1
  Apple Pascal 1.2/64 K, 1.2/128 K
  Apple Pascal 1.3/64 K, 1.3/128 K
- CP/M 2.2 für
  Microsoft Softcard, CP/M 2.20, 2.23
  Microsoft premium card, CP/M 2.26
  Microsoft Softcard II, CP/M 2.28

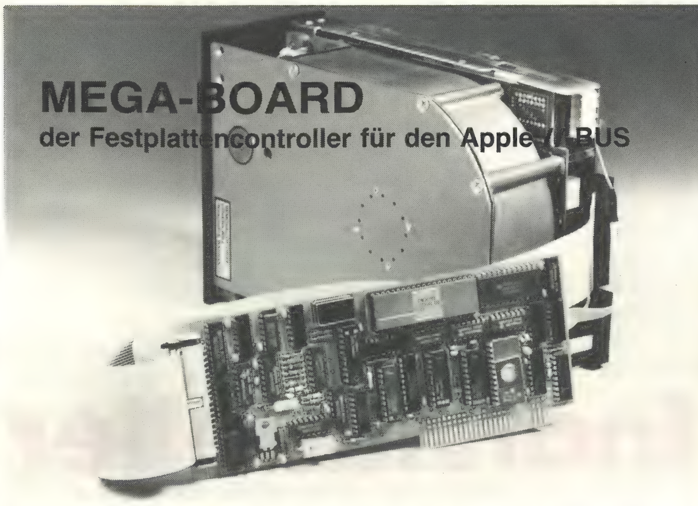
Sogar nach einem „Systemabsturz" oder Betriebssystemwechsel bleiben alle Dateien erhalten!
Peripherieslot und Speicherkapazität werden automatisch erkannt, sodaß das lästige Eingeben entfällt.
Mega Ramcard II® ist mit den erphi-AFDC2/AFDC3-Controller vollständig kompatibel, u.v.m.

Händleranfragen schriftlich erwünscht.

Mega Ramcard II, 1 MByte bestückt + Software (s.o.) + deutsches Handbuch, II/+/e	878,–
Mega Ramcard II, 256 K-RAM best. (bis 1 MB aufr.) + Software (s.o.) + dt. Hb., II/+/e	498,–
Apple-Works Anpassung	69,–
Disk II-Siemens-Laufwerk im Geh.+ Kabel f. orig. + comp. Contr. geeignet, II/+/e	298,–
Disk II-Siemens-Laufw.+ Contr.+ Kabel, II/+/e	359,–
Erphi-AFDC2-Controller + Autopatch Software + deutsches Handbuch, II/+/e	198,–
Disk II-Philips Laufwerk o. Geh., 2 x 80 Track-640 KB f. AFDC2 o. 3 modifiziert, II/+/e	298,–
Erphi-DuoDisk 1,2 MByte im Gehäuse + Erphi-AFDC3-Controller, II/+/e	898,–

REPARATUREN an Apple + Compatiblen Geräten + Zubehör führt unser Spezialteam garantiert zuverlässig + besonders kostengünstig aus.. **Sprechen Sie mit uns. Kostenvoranschlag auf Wunsch!**

aaa electronic gmbh
Telex 0772642 aaa-d
Habsburgerstraße 134
7800 FREIBURG, Tel. (0761) 276864
Bauelemente – Bausätze – µP's
Meßgeräte – Zubehör – Fachliteratur
Fachgeschäft für Elektronik + Mikrocomputer



Nicht jeder kann und möchte an seinen Apple //-Rechner unser MEGA-CORE oder die MDB 10/20 anschließen. Gerade für diesen Personenkreis bietet sich das MEGA-BOARD als der alternative und preiswerte Einstieg zum komfortablen Festplattenbetrieb an. Das Manual führt Sie zielsicher zu Ihren Wünschen:

- Festplattenbetrieb von 5-64 MBytes
- Betriebssystembereiche frei wählbar
- Booten von der Festplatte
- Betriebssysteme DOS, MS-DOS, CP/M, UCSD-Pascal, ProDOS menuegesteuert im Zugriff.

Zum Lieferumfang gehört der Controller MEGA-BOARD, alle Kabel mit Steckern, Installationssoftware, das ausführliche Manual und auf Wunsch die Festplatte zum Tagespreis.

Ein Produkt von:

FRANK & BRITTING
Elektronik Entwicklungs GmbH
Lange Straße 4, 7529 Forst
Telefon: 07251/103068-69
Telex: 7822452 fub d

Die Harddiskcontroller-Spezialisten

Ausgabe und Eingabe mit TYPETERM®

im Slot Ihres

APPLE II/IIe/IIgs

Das bedeutet: Computertextverarbeitung von der Schreibmaschinentastatur! Steckerfertig ohne Umbau.

Die neue CE-550! mit TYPETERM DM 1.318,–

TYPETERM-Interface DM 399,–
für alle BROTHER-Typenradschreibmaschinen ab AX-30 bis EM-811
(auch für Vorgängermodelle!)
Paketpreis z. B.:

EM-501 mit TYPETERM	DM 2056,–
EM-511 mit TYPETERM	DM 2332,–
EM-701 mit TYPETERM	DM 2388,–

TYPETERM – ein starkes Interface für starke Maschinen! Alle Cursor- und Ctl-Befehle. 4k ROM auf der Karte für DOS, PRODOS, CP/M, PASCAL. 2 Zeichensätze verfügbar z. B. deutsch u. ASCII. Alle Features: Hoch-/Tiefstellen, autom. Unterstreichen, var. Zeichen und Zeilenabst., autom. Papierzuführung usw.
TYPETERM – ein Produkt von

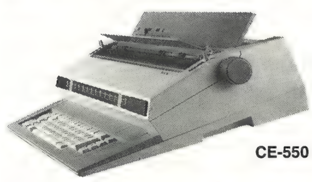
interkom electronic
Kock & Mreches GmbH
Postf., 3004 Isernhagen 4
Telefon 05139-87393

Ausgabe mit TYPETERM® JUNIOR

im Slot Ihres

APPLE II/IIe/IIgs

Paketpreis jetzt **DM 799,–!**
Schreibmaschine AX-10 mit Interface TYPETERM JUNIOR, steckfertig.



CE-550

brother
Die Zukunft heute

TYPETERM JUNIOR mit AX-10 – unser besonders günstiges Gespann, ebenfalls steckfertig. Mit TYPETERM JUNIOR kann die AX-10 mehr. Sie wird zum vollwertigen Typenraddrucker für Ihren Apple:

- 3 verschiedene Schriftstärken
- Automatisches Unterstreichen
- 2 Zeichensätze z.B. deutsch u. ASCII
- 2 Zeichenabstände
- 2k ROM auf der Karte für Ausgabe unter DOS, PRODOS, CP/M u. PASCAL.

TYPETERM JUNIOR – ein Produkt von

interkom electronic
Kock & Mreches GmbH
Postf., 3004 Isernhagen 4
Telefon 05139-87393

aaa electronic gmbh

Neue Preise # 2

NEU: Jetzt mit aaa-Computer-Kredit und bequemen monatlichen Raten! — LEER/BESTÜCKT

Industrie-Standard IBM-PC/XT+AT-compatible COMPUTER+Zubehör in Industrie-Qualität (X=XT, A=AT, X/A=XT+AT)

Mainboard XT-256, 8088 CPU, ØK (bis 256K-on board, oder 640 K mit Addr. auf Karte aufr.) + BIOS + 8 Slots, geprüft	79,––/227,––
Mainboard XT-256 mit 256 K-RAM, sonst. w.o., gepr.	317,––
Mainboard XT-640, 8088 CPU, ØK (bis 640 K-on board) gepr.	89,––/237,––
Mainboard XT-640 w.o. jedoch 256 K-RAM best. + gepr.	297,––
Mainboard XT-640 w.o. jedoch 640 K – vollbest. und geprüft	407,––
Mainboard TURBO XT-640 -4,7/8 MHz, ØK, best. + gepr.	99,––/276,––
Mainboard AT-1MB, 6/8 MHz, 80286 CPU, 16/24 bit (mit Sockel f. 80287) 512K-RAM best.(bis1MB aufrüstb.)+BIOS+Uhr m.Akku+8 Slots,gepr.	**1229,––**
64KB-RAM-Chip-Aufrüstsatz (9 Stück) einzeln geprüft! **X/A**	32,––
256 KB-RAM-Chip-Aufrüstung (9 Stück) einzeln geprüft! **X/A**	69,––
Disk-Controller gepr. (f. 2x360 KB-Drives), **X**	77,––
Disk-Controller gepr. (f. 4x360 KB-Drives)(Extra Kabel=39,–), **X**	34,––/94,––
Disk-Controller gepr. (f. 2x1,2 MB-Drives), **X**	229,––
Disk-Controller gepr. (f. 2x1,2 MB-Drives, auch f. 360 K geeignet), **A**	169,––
Harddisk-Controller f. 2xHD, **X**	287,––
Harddisk+Floppy-Controller (1 HD+2DD) gepr., **A**	597,––
Monochrome Karte geprüft, **X/A**	49,––/149,––
Monochrome Graph.+par.Print. Karte (Hercules comp.), **X/A**	55,––/168,––
Color RGB + Video Graphikkarte+Lightpen-Ausg., gepr., **X/A**	55,––/126,––
Parallel Printer-Karte gepr. o. Kabel (Extra Kabel = 49,–), **X/A**	29,––/ 49,––
RS 232 Interface (2xRS 232, 1 davon bestückt), **X/A**	29,––/ 73,––
RS 232+par. Printer Karte gepr., **X/A**	138,––
512K-RAM-Karte gepr. (ØKB)(mit 64 KB-RAM bestückbar), **X**	55,––/ 88,––
512K-RAM-Karte gepr. (ØKB)(mit 256 KB-RAM bestückbar-kurz), **X**	114,––
3 MB-RAM-Erweiterungsk. gepr. (ØK), (m.256 K-RAM bestückbar), **A**	317,––
2,5 MB-Multifunktionskarte, ØK, +RS 232,+par. Printer, gepr., **A**	397,––
384K-Multifunktionskarte, RS 232, par. Printer. Interf., Game/Joystick Port, Real Time Clock gepuff.,ØK+RAM-Disk/Spooler/Uhr Softw., **X**	55,––/189,––
Disk Multi I/O Karte (f. 2x360 K-Drive, 2xRS 232, 1xpar. Printer Int., Game/Joystick Port, Real Time Clock/Kalender gepuff.), **X**	59,––/169,––
Clock Karte gepuffert, geprüft, **X**	69,––
I/O Plus-II-Karte, 2xRS 232, (1xbest.), par. Printer-Interf., Game/Joystick Port, Real Time Clock gepuffert, geprüft, **X**	179,––
I/O Plus-Karte, 2xRS 232 (1xbest.), par. Printer-Interf., Game/Joystick Port, geprüft, **A**	159,––
EGA-comp. Enhanced Graphic Adapter Karte, mit 256 K-RAM-Bildschirmspeicher, Monochrome-Color Graphik (640x350 Punkte – 16 Farben)+par. Printer Int., voll best.+gepr., **X/A**	69,––/495,––
AD/DA-Wandler, 12 bit, 16/1, **X/A**	295,––
AD/DA-Wandler, 14 bit, 16/2, **X/A**	475,––
Option Board compatible Karte incl. Disk+Manual, (zum sichern kopiergeschützter Disketten, Kopierschutz wird mitübertragen), **X/A**	347,––
Eprom Writer, (f. 4 St. 2716/32/64/128/256/512 gleichzeitig), **X/A**	429,––
Prototype/Lochraster Leerkarte, durchkontaktiert, lang	64,––
Profi-Funktionstastatur ASCII + 15er Block ab **195,––**	/dt. ab 225,––
135W/150W-Netzteil mit eingeb. Ventilator, kurzschlußf., **X**	195,––/215,––
200W-Netzteil mit eing. Ventilator, kurzschlußfest, **A**	375,––
360KB-Diskdrive Slimline (Track Access Time <4 mS), R/W gepr., **X/A**	277,––
720KB-Diskdrive Slimline, 2x80 Tr. (TAT <2 mS), R/W gepr., **X/A**	337,––
1,2MB-Diskdrive Slimline, 2x80 Tr. (TAT <2 mS), R/W gepr., **X/A**	347,––
21 MB-NEC-Harddisk, X/A 979,–– / **Micro Science, X/A**	849,––
21 MB-NEC-HD+HD-Contr., X 1.285,–– / **MS+HD-Contr., X**	1.155,––
44 MB-NEC-Hardisk, X/A 1.957,–– / **44 MB-NEC+DTC, X**	2.349,––
21 MB, Harddisk+Controller in einem, anschlußfertig, **X**	1.298,––
Joystick f. IBM + Compatible, X/A	58,––
Maus incl. Software (MOUSE Systems comp.), **X/A**	158,––
Met.-Gehäuse aufkl. (Pl.f.4-Einb.-Drives), solide Ausf., X=198,––/	A=298,––

+++PC/XT+AT – KOMPLETT-SYSTEME AUFGEBAUT UND GEPRÜFT+++

XT-640, 8088 CPU (incl. Boot-Eprom) 256K-Ram incl. Lautsprecher, im IBM-look-like Met.-Gehäuse+Profi-Fkt. Tast.+135 W-Netzt. gepr.	777,––
XT-640, w.o.+Contr.+360KB Drive+Color RGB+Vid. Graphik-Karte	987,––
XT-640, w.o. jedoch statt Color mit Monochrome Karte	997,––
XT-640, w.o. jedoch mit Monochrome-Graphik-Printer-Karte	1.017,––
XT-640, 256 K-RAM+360 KB Drive+Disk Multi I/O+Color-Gr.-Karte	1.097,––
XT-640, 256 K-RAM+360 KB Drive+Color RGB+Video-Graphik-Karte+21 MB-Festplatte+Controller	**2.097,––**
AT-1MB, 6/8 MHz, 80286 CPU, 512 K-RAM, im IBM-look-like Met.-Gehäuse+Profi-Fkt.Tast+200W-Netzt.+Color Graph.-Karte+1,2MB-Drive, gepr.	**2.295,––**
AT-1MB, w.o.+21MB-Harddisk gepr.	3.785,––
Aufpreis für Turbo XT-640 -4,7/8 MHZ umschaltbares Board, **X**	39,––
Aufpreis für XT-640 M-board mit 640 K von 256 auf 640 K-RAM, **X**	119,––
Aufpreis f. EGA anstatt Color Graph.-Karte, **X/A**	475,––
MS-DOS 3.1 Diskette incl. Manual, **X/A**	159,––
MS-DOS+GW-Basic 3.2 sowie 4 dt. orig. Microsoft Hb. (Aufpreis in Verbindung mit Systemkauf), **X/A**	**189,––**
8087-Arithmetik-Prozessor, 5 MHz, X=327,–– / 80287, **A**	869,––
V 20-Prozessor, 8 MHz, X	37,––
PC 10-II 512 K-RAM, AGA-Karte, dt. Tast.+Laufw.+20 MB-Festplatte +Contr.+Monitor	3.887,––
AMIGA-Commodore, 512 K-RAM	1.939,––

APPLE-BUS COMPUTERPLATINEN + PERIPHERIE	LEER/BESTÜCKT
M-board II 48K, gesockelt, vollbest. + gepr.	57,––/337,––
M-board II 64K, gesockelt, vollbest. + gepr.	88,––/375,––
M-board II 64K + Z80 CPU, gesockelt, vollbest. + gepr.	95,––/378,––
M-board IIe 64K vollbestückt + gepr.	98,––/389,––
Superstarkes Netzteil – kurzschlußfest 5A=**115,––** b.z.w. 7A=	145,––
Entwicklungslochraster-Leerplatine durchkontaktiert	27,––
16K-Speicher-Karte geprüft	18,95/ 55,––
APPLE orig. 16K-Language Karte gepr.	98,––
Z-80 CPU-Karte geprüft (CP/M Vol. I.+II Manual dafür = 69,––)	18,95/ 64,––
Controller DOS 3.3-Karte geprüft	18,95/ 63,––
Auto-Controller DOS 3.2/3.3-Karte gepr.	24,––/ 97,––
Erphi AFDC 2 Controller + Autopatch Softw.+Handbuch, gepr.	198,––
Parallel-Drucker-Karte geprüft (Extra Kabel = 45,––)	18,95/ 74,––
Par.-Drucker Karte-Grappler comp. gepr. (Extra Kabel=45,––)	27,––/169,––
8bit par. Int. + 64K-Buffer + Kabel gepr. (Grappler comp.)	39,––/299,––
Seriell-Interface-Karte V24 geprüft	18,95/ 88,––
Super-Seriell-Interface Karte gepr.	34,––/169,––
APPLE-Original Super-Seriell-Interface gepr.	295,––
80-Zeichen/24-Zeilen-Karte geprüft	18,95/119,––
80Z/24Z-Karte + Softswitch-Schalter, gestochen scharf, gepr.	24,––/127,––
80Z/24Z-Softwitch Karte gepr. für IIe	15,––/ 39,––
80Z/24Z+64K-Ram+Softsw.-Karte best.+gepr. f. IIe	18,95/ 87,––
IEEE-488 Int. Karte gepr. (Extra Kabel=45,––)	29,––/297,––
PAL-Modulator/Color-Karte+UHF Modulator, geprüft	18,95/ 99,––
RGB-Color Karte gepr., (nur f. II+)	18,95/129,––
128K-Speicher-Leerkarte + Software + Manual	99,––
128K-Speicher-Karte + Software+Manual, geprüft	29,––/255,––
Mega Ramcard II, 256 KB best.+Software+dt. Hb., (bis 1 MB aufr.)	498,––
Mega Ramcard II, 1 MB best.+Software+dt. Hb., gepr.	878,––
EPROM-Burner (2716/32/64) geprüft	24,––/109,––
6522-VIA Karte geprüft	24,––/135,––
Clock Karte+Software+Manual, geprüft	24,––/ 94,––
Speech Karte+Software+Manual, geprüft	18,95/ 53,––
Wild Karte+Software+Manual (nicht f. IIe), geprüft	18,95/ 59,––
AD/DA-8Bit Karte+Software+ Manual, gepr.	29,––/269,––
UHF/TV-Modulator universell	27,50
Lüfter anclipsbar (220 V)	79,––

APOLLO//(48K) + UHF-Mod. + Gr/KI vollbest. + gepr.	617,––
APOLLO//(48K) + Disk II F + Contr. + 12"-Monitor	1.269,––
APOLLO//A (48K) + Gr/KI + 15er-Tastatur vollbest. + gepr.	677,––
APOLLO// ASKF (48K) vollbest. + gepr. sep.Keyboard + Gr/KI + 15er-Block mit Funkt.-Tasten im „IBM-look like" Gehäuse	835,––
Aufpreis f.o. Systeme von 48K zu 64K	55,––
Aufpreis von 48K zu 64K+Z80 CPU	89,––
Apollo IIe (64K) vollbest. und geprüft	675,––
Apollo IIe (64K)+80Z/24Z-Karte gepr.	698,––
Apollo IIe (128K)+80Z/24Z-Karte geprüft	755,––
Appile IIe (64K)+15er-Tastatur vollbestückt und geprüft	735,––
Apollo IIe ASKF (64 K) vollbestückt und geprüft	ab 845,––
Apollo IIe (64K)+Disk II F+Contr.+12" Monitor	1.387,––
APPLE IIe 128K+80Z/24Z Karte+Softw.	1.598,––

Disk-Drives voll APPLE II, 100%ig compatibel+R/W geprüft+Track ø Sensor:

Disk//+Contr. + Kabel DOS 3.3 (Siemens) im Geh.	359,––
Disk//(Siemens) im Geh. + Kabel	298,––
Disk//F-HS (High Speed 1/2 Höhe) im Gehäuse+Contr.+Kabel	379,––
Disk//F-HS (High Speed 1/2 Höhe) im Geh.+Kabel (Track Ac.<2mS)	318,––
Disk//F-HS-80 Track im Gehäuse + Kabel f. orig. Contr. geeignet	389,––
Disk//F-HS-80 w.o. + Contr. + DOS 3.3 + CP/M Patch-Softw.	489,––
Disk II-2x80 Track 640KB f. Erphi Controller modifiziert, o. Gehäuse	299,––
Erphi DuoDisk 1,2 MByte im Gehäuse + Erphi-AFDC 3-Contr., gepr.	898,––
APPLESOFT + Tutorial + Reference Manual engl.	75,––
APPLE-DOS 3.3 Manual engl. = **25,––**/Handbuch dt.	45,––
APPLE-Pascal Reference + Operating Manual engl.	48,––
APPLE-Fortran Manual engl.	69,––
CP/M-Softcard Vol. I + II, Manual engl.	69,––
Z80 Karte gepr. f. APPLE//c + RAM-Disk-Software + Hb.	447,––
Macintosh Umrüstung von 128K auf 512K	335,––
EPSON FX 85F/T+, 8bit/parallel 160 Z/S. NLQ + IBM comp.	1.097,––
EPSON FX 105+, 8bit/parallel 15" sonst w.o.	1.397,––
EPSON LQ-800, 24 Nadel, 180 Z/Sec. NLQ+IBM comp.	1.488,––
EPSON LQ-1000, wie 800 breite Ausf.	1.937,––
EPSON EX 800,	1.494,––
EPSON Traktoraufsatz für FX 80 + FX 85	129,50
APPLE/EPSON-Drucker Graphic-Interface + Kabel	169,––
MX 80/82, FX/RX 80, FX 85 Spezialfarbband-Kassette	15,95
BROTHER-Typenraddr. HR 15XL 8bit/par. neuestes Modell	1.087,––

Wir führen verschiedene Monitore von Zenith, Phillips, NEC, COGA und Sanyo mit Video – TTL (IBM Komp.) oder Coloreingang. Sowie umfangreiche Computer-Literatur (über 500 versch. Titel) – Bitte Preisliste anfordern! –

Disketten in Box + Aufkleber I. Wahl, 10er Pack/100er Pack./ Stückpreis:

5¼"	BASF	3M	FUJI	DATA MAGNETICS	NEUTRAL
1X, SS/SD	2,79/ 2,59				
1D, SS/DD	2,89/ 2,69	2,88/ 2,78	4,27/ 3,97	–,980/–,936	–,885/–,835
2D, DS/DD	3,49/ 3,29	3,88/ 3,68	5,87/ 5,47	1,180/–,996	1,085/–,895
1D/96TPi	3,89/ 3,59	4,88/ 4,68			
2D/96TPi	4,39/ 4,19	4,68/ 4,38	7,87/ 7,67		
2HD/1.2MB	7,59/ 7,29	7,98/ 7,58	9,87/ 9,57		
1X, Hard	4,29/ 4,19				
8"					
1X, SS/SD	4,49/ 4,39	6,68/ 6,48	6,97/ 6,67		
1D, SS/DD	4,79/ 4,59	8,58/ 8,28			
2D, DS/DD	6,49/ 5,99	9,88/ 9,58	10,17/ 9,77		
1X, Hard	4,49/ 4,39	6,68/ 6,48			
3½"					
Einseitig	4,69/ 4,49	5,88/ 5,58	7,67/ 7,17		
Zweiseitig	6,19/ 5,89	7,28/ 6,88	9,17/ 8,57		

Das sind Preise!

5¼" Disketten-Archivbox für 10 Disks (1 St./10 St.)	4,75/43,50
5¼" Disk-Karteikasten Kunststoff (ca. 80 Disk)	26,50
5¼" Disk-Karteikasten Kunststoff (ca. 80 Disk) Rauchglas	45,95
5¼" Disk-K.K.Rauchglas, abschließb. (f. ca. 80-100 Disks)	47,50
2000 Bl. Tabpapier (24 cm x 12" einf.) weiß oder grün/weiß perf.	45,––
4000 Et.-Aufkl. doppelr. (107 x 36 mm auf 240 x 12" perf. Trägerp.)	66,50
4000 Etik.-Aufkl. einreih. (107 x 36 mm auf 125 x 12" perf. Träg.)	65,––

OSZILLOSCOPE HAMEG ab Lager

Bei Vorauszahlung frei Empfangsstation unversichert in der BRD, ausgenommen Papier und Etiketten, sonst N.N. + V.S. ab DM 30,––

Öffnungszeiten: Mo, Di, Do, Fr v. 10-18 h, Mi u. Sa v. 10-14 h, la Sa v. 10-18 h
Telef. Best.: Mo, Di, Do, Fr von 10-19 h, Mi u. Sa wie Öffnungszeiten.

GEWÄHRLEISTUNG: 6 Monate, auf alle bei uns gekauften Geräte, durch unsere eigene Service-Werkstatt.

REPARATUREN an Apple + compatiblen Geräten + Zubehor führt unser Spezialistenteam garantiert zuverlassig + besonders kostengünstig aus. **Sprechen sie mit uns. Kostenvorschlag auf Wunsch!**

aaa electronic gmbh
Telex 0772642 aaa-d
Habsburgerstraße 134
7800 FREIBURG, Tel. (0761) 276864
Bauelemente – Bausatze – µP's
Meßgeräte – Zubehor – Fachliteratur
Fachgeschaft fur Elektronik + Mikrocomputer

Individueller Zeichensatz für den Epson FX-80

von Dipl.-Ökonom Edgar Meyzis

1. Ausgangslage

Texte können mit Hilfe von Mikrocomputern und angeschlossenen Druckern in vielfacher Weise gestaltet werden. Schriftzeichen sind jedoch nur begrenzt als Mittel des Ausdrucks einsetzbar, da ihr Punktraster in Festwertspeichern abgelegt ist. Dieser Sachverhalt wird besonders deutlich, wenn bestimmte Zeichen benötigt werden, die in dem standardmäßig verfügbaren Zeichensatz nicht enthalten sind (z.B. der griechische Buchstabe Pi). Der Matrixdrucker FX-80 von Epson bietet die Möglichkeit, Zeichen nach eigenen Vorstellungen zu definieren. In Verbindung mit einem Apple II wird beispielhaft ein Weg dazu aufgezeigt. Zwei Programme unterstützen das Vorgehen. Die einzelnen Schritte lassen sich wie folgt skizzieren:

– Drucker vorbereiten
– Gesamten Zeichensatz des Druckers in dessen RAM-Bereich kopieren
– Einen nationalen Zeichensatz als Träger der geänderten Zeichen bestimmen
– Zeichen redefinieren
– Zeichensatz aktivieren

Die einzelnen Schritte sollen im folgenden näher erläutert werden.

2. Vorbereitung des Druckers

Der Drucker verfügt über einen RAM-Bereich, der wahlweise als Pufferspeicher dienen oder einen Zeichensatz aufnehmen kann; ein Schalter bestimmt die jeweilige Funktion. Für den hier verfolgten Zweck ist der DIP-Schalter 4 des Schalterbausteins 1 auf „OFF" zu stellen. (Im Handbuch zum Drucker wird die Lage der Schalter beschrieben.) Dadurch wird die hardwaremäßige Voraussetzung geschaffen, um mit der Programmierung eigener Zeichen beginnen zu können.

3. Zeichensatz kopieren

Der gesamte Zeichensatz wird durch Übertragen der Bytefolge 1B,3A,00,00,00 an den FX-80 aus einem ROM des Drukkers in dessen RAM-Bereich kopiert (s. Programm FX80.INIT, Zeile 41). Dieser Vorgang wird mit „download" bezeichnet.

4. Nationaler Zeichensatz

Als nächstes ist ein nationaler Zeichensatz zu bestimmen, dessen Zeichen durch selbstdefinierte Zeichen ersetzt werden sollen. In unserem Fall wird der deutsche Zeichensatz durch Übertragen der Bytefolge gemäß Programm FX80.INIT, Zeile 45 gewählt. Die Bedeutung der einzelnen Bytes der Folge 1B,52,02 sei beispielhaft erläutert:

1B kennzeichnet den Beginn eines Steuerbefehls an den Drucker.
52 bedeutet „Wähle den Zeichensatz, der dem nachfolgenden Byte entspricht". Dabei spezifiziert 02 den deutschen Zeichensatz.

Weitere Angaben dazu sind dem Druckerhandbuch unter der Kurzbezeichnung „ESC R" zu entnehmen.

5. Exkurs: Zeichenaufbau

Bei der Definition eigener Zeichen ist zu beachten, daß die verschiedenen nationalen Zeichensätze über einen gemeinsamen Zeichenvorrat verfügen. Die in allen Sätzen vorkommenden Zeichen sind deshalb im RAM-Bereich auch nur einfach vorhanden. Daraus ergibt sich, daß u.U. die Zeichensätze im RAM- und im ROM-Bereich wechselweise genutzt werden müßten, wenn der gemeinsame Zeichenvorrat geändert würde. Die Bytefolgen, die die Umschaltung zwischen den Zeichensätzen bewirken, können nicht ohne weiteres in einen Text eingefügt werden, da für das steuernde Byte 1B kein Zeichen definiert ist. Textverarbeitungsprogramme, z.B. Wordstar, können jedoch entsprechend konfiguriert werden. Es wird deutlich, daß die Zeichen, die redefiniert werden sollen, wohlüberlegt zu bestimmen sind.

Bevor wir mit der Definition von Zeichen beginnen, müssen wir uns noch Klarheit über den technisch vorgegebenen Gestaltungsrahmen verschaffen. Die fest defi-

nierten Zeichen des FX-80 werden in einem Raster von 5 x 7 Punkten (Breite x Höhe) gedruckt. Der FX-80 verfügt aber über neun senkrecht angeordnete Drucknadeln. Die oberen sieben Nadeln werden für Zeichen ohne Unterlängen und die unteren sieben für Zeichen mit Unterlängen genutzt. Mit einem Byte können jedoch gleichzeitig acht Drucknadeln angesteuert werden. Von dieser Möglichkeit wird in der Praxis kein Gebrauch gemacht. Die beigefügten Programme schließen sogar die gleichzeitige Betätigung von acht Drucknadeln aus, da ein Byte mit dem Inhalt „FF" das Ende der Tabelle kennzeichnet, die zum Programm FX80.INIT gehört.
Für die Breite der Druckzeichen (fünf Rasterpunkte) stehen bei der Definition zehn Positionen (s. **Abb. 1**) zur Verfügung. Zwischen horizontal benachbarten Punkten soll jeweils eine Position frei bleiben, um ein einwandfreies Druckbild in der proportionalen Betriebsart zu gewährleisten. Auf diese Besonderheit soll hier jedoch nicht weiter eingegangen werden.

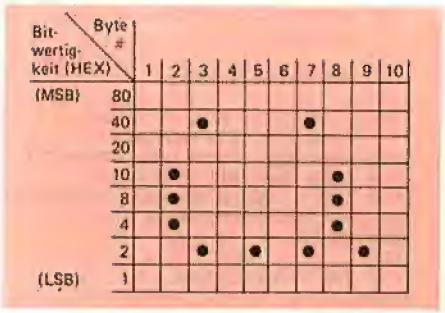


Abb. 1

Zusammengefaßt folgt für unsere Betrachtung, daß wir Zeichen in einem Druckraster von 5 x 8 Punkten (Breite x Höhe) definieren können und in der Horizontalen begrenzt in der Lage sind, halbe Punktabstände zu realisieren.

6. Zeichen redefinieren

Als nächste Frage ist zu klären, wie Positionen einzelner Punkte unserer Zeichen in den RAM-Bereich des Druckers eingegeben werden können, um die dort bereits vorhandenen zu ersetzen. Die Antwort darauf ist einfach: Jeder Spalte der Definitionsmatrix (s. Abb. 1) entspricht ein Byte, das von oben (MSB) nach unten (LSB) zu lesen ist. Die acht Positionen in jeder Spalte bezeichnen die einzelnen Bits mit den hexadezimalen Werten von 80, 40, 20, 10, 8, 4, 2 und 1. Die zehn Bytes, die ein Zeichen definieren, sind – links beginnend – an den Drucker zu übertragen.
Damit ist die Zeichendefinition aber noch nicht vollständig. Es fehlt die Anweisung an den FX-80, ob das jeweilige Zeichen mit den oberen oder mit den unteren acht Nadeln gedruckt werden soll. Dazu wird der Zeichendefinition ein Byte vorangestellt, das das sog. *Attribut* enthält. Für unsere Zwecke nimmt es den Wert „8B" für die oberen Nadeln und „0B" für die unteren 8 Nadeln an. Das Attribut gibt auch Auskunft über die Breite und die Lage eines Zeichens, bezogen auf die zehn möglichen horizontalen Positionen. Wir nutzen vereinfachend alle zehn Spalten und kennzeichnen die unbesetzten durch ein Byte mit dem Inhalt „00".
Unsere Zeichendefinition bedarf noch einer Ergänzung: Der bisher festgelegten Folge von elf Bytes ist noch ein zwölftes anzufügen, quasi als Schlußzeichen, das stets den Wert Null trägt. Wir wissen nun, daß ein neues Zeichen durch Übertragen von zwölf Bytes an den Drucker definiert werden kann.

Endlich können wir das erste Zeichen selbst definieren. Als Beispiel soll der Buchstabe „Umlaut-u" dienen, der sich im Originalzeichensatz kaum von seinem groß geschriebenen Äquivalent unterscheidet. Abb. 1 zeigt den neuen Buchstaben in der Definitionsmatrix, deren Spalten die zehn Bytes und deren Zeilen die einzelnen Positionen im jeweiligen Byte (d.h. die Bits) darstellen. Links am Rand sind die Wertigkeiten der einzelnen Bits vermerkt. Ergänzt um das Attribut und das „Schlußzeichen" ergibt sich für unsere erste „Neuschöpfung" die Bytefolge „8B,00,1C,42,00,02,00,42,1C,02,00"; sie ist im Programm 1 ab Zeile 50 enthalten. Bevor wir diese Bytefolge an den Drucker senden, müssen wir noch spezifizieren, welches Zeichen im RAM-Bereich durch das neudefinierte ersetzt werden soll. Dazu werden dem Drucker Steuerzeichen (1B,26,00) und zweimal der hexadezimale Wert des zu ändernden Zeichens (in unserem Fall 7D) übermittelt (s. FX80.INIT, Zeile 49).

7. Zeichensatz aktivieren

Der in den RAM-Bereich kopierte und modifizierte Zeichensatz muß aktiviert werden, damit er genutzt werden kann. Das erfolgt durch die Bytefolge „1B,25,01,00" (s. FX80.INIT, Zeile 85). Danach arbeitet der Drucker mit dem modifizierten Zeichensatz.

8. FX80.INIT

Dieses Programm enthält sechs redefinierte Zeichen. Es verbessert die Darstellung vorhandener Zeichen. Das Programm bildet aber auch die Basis für den nachfolgend beschriebenen Zeicheneditor.

9. FX80.PATCHER

Das bisher beschriebene Vorgehen ist recht mühsam. Das Applesoft-Programm FX80.PATCHER erleichtert die Bewältigung der Aufgabenstellung erheblich, indem es nicht nur die Definition neuer Zeichen wirksam unterstützt, sondern es zugleich ermöglicht, die Tabelle von FX80.INIT mühelos zu erweitern, zu verkürzen oder in Teilen zu ändern. FX80.PATCHER geht davon aus, daß das Maschinenprogramm unter dem Namen „FX80.INIT" abgespeichert ist.

FX80.PATCHER übernimmt folgende Funktionen:
– Editieren: Analyse des Aufbaus redefinierter Zeichen, die in FX80.INIT enthalten sind, und deren Änderung.
– Redefinieren: Bestimmen des zu ändernden Zeichensatzes sowie Erstellen neuer Zeichen.
– Löschen: Entfernen redefinierter Zeichen aus FX80.INIT.
– Abbrechen: Verlassen des Programms, ohne daß FX80.INIT zurückgeschrieben wird.
– Beenden: Beenden des Programms; FX80.INIT wird bei erfolgter Änderung auf Diskette geschrieben.

Das Programm befreit Sie von Rechenaufgaben. Innerhalb einer Matrix können Sie am Bildschirm einzelne Punkte des angestrebten Druckmusters setzen. Der Cursor wird dazu mit den Tasten „I, J, K, M" bewegt. Die Programmlogik stellt sicher, daß nur bis zu sieben Drucknadeln aus dem Bereich der oberen oder der unteren Nadeln angesteuert werden. Das Programm wurde in Applesoft-BASIC geschrieben, damit weitergehende Wünsche an den Editor leicht realisiert werden können. Die modulare Struktur unterstützt eine Erweiterung. Die ausführliche Dokumentation der Konstanten, Variablen und String-Definitionen erleichtert das Vorgehen.

Für die praktische Anwendung ist noch anzumerken, daß man mit Hilfe von FX80.INIT den FX-80 vor Aufnahme einer Druckarbeit oder vor dem Laden eines Textverarbeitungsprogramms initialisiert. Der Drucker arbeitet danach wie gewöhnlich mit der Ausnahme, daß der Pufferspeicher nicht verfügbar ist.

Und nun viel Erfolg bei der Gestaltung Ihrer Zeichen. Bedenken Sie, daß mit dieser Arbeit nur ein Einstieg vermittelt werden sollte. Die Möglichkeiten Ihres FX-80 reichen noch weiter.

Kurzhinweise

1. Zweck:
Gestaltung eigener Zeichensätze für FX80
2. Konfiguration:
II+/e/c; DOS 3.3 (kein ProDOS bei FX80.PATCH; ProDOS möglich bei fertigem FX80.INIT); Epson FX-80; Parallelschnittstelle mit C090/C1C1-Protokoll
3. Test:
RUN FX80.PATCHER
Wenn Zeichensatz fertig erstellt ist
BRUN FX80.INIT
4. Sammeldisk:
FX80.PATCHER
T.FX80.INIT (Big-Mac-Quelltext)
FX80.INIT (Objektcode)

FX80.PATCHER

```
1000 REM PATCHER FUER FX80-INIT
1005 :
1010 REM Z E I C H E N E D I T O R
1015 :
1020 GOSUB 1740: REM INITIALISIEREN
1025 GOTO 1290: REM MENUESTEUERUNG
1030 :
1035 REM TASTATURABFRAGE
1040 :
1045 POKE KSTRB,0
1050 HVAR = PEEK (KEYBD): IF HVAR
     < 128 THEN 1050
1055 HVAR = HVAR - 128: RETURN
1060 :
1065 REM CURSOR STEUERUNG
1070 :
1075 IVAR = 5:JVAR = 4
1080 POKE SHOW(IVAR,JVAR),171
1085 GOSUB 1035: ONERR GOTO 1095
1090 ON (HVAR - 72) GOTO 1125,1145,1165,
     1205,1185
1095 POKE 216,0
1100 IF HVAR = 83 THEN 1235
1105 IF HVAR = 65 THEN GOSUB 1955:
     EDITIEREN = 0: GOTO 1290:
     REM ABBRECHEN
1110 IF HVAR = 69 THEN 1260
1115 GOTO 1080
1120 :
1125 IF JVAR = OG THEN 1080: REM UP
1130 POKE SHOW(IVAR,JVAR),
     BIT(IVAR,JVAR)
1135 JVAR = JVAR - 1: GOTO 1080
1140 :
1145 IF IVAR = 0 THEN 1080: REM LEFT
1150 POKE SHOW(IVAR,JVAR),
     BIT(IVAR,JVAR)
1155 IVAR = IVAR - 1: GOTO 1080
1160 :
1165 IF IVAR = 9 THEN 1080: REM RIGHT
1170 POKE SHOW(IVAR,JVAR),
     BIT(IVAR,JVAR)
1175 IVAR = IVAR + 1: GOTO 1080
1180 :
1185 IF JVAR = UG THEN 386: REM DOWN
1190 POKE SHOW(IVAR,JVAR),
     BIT(IVAR,JVAR)
1195 JVAR = JVAR + 1: GOTO 1080
1200 :
1205 BIT(IVAR,JVAR) = 160: REM LOESCHEN
1210 POKE SHOW(IVAR,JVAR),160
1215 IF JVAR = 0 THEN Z0 = Z0 - 1:
     IF Z0 < = 0 THEN Z0 = 0:UG = 8
1220 IF JVAR = 8 THEN Z8 = Z8 - 1:
     IF Z8 < = 0 THEN Z8 = 0:OG = 0
1225 GOTO 1080
1230 :
1235 IF JVAR = 0 THEN Z0 = Z0 + 1:
     UG = 7
1240 IF JVAR = 8 THEN Z8 = Z8 + 1:
     OG = 1
1245 BIT(IVAR,JVAR) = 170: REM SETZEN
1250 POKE SHOW(IVAR,JVAR),170:
     GOTO 1085
1255 :
1260 REM EINGEBEN
1265 :
1270 Z0 = 0: Z8 = 0: IF EDITIEREN
     = 1 THEN 1715
1275 GOSUB 1400
1280 GOSUB 1995: GOSUB 2285: GOTO 1305
1285 :
1290 REM MENUE STEUERUNG
1295 :
1300 EDITIEREN = 0
1305 POKE 1977,127: GOSUB 1035
1310 IF CHR$ (HVAR) = "A" THEN 2905
1315 IF CHR$ (HVAR) = "B" THEN 2875
1320 IF CHR$ (HVAR) = "E" THEN
     POKE 1977,160: GOTO 1670
1325 IF CHR$ (HVAR) = "L" THEN
     POKE 1977,160: GOSUB 1470
1330 IF CHR$ (HVAR) = "R" THEN
     POKE 1977,160: GOTO 1345
1335 GOTO 1305
1340 :
1345 REM REDEFINIEREN
1350 :
1355 OG = 0: UG = 8: ZPSN = 1232:
     UNI$ = M3$: GOSUB 1530:
     POKE ZPSN,127
1360 GOSUB 1035: IF HVAR < 48 THEN 1360
1365 IF HVAR > 56 THEN 1360
1370 POKE ZPSN,(128 + HVAR):
     STZ = HVAR + 128
1375 ZPSN = 1104: UNI$ = M2$: GOSUB 1530:
     POKE ZPSN,127
1380 GOSUB 1035: CH = 128 + HVAR:
     POKE ZPSN,CH: CLEAN = FRE (0)
1385 GOSUB 1565: IF GEFUNDEN = 1
     THEN GOSUB 2305: GOTO 1305
1390 GOSUB 2245: GOTO 1065
1395 :
1400 REM BIT-MATRIX-->FKETTE%
1405 :
1410 MOD = 1: POKE SHOW(IVAR,JVAR),160:
     REM CURSOR LOESCHEN
1415 FKETTE%(6) = 139: REM ARGUMENT
1420 IF OG = 1 THEN FKETTE%(6) = 11
1425 FOR IVAR = 0 TO 9
1430 HVAR = 0: KVAR = 0: NVAR = 1
1435 FOR ZAEHLER = UG TO OG STEP - 1
1440 IF BIT(IVAR,ZAEHLER) = 170
     THEN HVAR = HVAR + NVAR
1445 BIT(IVAR,ZAEHLER) = 160:
     POKE SHOW(IVAR,ZAEHLER),160
1450 KVAR = KVAR + 1: NVAR =
     2 ↑ KVAR: NEXT
1455 IF HVAR = 255 THEN GOSUB 2285:
     IVAR = 10: NEXT: POP: GOTO 1305
1460 FKETTE%(7 + IVAR) = HVAR:
     NEXT: RETURN
1465 :
1470 REM LOESCHEN
1475 :
1480 ZPSN = 1106: UNI$ = M2$: GOSUB 1530:
     POKE ZPSN,127: REM ZEICHEN ?
1485 GOSUB 1035: POKE ZPSN,(128 + HVAR)
1490 GOSUB 1565: REM ZEICHENSUCHEN
1495 IF GEFUNDEN = 0 THEN 1520
1500 GOSUB 1600: REM CODE VERSCHIEBEN
1505 ZPSN = ZBEG:IVAR = LEN (ZEICHEN$):
     GOSUB 1640
1510 ZPSN = ZBEG:ZEICHEN$ = "":
     GOSUB 1810
1515 GOSUB 1850: REM ZEICHEN DARSTELLEN
1520 GOSUB 2310: RETURN
1525 :
1530 REM ZEILE SCHREIBEN
1535 :
1540 FOR ZAEHLER = 1 TO LEN (UNI$)
1545 NVAR = ASC (MID$ (UNI$,ZAEHLER,1))
1550 POKE ZPSN,128 + NVAR
1555 ZPSN = ZPSN + 1: NEXT : RETURN
1560 :
1565 REM ZEICHEN SUCHEN
1570 IF LEN (ZEICHEN$) = 0 THEN 1590
1575 FOR LOOP = 1 TO LEN (ZEICHEN$)
1580 IF ASC ( MID$ (ZEICHEN$,LOOP,1))
     = HVAR THEN GEFUNDEN = 1:
     ZAEHLER = LOOP: LOOP =
     LEN (ZEICHEN$) + 1: NEXT: RETURN
1585 NEXT
1590 GEFUNDEN = 0: RETURN
1595 :
1600 REM CODE LOESCHEN DURCH VERSCHIEBEN
1605 :
1610 IVAR = ANFG + (ZAEHLER - 1) * 17
1615 FOR PTRCOD = IVAR TO CE
1620 POKE PTRCOD, PEEK (PTRCOD + 17)
1625 NEXT: MOD = 1
1630 CE = CE - 17: RETURN
1635 :
1640 REM ANZEIGE LOESCHEN
1645 :
1650 FOR ZAEHLER = 1 TO IVAR
1655 POKE ZPSN,160: ZPSN = ZPSN + 1:
     IF ZPSN = 1320 THEN ZPSN = 1418
1660 NEXT: RETURN
1665 :
1670 REM ZEICHEN EDITIEREN
1675 :
1680 EDITIEREN = 1
1685 ZPSN = 1106: UNI$ = M2$:
     GOSUB 1530:
     POKE ZPSN,127:
     REM ZEICHEN AUSGEBEN
1690 GOSUB 1035: POKE ZPSN,(128 +
     HVAR): GOSUB 1565:
     REM ZEICHENSUCHEN
1695 IF GEFUNDEN = 0 THEN 1725:
     REM ANZEIGE LOESCHEN
1700 GOSUB 2060: REM ZEICHEN-->FKETTE%
1705 GOSUB 2100:
     REM FKETTE%-->BITMATRIX
1710 GOSUB 2245: GOTO 1065:
     REM HILFSMENUE
1715 GOSUB 1400: REM MATRIX-->FKETTE%
1720 GOSUB 2210: REM FKETTE%-->CODE
1725 GOSUB 2285: REM HILFSMENUE
     LOESCHEN
1730 EDITIEREN = 0: CLEAN = FRE (0):
     GOTO 1305
1735 :
1740 REM INITIALISIEREN
1745 :
1750 GOSUB 2335: PRINT D$;"NOMON,C,I,O":
     GOSUB 2595: REM PARAM/MENUE
1755 GOSUB 2635: GOSUB 2725: REM MATRIX
1760 GOSUB 2800: REM DRUCKERSTRING
1765 ZPSN = ZBEG: GOSUB 2845: GOSUB 1780:
     REM FX80 LADEN/ANALYS.
1770 RETURN
1775 :
1780 REM CODE ANALYSIER.
1785 :
1790 GOSUB 1895: GOSUB 1930:
     REM SATZ/ENDE?
1795 GOSUB 1810: GOSUB 1850:
     REM ZEICHENERMITTELN/DARSTELLEN
1800 RETURN
1805 :
1810 REM ZEICHEN ERMITTELN
1815 :
1820 FOR PTRCOD = (ANFG + 3) TO
     (CE - 17) STEP 17
1825 ZEICHEN$ = ZEICHEN$ + CHR$
     (PEEK (PTRCOD))
1830 NEXT
1835 IF ASC (ZEICHEN$) = 00 THEN
     ZEICHEN$ = ""
1840 CLEAN = FRE (0): RETURN
1845 :
1850 REM ZEICHEN DARSTELLEN
1855 :
1860 IF LEN (ZEICHEN$) = 0 THEN RETURN
1865 FOR ZAEHLER = 1 TO LEN (ZEICHEN$)
1870 POKE ZPSN,128 + ASC
     (MID$ (ZEICHEN$,ZAEHLER,1))
1875 ZPSN = ZPSN + 1: IF ZPSN = 1320
     THEN ZPSN = 1418
1880 NEXT
1885 RETURN
1890 :
1895 REM FX80-INIT ENDE ?
1900 :
1905 LAENGE = PEEK (PLAENGE) + PEEK
     (PLAENGE + 1) * 256
1910 FOR LOOP = ANFG TO (ADDR + LAENGE)
1915 IF PEEK (LOOP) = CMARKE THEN
     CE = LOOP: LOOP = ADDR + LAENGE + 1
1920 NEXT: RETURN
1925 :
1930 REM SATZ FESTSTELLEN
1935 :
1940 STZ = PEEK (ANFG - 1):
     POKE SPSN,(STZ + 176)
```

```
1945 RETURN
1950 :
1955 REM EDITIERHILFEN LOESCHEN
1960 :
1965 FOR IVAR = Ø TO 9
1970 FOR JVAR = Ø TO 8
1975 POKE SHOW(IVAR,JVAR),16Ø:
     BIT(IVAR,JVAR) = 16Ø
1980 NEXT: NEXT: ZØ = Ø: Z8 = Ø
1985 GOSUB 2285: RETURN
1990 :
1995 REM ZEICHEN AN CODE HAENGEN
2ØØØ :
2ØØ5 POKE ANF - 1,(STZ - 176):
     PTRCOD = CE - 5
2Ø1Ø FKETTE%(4) = CH - 128:FKETTE%(5) =
     FKETTE%(4)
2Ø15 FOR ZAEHLER = 1 TO 22
2Ø2Ø PTRCOD = PTRCOD + 1
2Ø25 POKE PTRCOD,FKETTE%(ZAEHLER)
2Ø3Ø NEXT: CE = PTRCOD
2Ø35 LVAR = ZBEG + LEN (ZEICHEN$)
2Ø4Ø IF LVAR > 1319 THEN LVAR =
     LVAR + 98
2Ø45 POKE LVAR,CH
2Ø5Ø POKE SPSN,STZ: ZEICHEN$ =
     ZEICHEN$ + CHR$ (CH - 128)
2Ø55 CLEAN = FRE (Ø): RETURN
2Ø6Ø REM ZEICHEN-->FKETTE%
2Ø65 :
2Ø7Ø LVAR = ANFG + 5 + (ZAEHLER - 1)
     * 17: REM ZEIGER-AUF-ZEICHEN-BEG.
2Ø75 PTRCOD = LVAR
2Ø8Ø FOR ZAEHLER = 6 TO 16
2Ø85 FKETTE%(ZAEHLER) = PEEK (PTRCOD)
2Ø9Ø PTRCOD = PTRCOD + 1: NEXT: RETURN
2Ø95 :
21ØØ REM FKETTE%-->BIT MATRIX
21Ø5 :
211Ø OG = Ø: UG = 7
2115 IF FKETTE%(6) = 139 THEN JVAR = OG:
     GOTO 2125
212Ø OG = OG + 1: UG = UG + 1
2125 FOR IVAR = Ø TO 9: KVAR = 7: MVAR
     = 128
213Ø FOR JVAR = OG TO UG
2135 HVAR = FKETTE%(IVAR + 7): IF HVAR =
     Ø THEN JVAR = UG
214Ø HVAR = HVAR - MVAR
2145 IF HVAR < Ø THEN 216Ø
215Ø FKETTE%(IVAR + 7) = FKETTE%(IVAR +
     7) - MVAR
2155 BIT(IVAR,JVAR) = 17Ø:
     POKE SHOW(IVAR,JVAR),17Ø
216Ø KVAR = KVAR - 1: MVAR = 2 ↑ KVAR
2165 NEXT: NEXT
217Ø ZAEHLER = Ø: ZØ = Ø: Z8 = ZØ
2175 IF OG = Ø THEN JVAR = Ø:
     GOSUB 219Ø: ZØ = ZAEHLER: RETURN
218Ø JVAR = 8: GOSUB 219Ø: Z8 = ZAEHLER:
     RETURN
2185 :
219Ø FOR IVAR = Ø TO 9
2195 IF BIT(IVAR,JVAR) = 17Ø THEN
     ZAEHLER = ZAEHLER + 1
22ØØ NEXT: RETURN
22Ø5 :
221Ø REM EDITERGEBNIS-->CODE
2215 :
222Ø PTRCOD = LVAR
2225 FOR ZAEHLER = 6 TO 16
223Ø POKE PTRCOD,FKETTE%(ZAEHLER)
2235 PTRCOD = PTRCOD + 1: NEXT: RETURN
224Ø :
2245 REM HILFSMENUE
225Ø :
2255 ZPSN = 1488: UNI$ = M4$: GOSUB 153Ø
226Ø ZPSN = 1616: UNI$ = M9$: GOSUB 153Ø
2265 ZPSN = 1744: UNI$ = M7$: GOSUB 153Ø
227Ø ZPSN = 1872: UNI$ = MØ$: GOSUB 153Ø
2275 CLEAN = FRE (Ø): RETURN
228Ø :
2285 REM ANZEIGEN LOESCHEN
229Ø :
2295 ZPSN = 1488: GOSUB 2315
23ØØ ZPSN = 1616: GOSUB 2315:
     ZPSN = 1744: GOSUB 2315:
     ZPSN = 1872: GOSUB 2315
23Ø5 ZPSN = 1232: GOSUB 2315: REM SATZ
231Ø ZPSN = 1104: REM ZEICHEN
2315 FOR ZAEHLER = ZPSN TO (ZPSN + 13)
```

```
232Ø POKE ZAEHLER,16Ø: NEXT
2325 RETURN
233Ø :
2335 REM KONSTANTEN
234Ø :
2345 ADDR = 32768: HIMEM: (ADDR - 1):
     REM MASCHINENPROGRAMM
235Ø CMARKE = 255: REM CODE ENDE
2355 KEYBD = - 16384: KSTRB = - 16368
236Ø PLAENGE = 43616: REM ENTHAELT
     PROGRAMMLAENGE: 48K-DOS!
2365 DIM SHOW(9,8): REM HOR,VERT MATRIX
     SCREEN POS
237Ø SPSN = 1546: REM SCREEN SATZ POS
2375 ZBEG = 129Ø: REM SCREEN ZEICHEN POS
238Ø :
2385 REM VARIABLEN
239Ø :
2395 ANFG = ADDR + 37: REM PATCHBEREICH
24ØØ DIM BIT(9,8): REM MATRIX MIT
     ZEICHEN FUER FX8Ø
24Ø5 CE = ANFG: REM CODE END ADDR
241Ø CH = Ø: REM ASCII-ZEICHEN
2415 CLEAN = Ø: REM FREE MEMORY
242Ø CODE = Ø: REM EINZUFUEGEN
2425 DIM FKETTE%(22): REM DRUCKERSTRING
243Ø EDITIEREN = Ø: REM FLAG
2435 HVAR = Ø: REM HILFSVAR
244Ø IVAR = Ø: JVAR = Ø: KVAR = Ø:
     LVAR = Ø: MVAR = Ø
2445 LAENGE = Ø: REM DES CODES
245Ø LOOP = Ø: REM ZAEHLER IN FOR/NEXT
2455 MOD = Ø: REM FLAG CODE AENDERUNG
246Ø OG = Ø: REM ARRAYBEGRENZER
2465 PTRCOD = ANFG: REM ZEIGER FUER CODE
247Ø STZ = Ø: REM ZEICHENSATZ
2475 UG = 8: REM ARRAYBEGRENZER
248Ø ZAEHLER = 1: REM ALLROUND
2485 ZØ = Ø: Z8 = Ø:
     REM FLAG-BESETZUNGS-MATRIX
249Ø ZPSN = ZBEG: REM ZAEHLER ZEICH POS
2495 :
25ØØ REM STRINGS
25Ø5 :
251Ø D$ = CHR$ (4): REM CTRL-D
252Ø UNI$ = " ": REM ARBEITSSTRING
2525 ZEICHEN$ = "":
     REM GEAENDERTE ZEICHEN
253Ø :
2535 M1$ = " ZEICHENDEFINITION FUER
     FX-8Ø"
254Ø M2$ = " ZEICHEN: "
2545 M3$ = " SATZ  : "
255Ø M4$ = " [A]BBRECHEN"
2555 M5$ = " [B]EENDEN"
256Ø M6$ = " [E]DITIEREN"
2565 M7$ = " [L]OESCHEN"
257Ø M8$ = " [R]EDEFINIEREN"
2575 M9$ = " [S]ETZEN"
258Ø MØ$ = " [E]INGEBEN"
2585 RETURN
259Ø :
2595 REM HAUPT-MENUE ANZEIGEN
26ØØ :
26Ø5 HOME: VTAB 1: PRINT M1$:
     VTAB 3: PRINT M2$
261Ø VTAB 5: PRINT M3$: VTAB 7:
     PRINT M4$
2615 VTAB 9: PRINT M5$: VTAB 11:
     PRINT M6$
262Ø VTAB 13: PRINT M7$: VTAB 15:
     PRINT M8$
2625 RETURN
263Ø :
2635 REM MATRIX
264Ø :
2645 HVAR = 173: IVAR = 1: KVAR = 1Ø:
     REM -,STEP
265Ø GOSUB 269Ø
2655 DATA  1682,1938,121Ø,1466,1722
266Ø DATA  1978,125Ø,15Ø6,1762,2Ø18
2665 HVAR = 186: IVAR = 2: KVAR = 9
267Ø DATA  181Ø,1Ø82,1338,1594,185Ø
2675 DATA  1122,1378,1634,189Ø
268Ø GOSUB 269Ø: RETURN
2685 :
2690 REM GITTER ZEICHNEN
2695 :
27ØØ FOR JVAR = 1 TO KVAR: READ ZPSN
27Ø5 FOR ZAEHLER = 1 TO 21 STEP IVAR
271Ø POKE (ZPSN + ZAEHLER),HVAR
2715 NEXT: NEXT: RETURN
```

```
272Ø :
2725 REM ARRAY MIT SCREEN ADDR FUELLEN
273Ø DATA 1812,1Ø84,134Ø,1596,1852
2735 DATA 1124,138Ø,1636,1892
274Ø FOR IVAR = Ø TO 8: READ ZPSN
2745 FOR JVAR = Ø TO 9
275Ø SHOW(JVAR,IVAR) = ZPSN
2755 ZPSN = ZPSN + 2: NEXT: NEXT
276Ø :
2765 REM SPACES-->MATRIX
277Ø :
2775 FOR JVAR = Ø TO 9
278Ø FOR IVAR = Ø TO 8
2785 BIT(JVAR,IVAR) = 16Ø
279Ø NEXT: NEXT: RETURN
2795 :
28ØØ REM DRUCKERSTRING VORBEREITEN
28Ø5 :
281Ø FKETTE%(1) = 27: FKETTE%(2) = 38
2815 FOR ZAEHLER = 18 TO 22
282Ø READ FKETTE%(ZAEHLER)
2825 NEXT
283Ø DATA  27,37,Ø1,ØØ,255
2835 RETURN
284Ø :
2845 REM FX8Ø-INIT LADEN
285Ø :
2855 UNI$ = "BLOADFX8Ø.INIT,A" +
     STR$ (ADDR)
286Ø PRINT D$; UNI$
2865 RETURN
287Ø :
2875 REM BEENDEN
288Ø :
2885 IF MOD = Ø THEN 2915: HVAR =
     CE - ADDR + 1
289Ø UNI$ = "BSAVEFX8Ø.INIT,A" +
     STR$ (ADDR) + ",L" + STR$ (HVAR)
2895 PRINT D$; UNI$
29ØØ :
29Ø5 REM ABBRECHEN
291Ø :
2915 POKE KSTRB,Ø: PRINT D$;"MON,C,I,O":
     HOME: END
```

FX80.INIT

```
BSAVE FX80.INIT,A$8000,L144

                    1    * Zeichensatz des FX80 modifizieren
                    2    *
                    3    * Adressen fuer Drucker in SLOT 1
                    4    *
                    5    BUSY     =    $C1C1
                    6    PRINTER  =    $C090
                    7    *
                    8    * Marke fuer Tabellenende
                    9    *
                    10   STOP     =    $FF
                    11   *
                    12   * Tabellenzeiger auf 1. Element
                    13   *
8000: A0 00         14   ADDR     LDY  #0
                    15   *
                    16   * Tabelle an Drucker uebertragen
                    17   *
8002: B9 1B 80      18   LOOP     LDA  TAB,Y
8005: 2C C1 C1      19   WAIT     BIT  BUSY
8008: 30 FB         20            BMI  WAIT
800A: 8D 90 C0      21            STA  PRINTER
                    22   *
                    23   * Tabellenende erreicht?
                    24   *
800D: C9 FF         25            CMP  #STOP
800F: F0 09         26            BEQ  FIN        ; ja
8011: C8            27            INY
8012: D0 EE         28            BNE  LOOP       ; nein
8014: EE 04 80      29            INC  LOOP+2
8017: 4C 02 80      30            JMP  LOOP
                    31   *
801A: 60            32   FIN      RTS
                    33   *
                    34   * Drucker normalisieren
                    35   *
801B: 1B 40         36   TAB      HEX  1B40
                    37   *
                    38   * Zeichensatz in den RAM-Bereich
                    39   * des Druckers kopieren
                    40   *
801D: 1B 3A 00      41            HEX  1B3A000000
8020: 00 00
                    42   *
                    43   * Deutschen Zeichensatz waehlen
                    44   *
8022: 1B 52 02      45            HEX  1B5202
                    46   *
                    47   *             Umlaut-u
                    48   *
8025: 1B 26 00      49   ANFG     HEX  1B26007D7D
8028: 7D 7D
802A: 8B 00 1C      50            HEX  8B001C420002

802D: 42 00 02
8030: 00 42 1C      51            HEX  00421C020000
8033: 02 00 00
                    52   *
                    53   *             Buchstabe-O
                    54   *
8036: 1B 26 00      55            HEX  1B26004F4F
8039: 4F 4F
803B: 8B 38 44      56            HEX  8B3844820082
803E: 82 00 82
8041: 00 82 44      57            HEX  00824438 0000
8044: 38 00 00
                    58   *
                    59   *             Umlaut-O
                    60   *
8047: 1B 26 00      61            HEX  1B26005C5C
804A: 5C 5C
804C: 8B 98 24      62            HEX  8B9824420042
804F: 42 00 42
8052: 00 42 24      63            HEX  00422498 0000
8055: 98 00 00
                    64   *
                    65   *             Umlaut-o
                    66   *
8058: 1B 26 00      67            HEX  1B26007C7C
805B: 7C 7C
805D: 8B 00 0C      68            HEX  8B000C520012
8060: 52 00 12
8063: 00 52 0C      69            HEX  00520C000000
8066: 00 00 00
                    70   *
                    71   *             Umlaut-A
                    72   *
8069: 1B 26 00      73            HEX  1B26005B5B
806C: 5B 5B
806E: 8B 8E 10      74            HEX  8B8E10284088
8071: 28 40 88
8074: 40 28 10      75            HEX  40281 08E0000
8077: 8E 00 00
                    76   *
                    77   *             Buchstabe-o
                    78   *
807A: 1B 26 00      79            HEX  1B26006F6F
807D: 6F 6F
807F: 8B 00 1C      80            HEX  8B001C220022
8082: 22 00 22
8085: 00 22 1C      81            HEX  00221C000000
8088: 00 00 00
                    82   *
                    83   * Zeichensatz aktivieren
                    84   *
808B: 1B 25 01      85            HEX  1B2501 00
808E: 00
                    86   *
808F: FF            87            HEX  FF         ; STOP
```

BRK – nur zum Anhalten?

von Michael G. Schneider

1. Einführung

Eine der Anweisungen im Befehlssatz des 6502-Prozessors, deren Möglichkeiten für gewöhnlich nicht erschöpfend genutzt werden, ist der BRK-Befehl. Üblicherweise wird dieser nur dazu benutzt, um ein Maschinenprogramm abrupt anzuhalten und gleichzeitig die Inhalte der verschiedenen Register und des Programmzählers auszugeben. Das im Apple II befindliche Autostart-ROM ermöglicht es jedoch, vollständige Kontrolle darüber auszuüben, durch welche Routine ein BRK behandelt werden soll.
Ich werde im weiteren zunächst einmal die genauen Abläufe im Inneren des 6502 bei Erkennen eines BRKs erläutern. Daran schließt sich dann eine sinnvolle praktische Anwendung an. Und zwar soll eine Utility vorgestellt werden, mit deren Hilfe es möglich ist, Assemblerprogramme verschiebbar zu machen.

Wollte man bisher eine Routine in unterschiedlichen Speicherbereichen ausführen, so mußte man sich bei den Sprüngen auf Branch-Befehle (BEQ, BCC usw.) beschränken, denn diese geben keine absolute Sprungadresse an, sondern definieren im Unterschied zu JMP oder JSR eine Adresse relativ zum momentanen Inhalt des Programmzählers. Der Zugriff auf bestimmte Speicherzellen (etwa durch LDA, STA, AND usw.) mußte in jedem Fall über absolute Adressen erfolgen, denn der 6502 kennt hierbei keine relative Adressierung.
Mit der von mir vorgestellten Technik wird es möglich, alle Anweisungen mit relativen Adressen zu versehen. Damit können dann Unterprogramme völlig unabhängig von der Ladeadresse aufgerufen und Stellen innerhalb des eigenen Codes adressiert werden.

2. Interrupts

Wenn ein Programm von einem Prozessor ausgeführt wird, so erfolgt dies normalerweise Schritt für Schritt. Um jedoch auch auf externe Ereignisse möglichst schnell reagieren zu können, sind Prozessoren mit der Fähigkeit zum Interrupt ausgestattet.
Nehmen wir einmal an, daß ein Computer einen technischen Prozeß überwacht und daß eines der angeschlossenen Meßgeräte einen kritischen Zustand registriert. Dann ist es in der Regel nicht vertretbar, daß darauf gewartet wird, bis der Computer (die Software) dieses Gerät überprüft. Vielmehr muß der Prozessor unmittelbar über die neue Situation informiert werden. Das Meßgerät ist dafür mit der CPU über eine spezielle Leitung verbunden und kann über diese melden, daß eine besondere Aktivität erforderlich ist. Sobald der Prozessor diese Nachricht erhält, unterbricht er sofort das laufende Programm und arbeitet eine spezielle Unterbrechungsroutine ab. Da er aber nach deren Beendigung wieder zu seiner eigentlichen Aufgabe zurückkehren soll, muß er sich zuvor die Adresse, an welcher er unterbrochen wurde, sowie seinen internen Zustand (Flags und evtl. auch Register) merken.

Der 6502 besitzt nun 3 derartige Leitungen, nämlich RESET, IRQ und NMI. Davon ist RESET dem normalen Benutzer eines Apple II wohl noch am geläufigsten, denn er kann diesen Interrupt einfach durch Betätigung der RESET-Taste aktivieren. Man benutzt ihn, um den Apple II in einen wohldefinierten Anfangszustand zu versetzen. Die beiden anderen Anforderungen können im wesentlichen nur durch Interfacekarten (z.B. eine Uhr) ausgelöst werden. Der IRQ-Impuls unterscheidet sich vom NMI dadurch, daß er vom 6502 auch ignoriert werden kann. Falls nämlich das Interrupt Disable Bit durch einen SEI-Befehl gesetzt wurde, so übersieht der 6502 sämtliche IRQ-Anforderungen.

Wenn ein IRQ oder ein NMI zum Prozessor durchdringt, schließt dieser erst einmal den momentan bearbeiteten Befehl ab. Danach wird der Inhalt des Programmzählers (höherwertiges Byte zuerst), gefolgt von dem P-Register (Prozessor Status Register, Flags), auf dem Stack abgelegt. Somit ist gesichert, daß nach der Behandlung des Interrupts der Zustand zum Zeitpunkt der Unterbrechung wiederhergestellt werden kann. Um weitere IRQs (nicht aber NMIs) zurückzuweisen, wird daran anschließend das Interrupt Disable Bit gesetzt.

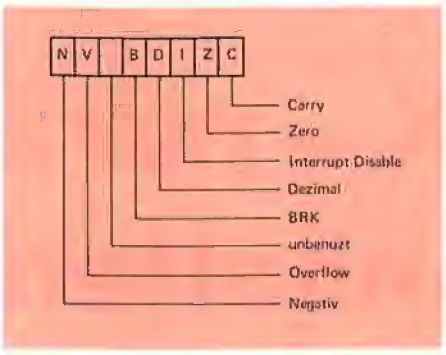
Es sollte ausdrücklich darauf hingewiesen werden, daß die oben beschriebenen Tätigkeiten im Prozessor fest „verdrahtet" sind. Man darf also nicht etwa im Speicherbereich nach Anweisungen suchen, die eben diese Aufgaben erfüllen.
Es ist weiterhin erwähnenswert, daß der gesicherte Programmzähler immer auf das letzte Byte der zuletzt ausgeführten Anweisung zeigt. Er gibt also grundsätzlich die Adresse der folgenden Anweisung minus 1 an. Dies ist auch konsistent mit der Behandlung eines JSR-Befehls. Wird

nämlich z.B. ein JSR auf $1234 ausgeführt, so wird als Rücksprungadresse nicht etwa $1237 (= $1234+3) sondern $1236 (= $1234+2) auf dem Stack abgelegt. Trifft der 6502 dann auf ein RTS, so wird die oben auf dem Stack befindliche Adresse in den Programmzähler transferiert, wo sie sofort inkrementiert wird (= $1236+1).

Nach diesem in die Hardware integrierten Teil einer Interruptbehandlung geht es nun per Software weiter. Dazu befinden sich am Ende des Speicherbereichs 3 Adressen, welche angeben, wo nach Erkennen eines NMIs ($FFFA-$FFFB), eines RESETs ($FFFC-$FFFD) oder eines IRQs ($FFFE-$FFFF) fortgefahren werden soll. Wenn also z.B. ein IRQ ausgelöst wurde, wird nach der oben beschriebenen Rettung von Programmzähler und P-Register bei $FA40 (das ist der Inhalt von $FFFE-$FFFF) gestartet.
Das dann ablaufende Programm muß erst einmal die übrigen Register retten, sofern es diese benutzen möchte. Danach überprüft es, von welcher Stelle der Interrupt ausgelöst wurde, um das externe Ereignis dann in spezifischer Weise behandeln zu können. Verlassen wird eine solche IRQ- oder NMI-Routine durch den RTI-Befehl. Dieser restauriert zunächst die Flags, indem er das oberste Byte vom Stack in das P-Register holt. Anschließend kehrt er ebenso wie der normale RTS-Befehl zum unterbrochenen Programm zurück, denn die Information über die Rücksprungadresse befindet sich dann oben auf dem Stack.

Es erhebt sich jetzt aber sicherlich die Frage, warum denn so ausführlich die Interrupts diskutiert werden, wo doch die Behandlung eines BRKs im Vordergrund stehen sollte. Doch dies kann sehr einfach beantwortet werden. Es ist nämlich so, daß der 6502-Prozessor, sobald er auf ein BRK trifft, eine Interruptsequenz analog dem IRQ durchläuft. Somit wird also auch bei einem BRK der Programmzähler sowie das P-Register auf dem Stack abgelegt und dann zur IRQ-Routine verzweigt. Damit diese Routine aber ein BRK von einem echten IRQ unterscheiden kann, wird das P-Register, bevor es gerettet wird, modifiziert. Und zwar wird im Falle eines BRKs das BRK-Bit (s. **Abbildung 1**) gesetzt und ansonsten zurückgesetzt.
Eine besondere Behandlung des BRKs durch den 6502, die sich vom normalen IRQ unterscheidet, muß noch kurz erwähnt werden. Wenn ein BRK auf der Adresse A angetroffen wurde, so wird die Adresse A + 2 auf dem Stack gerettet. Dies ist aber sicherlich schon jedermann aufgefallen, denn wenn z.B. ein BRK auf der Adresse $0300 ausgeführt wird, so

Abbildung 1



gibt der Monitor neben den Inhalten der Register auch die Adresse $0302 aus.

Doch wie geht es nun in der Software weiter, nachdem ein BRK-Befehl vom Prozessor registriert wurde? Wir müssen uns dafür die IRQ-Routine ($FA40) im *alten* Autostart-ROM ansehen. (Dies gilt nicht für die neuen IIe/c-ROMs und den alten IIc, denn bei letzterem wird beispielsweise nach $C806 verzweigt, um mögliche Maus-Interrupts zu berücksichtigen.)

```
IRQ   STA $45
      PLA
      PHA
      ASL
      ASL
      ASL
      BMI BREAK
      JMP ($03FE)
BREAK PLP
      JSR SAV1
      PLA
      STA $3A
      PLA
      STA $3B
      JMP ($03FØ)
```

Man erkennt, daß zunächst der Akkumulator in der Zero-Page gerettet wird. Anschließend wird das P-Register vom Stack in den Akkumulator kopiert, um das BRK-Bit zu isolieren. Falls dieses tatsächlich gesetzt war, wird zum Label BREAK verzweigt. Nachdem die Routine SAV1 alle Register in der Zero-Page gespeichert hat, wird auch noch die Rücksprungadresse vom Stack genommen und ebenfalls in der Zero-Page abgelegt.
Der abschließende indirekte Sprung JMP ($03F0) ist der für uns wichtigste Teil dieser Routine. Er verlagert nämlich die Kontrolle über die weiteren Ereignisse aus dem ROM-Bereich heraus in den RAM-Bereich. Im Normalzustand befindet sich in den beiden Speicherzellen $03F0 und $03F1 die Adresse desjenigen Unterprogramms, welches für die Ausgabe der Register und den Sprung in den Monitor zuständig ist. Doch kann man natürlich auch sehr einfach die Adresse einer eigenen Routine eintragen, welche gar nichts mit der üblichen Bedeutung der BRK-Anweisung zu tun haben muß. Da alle Register und der Programmzähler zum Zeitpunkt der Unterbrechung gerettet wurden, kann diese Routine eine beliebige Aufgabe erfüllen und nach ihrer Beendigung zum unterbrochenen Programm zurückkehren.
Bevor eine derartige Anwendung vorgestellt wird, soll noch einmal kurz der Ablauf nach Erkennen eines BRKs zusammengefaßt werden:
Trifft der 6502-Prozessor bei der Ausführung eines Programms auf einen BRK-Befehl, so setzt er zunächst das BRK-Bit seines P-Registers. Danach wird so verfahren, als ob ein IRQ-Impuls die CPU erreicht hätte. Dies bedeutet, daß der Inhalt des Programmzählers (Adresse vom BRK plus 2) und das P-Register auf dem Stack abgelegt werden. Über den Vektor in $FFFE und $FFFF wird dann die Verarbeitung mit der IRQ-Routine des Autostart-ROMs (auf $FA40) fortgesetzt. Zusätzlich zum P-Register und dem Programmzähler, welche beide vom Stack entfernt werden, sichert dieser Code auch alle übrigen Register in der Zero-Page. Schließlich wird dann die Kontrolle an ein Programm übergeben, dessen Anfangsadresse sich in $03F0 und $03F1 befinden muß.

3. Aufgabenstellung

Normalerweise wird bei der Erstellung eines Maschinenprogramms eine feste Startadresse eingeplant. Für Utilities hingegen ist es sehr vorteilhaft, wenn diese Adresse auch nachträglich noch verändert werden kann. Ein Benutzer ist nämlich oftmals daran interessiert, eine Utility in für ihn günstige Bereiche zu laden und zu starten.
Um dies zu ermöglichen, dürfen bei der Programmierung nur solche Anweisungen benutzt werden, welche die Adresse einer Speicherzelle nicht absolut festlegen. Im Befehlssatz des 6502 trifft dies lediglich auf die bedingten Sprünge zu. Für den Transfer von Daten zwischen CPU und Speicher kennt der 6502 leider gar keine Befehle mit der Adressierungsart „relativ".
Im folgenden soll nun eine Utility vorgestellt werden, mit deren Hilfe es möglich ist, alle Befehle, welche eine absolute Adresse erwarten, auch mit einer relativen Adresse auszustatten. So soll dann ein LDA $1234 nicht mehr den Inhalt von $1234, sondern den von $1234 + Programmzähler in den Akkumulator laden. In der **Abbildung 2** sind alle in Frage kommenden Befehle mit den zugehörigen Op-Codes aufgelistet. Wenn darin ein Eintrag fehlt (z.B. für BIT adr,X), so liegt dies daran, daß der 6502 über eine solche Anweisung nicht verfügt.
Es ist natürlich nicht möglich, den 6502 tatsächlich mit neuen Op-Codes für die

PEEKER Börse

Gelegenheitsanzeigen / Kleinanzeigen

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 (nicht gewerblich)

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 ☐ gewerblich
☐ biete Hardware ☐ biete Software ☐ Chiffre ☐ nicht gewerblich

Bitte den Text mit Schreibmaschine oder in Druckbuchstaben ausfüllen

Jeweils 32 Buchstaben pro Zeile – einschl. Satzzeichen und Wortzwischenräume. Bitte Absender nicht vergessen. Mindestens 2 Zeilen (1 Druckzeile à 32 Buchstaben DM 5,50 nicht gewerblich, DM 11,– gewerblich + MwSt.) – Chiffregebühr DM 6,– + MwSt.

PEEKER

Produkt-Karte

Zu der in **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

_____

_____

_____

_____

_____

_____

_____

Ich interessiere mich für Beiträge über

☐ Apple ☐ IBM ☐ Atari ☐ _____

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

Bitte freimachen

POSTKARTE

Peeker-Börse
Anzeigen-Service

Dr. Alfred Hüthig Verlag

Postfach 10 28 69

6900 Heidelberg 1

Bitte freimachen

POSTKARTE

Inserent

Straße

PLZ/Ort

Bitte freimachen

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.

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

Damit erleichtern Sie dem Hersteller eine gezielte Beantwortung Ihrer Anfrage.

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

PEEKER

Abbildung 2

	adr	adr,X	adr,Y	(adr)
ADC	6D	7D	79	
AND	2D	3D	39	
ASL	ØE	1E		
BIT	2C			
CMP	CD	DD	D9	
CPX	EC			
CPY	CC			
DEC	CE	DE		
EOR	4D	5D	59	
INC	EE	FE		
JMP	4C			6C
JSR	2Ø			
LDA	AD	BD	B9	
LDX	AE		BE	
LDY	AC	BC		
LSR	4E	5E		
ORA	ØD	1D	19	
ROL	2E	3E		
ROR	6E	7E		
SBC	ED	FD	F9	
STA	8D	9D	99	
STX	8E			
STY	8C			

relative Adressierung auszustatten. Dies wäre nur durch einen Neu-Entwurf des Prozessors zu schaffen. Wir müssen uns daher mit einer softwaretechnischen Lösung begnügen. Aber dies ist jetzt auch der Punkt, wo der BRK-Befehl seine Fähigkeiten unter Beweis stellen kann. So soll nämlich ein BRK vor einer Anweisung mit einer absoluten Adresse deren Adressierungsart in „relativ" umändern. Betrachten wir als Beispiel einmal den folgenden Code.

```
NOP
LDA $4321
BRK
STA $1234
NOP
```

Diese Anweisungsfolge holt zunächst den Inhalt der Speicherzelle $4321 in das A-Register. Danach soll dann nicht etwa das Programm gestoppt, sondern vielmehr der Akkumulator in derjenigen Speicherzelle abgelegt werden, welche $1234 entfernt ist. Die zu erstellende Utility muß daher in der Lage sein, die einem BRK folgende Anweisung zu analysieren und diese dann mit der korrekten, absoluten Adresse auszuführen.
Doch zunächst muß erst einmal festgelegt werden, worauf eine relative Adresse bezogen wird. Es bietet sich z.B. die Adresse des BRKs oder aber die der folgenden Anweisung an. Ich habe bezüglich dieser Frage eine Entscheidung getroffen, welche mit der Behandlung von bedingten Sprüngen in Einklang steht. Gibt man etwa im Monitor den Befehl 300:10 00 ein, so wird dieser Code als BPL $0302 disassembliert. Man sieht also, daß die relative Adresse bei den Branch-Befehlen immer bezüglich der nächsten Anweisung gedeutet wird. Ich habe dieses Prinzip übernommen, so daß die absolute Adresse des obigen Beispiels zu $1234 plus Adresse des zweiten NOPs berechnet werden kann. Läge dieses NOP auf $1000, so müßte der Akkumulator in $2234 gespeichert werden.

Nehmen wir einmal an, daß meine BRK-Routine bereits in den BRK-Vektor ($03F0 und $03F1) eingehängt wurde. Welche Aufgaben muß sie dann erfüllen? Die Rettung des Programmzählers und der Register wurde bereits vom Autostart-ROM erledigt, so daß sie sich nur noch um ihre eigentliche Aufgabe zu kümmern hat. Und zwar muß sie zuerst einmal den dem BRK folgenden Op-Code in einen ihr zur Verfügung stehenden Bereich kopieren. Danach wird sie aus der relativen Adresse und dem gesicherten Programmzähler eine absolute Adresse errechnen und diese hinter dem Op-Code abspeichern. Nachdem alle zum Zeitpunkt des BRKs vorliegenden Registerinhalte wieder restauriert wurden, kann dann diese generierte Anweisung ausgeführt und zum unterbrochenen Programm (Adresse des BRKs plus 4) zurückgekehrt werden.

Eine gewisse Sonderbehandlung muß bei einem JSR-Befehl mit relativer Adresse erfolgen. Denn selbst wenn die absolute Adresse des Unterprogramms berechnet wurde, kann nicht einfach ein JSR-Sprung dorthin ausgeführt werden. Dadurch würde nämlich als Rücksprungadresse eine Adresse auf dem Stack abgelegt, welche innerhalb des Codes für die BRK-Behandlung läge. Beim nächsten RTS soll aber doch hinter die JSR-Anweisung (also in das Anwenderprogramm) zurückgekehrt werden. Aus diesem Grund muß die Ablage der Rücksprungadresse auf dem Stack von der BRK-Utility explizit erledigt und zum Unterprogramm dann nur mittels eines JMPs verzweigt werden.

4. Implementation

Der Code für meinen BRK-Prozessor (s. Listing **BRK**) besitzt eine Länge von etwa 180 Bytes, so daß er sehr gut in den $0300-Bereich paßt. Er wurde aber so geschrieben, daß er bei jeder Adresse der Form $xy00 gestartet werden kann. Diese Möglichkeit konnte natürlich nicht aus dem Trick mit dem BRK-Befehl bezogen werden. Vielmehr mußte dieser Code „echt" verschiebbar sein.

Aus der Zero-Page werden für den Zeiger BEF_PTR die beiden Speicherzellen 0 und 1 benötigt, die aber immer gerettet werden. MON_PC bezeichnet diejenige Stelle, in welcher vom Autostart-ROM der Programmzähler abgelegt wurde. Die Speicherzellen, die für die übrigen Register benutzt wurden, müssen meiner Utility nicht direkt bekannt sein. Der Monitor stellt nämlich eine Routine (MON_REST) zur Verfügung, welche deren Inhalte zurück in die Register holt. MON_BRK ist dasjenige Programm, welches normalerweise nach einem BRK aufgerufen wird. Ich benutze es, wenn dem BRK ein ungültiger Op-Code folgt, so daß in diesem Fall das BRK tatsächlich zum Anhalten verwendet wird.

Die Zeilen 35-43 stellen ein kurzes Unterprogramm dar, welches den BRK-Vektor mit der Anfangsadresse von BRK_PROZ belegt. Also bevor BRK_PROZ benutzt werden kann, muß diese Routine einmal aufgerufen werden. Die Zeilen 38-41 sind der übliche Trick, um herauszubekommen, in welche Seite die Utility geladen wurde (s. z.B. „Apple II Reference Manual", Seite 81).

In Zeile 50 beginnt nun die eigentliche Utility, die immer dann die Kontrolle erhält, wenn vom Prozessor ein BRK ausgeführt werden sollte. Die Zeilen 50-53 sichern die beiden Speicherzellen der Zero-Page, indem sie deren Inhalt auf dem Stack ablegen.

Es wurde bereits erwähnt, daß BRK_PROZ eine Anweisung aufbauen muß. Dieses wird ab dem Label BEFEHL (Zeile 229) geschehen. Um einen Zeiger auf diese Stelle zu haben, wird in den Zeilen 59-63 der Pointer BEF_PTR mit dieser Adresse initialisiert.

Auf Grund der beschriebenen Eigenart des 6502 gibt MON_PC die Adresse des BRKs plus 2 an. Dies wird in den Zeilen 68-73 korrigiert, indem von MON_PC 2 subtrahiert wird. Falls es später tatsächlich zu einem Sprung in den Monitor kommen sollte, wird daher die wirkliche Adresse des BRKs ausgegeben.

Indem in den Zeilen 83-86 zu dieser Adresse 4 hinzuaddiert wird, erhält man die Adresse derjenigen Anweisung, bei welcher im Normalfall (kein relativer JMP oder JSR) nach der Simulation fortgefahren werden soll. Diese Adresse wird in den Zeilen 91-96 in das Adreßfeld eines JMP-Befehls gespeichert, welcher im Anschluß an den aufgebauten Befehl ausgeführt wird.

Die Zeilen 100-103 kopieren den dem BRK folgenden Op-Code des Anwenderprogramms in die BRK_PROZ-Routine. Daran anschließend muß überprüft werden, ob überhaupt einer der Codes aus der Abbildung 2 vorliegt. Dies wird von den Zeilen 108-134 erledigt. Diese sind sicherlich einfacher zu verstehen, wenn nicht die Abbildung 2, sondern die **Abbildung 3** angesehen wird. In dieser übersichtlicheren Darstellung wurden alle zulässigen Codes mit einem „*" markiert.

Abbildung 3

Ihr entnimmt man, daß die Codes im wesentlichen nach einem Schema aufgebaut sind. Ausnahmen von dieser Regel, welche in den Zeilen 108-118 abgefragt werden, wurden in der Abbildung 3 durch senkrechte Striche kenntlich gemacht.
Sofern ein ungültiger Code angetroffen wurde, wird in Zeile 139 zur BRK-Behandlung des Monitors verzweigt. Diese gibt den Programmzähler sowie die Register (jeweils aus der Zero-Page) aus und endet dann im Monitor.
Andernfalls wird in 144 der Op-Code gesichert, da er später noch einmal benötigt wird. Zwar liegt wegen der ANDs nicht mehr unbedingt der ursprüngliche Op-Code vor, doch reicht auch dieser modifizierte Wert noch aus. Es wird später nämlich lediglich ein Test auf Gleichheit mit $20 durchgeführt. Die Zeilen 150-155 holen dann die dem Op-Code folgende relative Adresse, und 160-167 addieren die Rücksprungadresse, um so die absolute Adresse zu erhalten. Diese beiden Bytes werden anschließend durch die Zeilen 172-177 in dem Operandenfeld der erzeugten Anweisung abgelegt.

Falls nun nicht gerade eine JSR-Anweisung aufgebaut wurde, sind im wesentlichen alle Aufgaben erledigt. Die Zeilen 181-184 stellen dies fest und löschen als Signal dafür das Carry-Bit.
Wenn aber doch ein JSR vorgelegen hat, ist eine besondere Behandlung erforderlich: Zunächst einmal wird in den Zeilen 188-190 der JSR-Code in einen JMP-Code umgewandelt. Dann wird in 195-202 die auf dem Stack abzulegende Information über die Rücksprungadresse (eigentliche Rücksprungadresse minus 1, Adresse des BRKs plus 3) berechnet. Da sich aber oben auf dem Stack noch die geretteten Speicherzellen 0 und 1 befinden, dürfen die beiden Werte noch nicht abgelegt werden. Als Zeichen dafür, daß die X- und Y-Register diese Information enthalten, wird in 204 das Carry-Bit gesetzt.

Die Zeilen 208-211 stellen den vorgefundenen Zustand der Zero-Page wieder her. Und falls der Op-Code tatsächlich ein JSR gewesen ist, wird in 219-222 die Rücksprungadresse für das nächste RTS auf den Stack geschrieben. Abschließend wird dann in 227-231 der Zustand zum Zeitpunkt des BRKs restauriert, die generierte Anweisung ausgeführt und zum Anwenderprogramm zurückgesprungen.

Da der Programmfluß nicht ganz einfach ist, will ich die entscheidenden Anweisungen noch einmal kurz zusammenfassen. Angenommen, das Anwenderprogramm enthält die folgenden Zeilen:

```
        BRK
        Opc Adr
ZURUECK NOP
```

Wenn nun Opc weder ein JMP noch ein JSR ist (also z.B. ein LDA, EOR, ROR usw.), wird die Anweisung mit der absoluten Adresse bei BEFEHL aufgebaut und ausgeführt. Der anschließende Sprung wurde zuvor auf das Label ZURUECK gerichtet.
Für den Fall, daß Opc ein JMP ist, werden die 6 Bytes zwar auch wie oben mit den entsprechenden Werten belegt, der Sprung auf BEFEHL + 3 gelangt jedoch niemals zur Ausführung, da bereits ein anderer auf BEFEHL erfolgt.
Und falls Opc ein JSR ist, wird das Byte in BEFEHL in einen JMP-Code umgewandelt. Das normalerweise vom JSR selbst erledigte Ablegen der Rücksprungadresse wird in diesem Fall von der BRK-Routine übernommen.

5. Ein Beispiel

Angenommen, der BRK-Prozessor wurde an eine Adresse der Form $xy00 geladen und die Prozedur BRK_INIT danach einmal aufgerufen, dann kann das ebenfalls abgedruckte Programm BRK-Prozessor-Test (s. Listing **BRK.TEST**) an irgendeine Adresse geladen und dort gestartet werden. Es gibt einige Texte aus und berechnet die Summe zweier Zahlen. Dabei werden eine Reihe von Befehlen mit einer relativen Adresse ausgeführt (am rechten Rand jeweils durch „<<<“ markiert). Sehen wir uns das einmal etwas genauer an:

In den Zeilen 16-17 findet man schon direkt eine Anwendung der BRK-Utility. Die Länge des relativen Sprungs beträgt $0082 Bytes, die von dem Assembler durch die Differenz START - ENTRY_1 berechnet wird.
Anstatt nun hinter jede Anweisung mit einer relativen Adresse ein weiteres Label zu schreiben, wurde ein eleganterer Weg gewählt. In der Zeile 109 soll etwa zur Routine PRT_TXT gesprungen werden. Wenn man nun aber beachtet, daß „*“ dort den Wert $8089 besitzt, so kann die Differenz durch PRT_TXT - (* + 3) berechnet werden. Löst man dann noch die Klammer auf, so ergibt sich die dort angegebene relative Adresse.

Es ist darauf zu achten, daß der Assembler bei den Op-Codes, die von BRK_PROZ bearbeitet werden sollen, die 3 Byte langen Befehle wählt. Beim verwendeten Merlin-Assembler wurde dies durch Anhängen des „>“ an den Op-Code erreicht. Wenn also die relative Adresse kleiner oder gleich $00FF ist, darf auf keinen Fall die Zero-Page-Adressierung erzeugt werden. Der entsprechende Op-Code würde von der BRK-Utility als unzulässig zurückgewiesen werden.

Kurzhinweise

1. Zweck:
BRK-Utility für die Bearbeitung absoluter Adressen in nicht-relokativen Programmen.
2. Konfiguration:
II+ sowie bedingt IIe/c mit neuen ROMs; DOS 3.3 oder ProDOS
3. Test:
BRUN BRK
BLOAD BRK.TEST
CALL -151
8000G
4. Sammeldisk:
T.BRK (Big-Mac-Quelltext)
BRK
T.BRK.TEST (Big-Mac-Quelltext)
BRK.TEST

BRK

```
BSAVE BRK, A$300,L177
                        1           ORG  $300
                        4   ****************************************
                        5   *                                      *
                        6   *        BRK-PROZESSOR                 *
                        7   *                                      *
                        8   *        Michael G. Schneider          *
                        9   *                                      *
                       14   ****************************************
                       15
                       16   BEF_PTR  EQU  Ø
                       17
                       18   MON_PC   EQU  $3A
                       19
                       20   JSR_CODE EQU  $20
                       21   JMP_CODE EQU  $4C
                       22
                       23   SYS_STCK EQU  $100
                       24
                       25   BRK_VKTR EQU  $3FØ
                       26
                       27   MON_BRK  EQU  $FA59
                       28   MON_REST EQU  $FF3F
                       29   MON_RTS  EQU  $FF58
                       30   *-------------------------------------
                       31
                       32   * Initialisiere den BRK-Vektor mit der
                       33   * Adresse von BRK_PROZ.
                       34
Ø300: A9 1Ø            35   BRK_INIT LDA  #BRK_PROZ
Ø302: 8D FØ Ø3         36            STA  BRK_VKTR
                       37
Ø305: 2Ø 58 FF         38            JSR  MON_RTS
Ø308: BA               39            TSX
Ø309: BD ØØ Ø1         4Ø            LDA  SYS_STCK,X
Ø30C: 8D F1 Ø3         41            STA  BRK_VKTR+1
                       42
Ø30F: 6Ø               43            RTS
                       44   *-------------------------------------
                       45
                       46   * Sichere den Inhalt von Ø und 1 auf dem
                       47   * System-Stack. Diese Speicherplätze
                       48   * werden für BEF_PTR gebraucht.
                       49
Ø31Ø: A5 ØØ            5Ø   BRK_PROZ LDA  BEF_PTR
Ø312: 48               51            PHA
Ø313: A5 Ø1            52            LDA  BEF_PTR+1
Ø315: 48               53            PHA
                       54
                       55   * Laß BEF_PTR auf diejenige Adresse
                       56   * in BRK_PROZ zeigen, wo die Anweisung
                       57   * aufgebaut werden soll.
                       58
Ø316: A9 AB            59            LDA  #BEFEHL
Ø318: 85 ØØ            6Ø            STA  BEF_PTR
                       61
Ø31A: AD F1 Ø3         62            LDA  BRK_VKTR+1
Ø31D: 85 Ø1            63            STA  BEF_PTR+1
                       64
                       65   * Stelle die Adresse des soeben ausge-
                       66   * führten BRKs fest ...
                       67
Ø31F: A6 3B            68            LDX  MON_PC+1
Ø321: A5 3A            69            LDA  MON_PC
Ø323: 38               7Ø            SEC
Ø324: E9 Ø2            71            SBC  #2
Ø326: BØ Ø1            72            BCS  BRK_PRZ1
Ø328: CA               73            DEX
                       74
                       75   * ... und laß MON_PC darauf zeigen.
                       76
Ø329: 85 3A            77   BRK_PRZ1 STA  MON_PC
Ø32B: 86 3B            78            STX  MON_PC+1
                       79
                       8Ø   * Addiere 4 hinzu und erhalte so die
                       81   * Adresse des übernächsten Befehls, ...
                       82
Ø32D: 18               83            CLC
Ø32E: 69 Ø4            84            ADC  #4
Ø33Ø: 9Ø Ø1            85            BCC  BRK_PRZ2
Ø332: E8               86            INX
                       87
                       88   * ... die dann in das Adreßfeld der JMP-
                       89   * Anweisung gespeichert wird.
                       9Ø
Ø333: AØ Ø4            91   BRK_PRZ2 LDY  #4
Ø335: 91 ØØ            92            STA  (BEF_PTR),Y
                       93
Ø337: C8               94            INY
Ø338: 8A               95            TXA
Ø339: 91 ØØ            96            STA  (BEF_PTR),Y
                       97
                       98   * Kopiere den Op-Code hinter dem BRK ...
                       99
Ø33B: AØ Ø1           1ØØ            LDY  #1
Ø33D: B1 3A           1Ø1            LDA  (MON_PC),Y
Ø33F: 88              1Ø2            DEY
Ø34Ø: 91 ØØ           1Ø3            STA  (BEF_PTR),Y
                      1Ø4
                      1Ø5   * ... und überprüfe, ob dieser erlaubt ist
                      1Ø7
Ø342: C9 2Ø           1Ø8            CMP  #$2Ø
Ø344: FØ 23           1Ø9            BEQ  GOOD_CDE
                      11Ø
Ø346: C9 ØC           111            CMP  #$ØC
Ø348: FØ 1C           112            BEQ  BAD_CDE
                      113
Ø34A: C9 BC           114            CMP  #$BC
Ø34C: FØ 1B           115            BEQ  GOOD_CDE
                      116
Ø34E: C9 9E           117            CMP  #$9E
Ø35Ø: FØ 14           118            BEQ  BAD_CDE
                      119
Ø352: 29 1F           12Ø            AND  #%ØØØ11111
                      121
Ø354: C9 19           122            CMP  #$19
Ø356: FØ 11           123            BEQ  GOOD_CDE
                      124
Ø358: C9 ØC           125            CMP  #$ØC
Ø35A: FØ ØD           126            BEQ  GOOD_CDE
                      127
Ø35C: 29 ØF           128            AND  #%ØØØØ1111
                      129
Ø35E: C9 ØD           13Ø            CMP  #$ØD
Ø36Ø: FØ Ø7           131            BEQ  GOOD_CDE
                      132
Ø362: C9 ØE           133            CMP  #$ØE
Ø364: FØ Ø3           134            BEQ  GOOD_CDE
                      135
                      136   * Es liegt ein ungültiger Code vor, also
                      137   * halte das Programm an.
                      138
Ø366: 4C 59 FA        139   BAD_CDE  JMP  MON_BRK
                      14Ø
                      141   * Er ist gültig. Sichere ihn erst einmal
                      142   * auf dem Stack.
                      143
Ø369: 48              144   GOOD_CDE PHA
                      145
                      147   * Hole die relative Adresse in die
                      148   * Register <A,X> und ...
                      149
Ø36A: AØ Ø3           15Ø            LDY  #3
Ø36C: B1 3A           151            LDA  (MON_PC),Y
Ø36E: AA              152            TAX
                      153
Ø36F: 88              154            DEY
Ø37Ø: B1 3A           155            LDA  (MON_PC),Y
                      156
                      157   * ... addiere die Rücksprungadresse, um
                      158   * die absolute Adresse zu erhalten.
                      159
Ø372: 18              16Ø            CLC
Ø373: AØ Ø4           161            LDY  #4
Ø375: 71 ØØ           162            ADC  (BEF_PTR),Y
Ø377: 48              163            PHA
                      164
Ø378: 8A              165            TXA
Ø379: C8              166            INY
Ø37A: 71 ØØ           167            ADC  (BEF_PTR),Y
                      168
                      169   * Lege die absolute Adresse dann in das
                      17Ø   * Operandenfeld der erzeugten Anweisung.
                      171
Ø37C: AØ Ø2           172            LDY  #2
Ø37E: 91 ØØ           173            STA  (BEF_PTR),Y
                      174
Ø38Ø: 68              175            PLA
Ø381: 88              176            DEY
Ø382: 91 ØØ           177            STA  (BEF_PTR),Y
                      178
                      179   * War der Befehl etwa ein JSR ?
                      18Ø
Ø384: 68              181            PLA
Ø385: 18              182            CLC
Ø386: 49 2Ø           183            EOR  #JSR_CODE
Ø388: DØ 12           184            BNE  EXIT_PRP
                      185
```

```
                    186  * Ändere das JSR in ein JMP um, ...
                    187
038A: A0 00         188  JSR_BHDG LDY  #0
038C: A9 4C         189           LDA  #JMP_CODE
038E: 91 00         190           STA  (BEF_PTR),Y
                    191
                    192  * ... berechne die Rücksprungadresse
                    193  * minus 1 und sichere sie in <X,Y>.
                    194
0390: 18            195           CLC
0391: A5 3A         196           LDA  MON_PC
0393: 69 03         197           ADC  #3
0395: AA            198           TAX
                    199
0396: A5 3B         200           LDA  MON_PC+1
0398: 69 00         201           ADC  #0
039A: A8            202           TAY
                    203
039B: 38            204           SEC
                    205
                    206  * Stelle die alte Zero-Page wieder her.
                    207
039C: 68            208  EXIT_PRP PLA
039D: 85 01         209           STA  BEF_PTR+1
039F: 68            210           PLA
03A0: 85 00         211           STA  BEF_PTR
                    212
03A2: 90 04         213           BCC  EXIT
                    214
                    215  * Der Befehl war ein JSR, also muß jetzt
                    216  * die Rücksprungadresse auf dem Stack
                    217  * abgelegt werden.
                    218
03A4: 98            219           TYA
03A5: 48            220           PHA
03A6: 8A            221           TXA
03A7: 48            222           PHA
                    223
                    224  * Restauriere den Zustand zum Zeitpunkt
                    225  * des BRKs.
                    226
03A8: 20 3F FF      227  EXIT     JSR  MON_REST
                    228
                    229  BEFEHL   DS   3
                    230
03AE: 4C FF FF      231           JMP  $FFFF
                    232  *-------------------------------------
```

BRK.TEST

```
BSAVE BRK.TEST, A$8000,L152

                      1           ORG  $8000
                      4  ****************************************
                      5  *                                      *
                      6  *        BRK-PROZESSOR-TEST            *
                      7  *                                      *
                      8  *        Michael G. Schneider          *
                      9  *                                      *
                     14  ****************************************
                     15
8000: 00             16  ENTRY    BRK
8001: 4C 82 00       17           JMP  START-ENTRY_1        ;<<<
                     18  ENTRY_1
                     19  *-------------------------------------
                     20
                     21  MON_CR   EQU  $FD8E
                     22  MON_PRHX EQU  $FDDA
                     23  MON_COUT EQU  $FDED
                     24  *-------------------------------------
                     25
8004: 3F 46          26  WERTE    HEX  3F46
                     27  SUMME    DS   1
                     28
                     29  TXT
8007: 8D 8D          30  TXT_ANF  HEX  8D8D
8009: C4 E9 E5       31           ASC  "Dies ist ein BRK-Test"
800C: F3 A0 E9 F3 F4 A0 E5 E9
8014: EE A0 C2 D2 CB AD D4 E5
801C: F3 F4
801E: 8D 8D 00       32           HEX  8D8D00
                     33
8021: C4 E9 E5       34  TXT_SU_0 ASC  "Die Summe von "
8024: A0 D3 F5 ED ED E5 A0 F6
802C: EF EE A0
802F: 00             35           HEX  00
8030: A0 F5 EE       36  TXT_SU_1 ASC  " und "
8033: E4 A0
8035: 00             37           HEX  00
8036: A0 E9 F3       38  TXT_SU_2 ASC  " ist "
8039: F4 A0
803B: 00             39           HEX  00
                     40  *-------------------------------------
                     41
                     42  * Gib den Text aus, dessen Index sich im
                     43  * Y-Register befindet.
                     44
803C: 00             45  PRT_TXT  BRK
803D: B9 C7 FF       46           LDA> TXT-*-3,Y            ;<<<
8040: F0 06          47           BEQ  PRNT_END
8042: 20 ED FD       48           JSR  MON_COUT
8045: C8             49           INY
8046: D0 F4          50           BNE  PRT_TXT           ;immer !
                     51
8048: 60             52  PRNT_END RTS
                     53  *-------------------------------------
                     54
                     55  * Stelle die zwei zu addierenden Zahlen
                     56  * vor.
                     57
8049: A0 1A          58  TEST_SU  LDY  #TXT_SU_0-TXT
804B: 00             59           BRK
804C: 20 ED FF       60           JSR> PRT_TXT-*-3          ;<<<
                     61
804F: A2 00          62           LDX  #0
8051: 00             63           BRK
8052: BD AF FF       64           LDA> WERTE-*-3,X          ;<<<
8055: 20 DA FD       65           JSR  MON_PRHX
                     66
8058: A0 29          67           LDY  #TXT_SU_1-TXT
805A: 00             68           BRK
805B: 20 DE FF       69           JSR> PRT_TXT-*-3          ;<<<
                     70
805E: A2 01          71           LDX  #1
8060: 00             72           BRK
8061: BD A0 FF       73           LDA> WERTE-*-3,X          ;<<<
8064: 20 DA FD       74           JSR  MON_PRHX
                     75
8067: A0 2F          76           LDY  #TXT_SU_2-TXT
8069: 00             77           BRK
806A: 20 CF FF       78           JSR> PRT_TXT-*-3          ;<<<
                     79
                     80  * Addiere die beiden Zahlen und leg das
                     81  * Ergebnis im Speicher ab.
                     82
806D: A2 00          83           LDX  #0
806F: 00             84           BRK
8070: BD 91 FF       85           LDA> WERTE-*-3,X          ;<<<
                     86
8073: 18             87           CLC
8074: E8             88           INX
8075: 00             89           BRK
8076: 7D 8B FF       90           ADC> WERTE-*-3,X          ;<<<
                     91
8079: E8             92           INX
807A: 00             93           BRK
807B: 9D 86 FF       94           STA> WERTE-*-3,X          ;<<<
                     95
                     96  * Gib die Summe aus.
                     97
807E: 00             98           BRK
807F: AD 84 FF       99           LDA> SUMME-*-3            ;<<<
8082: 20 DA FD      100           JSR  MON_PRHX
                    101
8085: 60            102           RTS
                    103  *-------------------------------------
                    104
                    105  * Drucke den Begrüßungstext.
                    106
8086: A0 00         107  START    LDY  #TXT_ANF-TXT
8088: 00            108           BRK
8089: 20 B0 FF      109           JSR> PRT_TXT-*-3          ;<<<
                    110
                    111  * Addiere die beiden Zahlen.
                    112
808C: 00            113           BRK
808D: 20 B9 FF      114           JSR> TEST_SU-*-3          ;<<<
                    115
                    116  * Gib noch zwei Returns aus.
                    117
8090: 20 8E FD      118           JSR  MON_CR
8093: 20 8E FD      119           JSR  MON_CR
                    120
                    121  * Verzweige in den Monitor.
                    122
8096: 00            123           BRK
8097: 00            124           BRK
                    125  *-------------------------------------
```

Kaufen
Verkaufen
Kontakte knüpfen

Hüthig PUBLIKATION

PEEKER Börse

bietet Ihnen einen kostengünstigen und erfolgversprechenden Weg, mit einer Kleinanzeige an Ihr Ziel zu gelangen. Problemlos und schnell mit dem anhängenden Coupon.

PEEKER Börse

Anzeigenauftrag

GELEGENHEITSANZEIGEN
KLEINANZEIGEN

Bitte veröffentlichen Sie in der nächstmöglichen Ausgabe für mich folgenden Text als

- ☐ private/nicht gewerbliche Kleinanzeige (mindestens 2 Druckzeilen, DM 5,50 je Zeile zuzügl. Mwst.)
- ☐ gewerbliche Kleinanzeige (mindestens 2 Druckzeilen, DM 11,– je Zeile zuzügl. Mwst.)
  Gewerbliche Anzeigen müssen wir aus wettbewerbsrechtlichen Gründen mit G kennzeichnen.
- ☐ unter Chiffre (Chiffre-Gebühr DM 6,– zuzügl. Mwst.)

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

Preise privat nicht gewerblich +Mwst.	Preise gewerblich +Mwst.
Zeilen	Zeilen
1 DM 5,50	1 DM 11,–
2 DM 11,–	2 DM 22,–
3 DM 16.50	3 DM 33,–
4 DM 22,–	4 DM 44,–
5 DM 27,50	5 DM 55,–
6 DM 33,–	6 DM 66,–
zuzüglich evtl. Chiffregebühr	

Eine Druckzeile umfaßt 32 Buchstaben. Die ersten Wörter Ihrer Anzeige drucken wir in **fetter** Schrift. Nach jedem Wort ein Kästchen freilassen. Jede angefangene Zeile wird als ganze berechnet. Falls Name/Anschrift/Telefon-Nr. in der Anzeige erscheinen sollen, tragen Sie bitte auch diese Angaben in die Kästchen ein. Bitte den Text mit Schreibmaschine oder mit Druckbuchstaben eintragen.

Anzeigenschluß für Heft 3/87 am 2.2.87

Rubriken:
- ☐ suche Hardware
- ☐ biete Hardware
- ☐ suche Software
- ☐ biete Software
- ☐ verschiedenes

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

1 Zeile
2 Zeilen
3 Zeilen
4 Zeilen
5 Zeilen
6 Zeilen

Meine Anschrift:

Name/Vorname Telefon

Straße PLZ/Ort Unterschrift

Leistungsfähiger Pattern-Matcher für Applesoft-Strings

von Dipl.-Inform. Dieter Greiner

1. Was ist Pattern-Matching?

Pattern-Matching (PM) ist ein Verfahren, bei dem versucht wird, einen Ausdruck (Muster, Pattern) durch Ersetzen der darin enthaltenen Variablen an einen anderen Ausdruck, der keine Variablen enthält, anzugleichen (gleichmachen, matchen). Dazu ein kurzes Beispiel: Wir wollen die Ausdrücke f(X,X,1) und f(2,2,1) matchen, wobei X im ersten Ausdruck eine Variable bezeichnen soll. Dazu wird die Variable X im ersten Ausdruck durch „2" ersetzt. Der Match-Prozeß wäre jedoch erfolglos verlaufen, hätte der 2. Ausdruck f(2,1,1) gelautet, denn erstes und zweites Argument sind verschieden (2<>1); deshalb können die beiden Ausdrücke nicht angeglichen werden.
Besondere Bedeutung haben PM-Verfahren durch ihre Anwendungen in der Künstlichen Intelligenz (KI; Artificial Intelligence = AI) erlangt. Beispiele:
– Maschinelle Sprachverarbeitung: Erkennung von Satzmustern
– Intelligente Datenbanksysteme: Anfragen werden als Muster mit Variablen formuliert
– Expertensysteme: Mustererkennung spielt eine wichtige Rolle in der sog. Inferenzmaschine
– Logikprogrammiersprachen, Produktionssysteme und Theorembeweiser: Dort trifft man meist das leistungsfähigste PM-Verfahren an, nämlich die sog. Unifikation. (Bei der Unifikation dürfen beide anzugleichenden Ausdrücke Variablen enthalten.)

2. Pattern-Matching in Applesoft

Hier sind anstelle von Ausdrücken zwei normale BASIC-Strings gegeben, die gematcht werden sollen. Der erste String, der auch *Muster (Pattern)* genannt wird, kann neben ganz gewöhnlichen ASCII-Zeichen auch noch 4 Arten von speziell gekennzeichneten Variablen enthalten:

1. Segmentvariablen (Kennzeichen: „#")
2. anonyme Segmentvariablen (Kennzeichen: „*")
3. Elementvariablen (Kennzeichen: „/")
4. anonyme Elementvariablen (Kennzeichen: „?")

Auf nicht-anonyme Variablen sollte nach dem „#" oder „/" eine Ziffer zwischen 0 und 9 folgen, die die Nummer der Variablen angibt. Wird die Ziffer weggelassen, setzt das Programm die Nummer 0 ein. Beispiele für Muster sind:

```
"ABC#1"     : #1 ist Segmentvariable
"/Ø#1#1"    : /Ø ist Elementvariable und
              #1 Segmentvariable
"/Ø"        : / ist die Elementvariable
              mit der Nummer Ø
"ABC*E"     : * ist anonyme Segmentvariable
"AB??E"     : String enthält 2 anonyme
              Elementvariablen
```

Der zweite String ist als reine Zeichenkette zu betrachten, deren Zeichen keine besondere Bedeutung haben. Dieser String wird im folgenden *Matchstring* genannt, weil er gegen das Muster „gematcht" wird.
Das Problem beim PM besteht nun darin, alle Variablen im Muster so zu ersetzen, daß Muster und Matchstring gleich werden. Dabei kann wegen der fehlenden Numerierung jede der anonymen Variablen im Muster durch einen anderen String ersetzt werden. Für die Ersetzung gelten folgende Regeln:
1. Segmentvariablen dürfen durch einen beliebig langen, evtl. auch leeren String ersetzt werden.
2. Elementvariablen dürfen nur durch einen String der Länge 1, d.h. durch ein einzelnes Zeichen ersetzt werden.
Ist obiges Problem lösbar, so sagt man auch kurz, daß Muster und Matchstring gegeneinander „matchen".

3. Aufruf von Applesoft aus

Der Pattern-Matcher erhält 3 oder 4 Argumente, das vierte Argument ist optional. Von Applesoft aus wird der Pattern-Matcher mit folgender Anweisung aufgerufen:

```
CALL 379Ø1, <Muster>, <Matchstring>,
            <Matchresultat>,
            <Name eines Stringarrays>
            (= optional)
```

Erläuterung:
<Muster> ist der Name einer Stringvariablen, die das unter Punkt 2 beschriebene Muster enthält.
<Matchstring> ist der Name einer Stringvariablen, die den unter Punkt 2 beschriebenen Matchstring enthält.
<Matchresultat> ist der Namen einer Integervariablen, die nach Rückkehr aus dem Pattern-Matcher den Wert 1 enthält, falls <Muster> und <Matchstring> matchen; sonst enthält sie den Wert 0.

<Name eines Stringarrays> ist der Name eines eindimensionalen Stringarrays, das mindestens 10 Arrayelemente enthalten und vor dem Aufruf dimensioniert worden sein muß. Falls <Muster> und <Matchstring> matchen, enthält dieser Array die Ersetzungen für die nicht-anonymen, numerierten Segment- und Elementvariablen, wobei der i-te Eintrag im Stringarray der Ersetzung der Segment/Elementvariablen mit der Nummer i entspricht. Das Argument (einschließlich des vorausgehenden Kommas) ist optional und muß nur angegeben werden, wenn in <Muster> numerierte, d.h. nicht-anonyme Variablen vorkommen.

4. Programmeingabe und Fehlermeldungen

Das Programm wurde mit Hilfe des Big-Mac-Assemblers von G. Bredon unter dem Betriebssystem DOS 3.3 auf einem Apple II+ erstellt. Es enthält außer dem „=“-Zeichen, das bei anderen Assemblern durch „EQU“ und „EPZ“ (bei LISA 2.5) zu ersetzen ist, keine außergewöhnlichen Assemblerdirektiven oder Opcodes (z.B. vom 65C02). Falls der Hex-Code direkt eingegeben wird, muß dies im Monitor ab Adresse $9400 erfolgen. (Die Eingabe von Hex-Codes wurde im Peeker-Doppelheft 12/84-1/85 im Artikel „Ampersand macht's möglich“ von Dr. Jürgen B. Kehrel ausführlich beschrieben.) Nach der Eingabe im Monitor oder nach erfolgter Assemblierung wird das Programm durch „BSAVE MATCH, A$9400, L509“ auf Diskette abgespeichert.
Vor der ersten Benutzung muß der Initialisierungsteil des Programms aufgerufen werden, um es durch Versetzen des HIMEM-Pointers vor Zerstörung zu schützen. Da der Initialisierungsteil am Programmanfang liegt, kann das Laden des Programms und seine Initialisierung mit dem Befehl BRUN MATCH erledigt werden. (Der Initialisierungsteil liegt unter HIMEM und wird daher durch angelegte Strings zerstört.) Wichtig ist, daß die Programminitialisierung vorgenommen wird, *bevor* der BASIC-Interpreter Speicherplatz für irgendwelche Stringvariablen beansprucht.

Das Array, das dem Pattern-Matcher im CALL-Aufruf als Argument übergeben werden kann, sollte zuvor dimensioniert werden, weil sonst ein OUT OF DATA ERROR erfolgt. Außerdem meldet der Rechner einen ILLEGAL QUANTITY ERROR, wenn das Array nicht eindimensional ist oder so dimensioniert wurde, daß es weniger als 10 Elemente umfaßt.
Wurde dem Pattern-Matcher durch den CALL-Aufruf kein Stringarray als drittes Argument übergeben und kommen im Muster dennoch numerierte Variablen vor, so bricht das Programm mit der Fehlermeldung SYNTAX ERROR ab.
Der Pattern-Matcher arbeitet rekursiv und benötigt zur Sicherung von Rücksprungadressen und Speicherplatzinhalten den 6502-Hardwarestack ($100-$1FF). Falls dem Programm unmittelbar vor einem rekursiven Aufruf auf dem Stack nicht mehr als 12 Bytes zur Verfügung stehen, wird die Kontrolle dem BASIC-Errorhandler übergeben, der die Ausgabe der Fehlermeldung FORMULA TOO COMPLEX veranlaßt. Wurde das Error-Trapping nicht zuvor mit „ON ERROR GOTO“ angeschaltet, bricht das laufende BASIC-Programm ab.

5. Beispiele

Die Tabelle (s. u.) gibt eine Übersicht über die Leistungsfähigkeit des Pattern-Matchers. Sämtliche Angaben beziehen sich auf den Aufruf „CALL 37901, A$,B$,RES%,C$“, wobei das Array C$ mit „DIM C$(9)“ dimensioniert sein soll.

6. Arbeitsweise des Programms

Vor Aufruf des Pattern-Matchers sind alle Segment- und Elementvariablen ungebunden, d.h. sie haben keinen Wert. Ungebundene Stringvariablen sind dadurch gekennzeichnet, daß ihre 3 Deskriptorenkomponenten (Länge, Zeiger auf Zeichenkette low und high) Null sind. Dies ist sinnvoll, weil der BASIC-Interpreter Zeichenketten nie in der Zero-Page abspeichert und somit ungebundene Variablen eindeutig und leicht anhand der dritten Deskriptorenkomponente erkannt werden können. Im Laufe des Matching-Prozesses werden Segment- und Elementvariablen an Teilstrings des Matchstrings gebunden und wieder entbunden (ungebunden gemacht), wenn sich mit einer Bindung keine Angleichung des Musters und des Matchstrings erzielen läßt. All dies geschieht nach einem systematischen Verfahren, das *Backtracking* (Zurücksetzverfahren) heißt und im folgenden anhand der Behandlung von Segmentvariablen im Pattern-Matcher näher erklärt werden soll.
Angenommen, wir wollen das Muster „#0“ gegen den Matchstring „HALLO“ matchen. Obwohl dieses Beispiel sehr einfach aussieht, verdeutlicht es, wie das Backtrackingverfahren funktioniert.
Eine wichtige Rolle im Programm spielen die beiden Variablen P1 und P2, die stets die Position im Muster bzw. Matchstring angeben, bis zu der Muster und Matchstring bereits gematcht sind. Beim ersten Aufruf von MATCH ist P1 = P2 = 0, weil noch nichts gematcht wurde. Anschließend probiert das Programm aus, wie lang das Segment ab Position P2 im Matchstring sein muß, an das die Segmentvariable #0 gebunden wird, damit auch der Reststring nach der Segmentvariablen im Muster und der Reststring im Matchstring matchen. Das Programm versucht es zuerst mit einem Segment der Länge Null und ruft dazu den Pattern-Matcher rekur-

A$	B$	RES%	C$
"AABB"	"AABB"	1	–
""	""	1	–
"*.BAS"	"PRO.BAS"	1	–
"APPLE"	"DIE APPLE-RECHNER"	1	–
"7???"	"7056"	1	–
"?A?"	"BAP"	1	–
"*.BAS"	"PRO.BAS"	1	C$(0)="PRO"
"#0#1#1#0"	"ANNA"	1	C$(0)="A" C$(1)="N"
"/0/1/1/0"	"ANNA"	1	C$(0)="A" C$(1)="N"
"#0#1#2#0#2#1"	"ABCDEFGHIABCGHIDEF"	1	C$(0)="ABC" C$(1)="DEF" C$(2)="GHI"
"#0#1/2/2A#0"	"ABCDXXAAB"	1	C$(0)="AB" C$(1)="CD" C$(2)="X"
"/*/"	"ANNA"	1	C$(0)="A"
"#1#2-#2—#1-#1"	"ABCDEF-DEF--ABC-ABC"	1	C$(1)="ABC" C$(2)="DEF"
"/#1A#2/?/"	"X111A222X X"	1	C$(0)="X" C$(1)="111" C$(2)="222"
"#1/#2/"	"ANNA"	1	C$(0)="A" C$(1)="" C$(2)="NN"
"AAB"	"AABB"	0	
"???"	"AABB"	0	
"##"	"DAD"	0	
"/////"	"NICHT"	0	

siv mit den Positionen P1'=P1+2 (#0-Segmentvariable ist gematcht → Überspringen des Strings „#0") und P2'=P2 auf (weil die Segmentvariable zuerst gegen ein Segment der Länge Null gematcht wird). Kehrt der Pattern-Matcher erfolgreich aus seinem rekursiven Aufruf zurück (Zero-Flag = 1), so weiß er, daß Muster und Matchstring ab P1 und P2 matchen, und es wird aus der aktuellen MATCH-Inkarnation wiederum Erfolg gemeldet. In unserem Beispiel hätte der Pattern-Matcher keinen Erfolg, da im rekursiven Aufruf der (aussichtslose) Versuch unternommen wird, die Reststrings „" und „ALLO" zu matchen. Deswegen setzt das Programm zurück („backtrackt"), d.h. es belegt die Variablen P1 und P2 mit ihren alten Werten, die vor dem rekursiven Aufruf auf dem Stack gesichert wurden, bindet die Segmentvariable #0 an ein Segment der Länge 1 ab Position P2 und ruft erneut den Pattern-Matcher auf, diesmal mit den Positionen P1'=P1+2 und P2'=P2+1 (weil jetzt die Segmentvariable an das erste Zeichen ab Position P2 im Matchstring gebunden ist). Dieser Aufruf wird wieder Mißerfolg melden, weil das Programm jetzt versucht, die Strings „" und „LLO" zu matchen.
Dieser rekursive Aufruf von MATCH und das anschließende Austesten eines um 1 längeren Segments wiederholt sich solange, bis sowohl P1 als auch P2 am Ende des Musters bzw. Matchstrings angelangt sind, d.h. die Strings „" und „" gematcht werden, was zu einem Erfolg führt. (Zu diesem Zeitpunkt ist die Segmentvariable #0 an den String „HALLO" gebunden). Dieser Erfolg wiederum wird in die nächsthöhere Programmebene zurückgeliefert (Rücksprung mit RTS) und führt bei unserem Beispiel zum Setzen der dem Programm übergebenen Integer-Erfolgsvariablen.

Beim Backtracking kann es mitunter vorkommen, daß das Programm keine Möglichkeiten mehr hat, die Segmentvariable an ein noch längeres Segment des Matchstrings zu binden, nämlich dann, wenn P2 + Segmentlänge > Länge des Matchstrings ist. Ist dies der Fall, so wird die Segmentvariable entbunden und die aktuelle Inkarnation von MATCH meldet Mißerfolg.
Natürlich muß eine Segmentvariable anders behandelt werden, wenn sie bereits an ein Segment gebunden ist. In diesem Falle wird zunächst überprüft, ob das Segment, an das die Segmentvariable gebunden ist, und der Teilstring gleicher Länge ab Position P2 im Matchstring gleich sind. Wenn nicht, wird das Zeroflag gelöscht, um Mißerfolg anzuzeigen und mit RTS in die aufrufende Routine zurückgesprungen. Andernfalls wird P1 um 1 bzw. 2 (falls eine numerierte Segmentvariable vorliegt), P2 um die Segmentlänge erhöht und an den Anfang von MATCH gesprungen, wo von neuem ab den Positionen P1 und P2 gematcht wird.

7. Einsatzgebiete des Programms

Als Einsatzgebiete des Pattern-Match-Programms kommen in Frage:

1. Dialogprogramme: Das Paradebeispiel für den Einsatz von PM-Verfahren ist das ursprünglich in LISP geschriebene Dialogprogramm ELIZA von J.Weizenbaum, bei dem sich der Benutzer einem Computerpsychiater anvertrauen kann. Das Programm erkennt dabei mit Hilfe eines Pattern-Matchers vom Benutzer eingegebene Frage- und Antwortmuster und kann (oft) mit passenden Antworten und Gegenfragen reagieren.

2. Chemische Datenbanken: Ein Chemiker interessiert sich z.B. dafür, bei welchen Molekülen zwischen zwei XYZ-Gruppen zwei weitere gleiche Gruppen vorkommen. Dazu muß man nur das Muster „*XYZ*/1#*/1#*XYZ*" gegen alle in der Datenbank gespeicherten Moleküle matchen und im Falle eines Matcherfolgs das entsprechende Molekül ausgeben. (Die Kombination /1# ist notwendig, weil leere Gruppenbezeichnungen unsinnig sind.)

3. Assembler: Eine Assembler-Sourcezeile hat meist den Aufbau

<Label> <Mnemonic> <Operand> ;<Kommentar>

Dieses Muster kann mit Hilfe des Pattern-Matchers einfach erkannt werden, indem man die Sourcezeile gegen das Muster „/5#1 /6#2 #3 ;#4" matcht. (Die Elementvariablen sind notwendig, weil Labelnamen und Mnemonics keine Leerstrings sein dürfen.) Bei Matcherfolg stellt die Konkatenation (Elementvariable 5 + Segmentvariable 1) den Labelnamen, die Konkatenation (Elementvariable 6 + Segmentvariable 2) das Mnemonic, die Segmentvariable 3 den Operanden (evtl. leer) und die Segmentvariable 4 den Kommentartext dar. Dem Autor ist ein in BASIC geschriebener MC68000-Assembler bekannt, der PM zum Parsen des Quellcodes verwendet.

4. Dateidienstprogramme: Ein gutes Beispiel hierfür ist das File-Verwaltungsprogramm FID, das auf der DOS-3.3-Systemdiskette mitgeliefert wird und die Verwendung von Wildcardzeichen (Jokerzeichen) erlaubt. Diese Zeichen entsprechen unseren anonymen Segmentvariablen. Wildcardzeichen sind auch bei den Betriebssystemen UCSD-Pascal und CP/M zur Spezifizierung von Filenamen erlaubt.

5. INSTRING: Mit dem Pattern-Matcher kann auch der INSTRING-Befehl nachgebildet werden. Angenommen, es soll überprüft werden, ob der String „XYZ" im String A$ enthalten ist. Das kann mit der folgenden Befehlsfolge geschehen:

```
10 DIM C$(9)
20 B$="*"+"XYZ"+"*"
30 CALL 37901,B$,A$,RES%,C$
```

Falls der String „XYZ" in A$ enthalten ist, enthält die Integervariable RES% nach Abarbeitung der Zeile 30 den Wert 1, wobei LEN(C$(0))+1 die Position angibt, an der „XYZ" erstmals in A$ gefunden wurde.

Das waren nur einige Beispiele, die zeigen, wo sich Pattern-Matching vorteilhaft einsetzen läßt. Interessierte Peeker-Leser werden sicher noch mehr Anwendungsmöglichkeiten finden. Viel Spaß beim Matchen!

Kurzhinweise

1. Zweck:
Pattern-Matching (Musterabgleichung) für Strings in Applesoft
2. Konfiguration:
Apple II+/e/c; DOS 3.3 (kein ProDOS wegen HIMEM-Änderung)
3. Test:
RUN MATCH.TEST
4. Sammeldisk:
MATCH.TEST (Demo-Programm)
T.MATCH (Big-Mac-Quelltext)
MATCH (Objektcode)

MATCH.TEST

```
10 PRINT CHR$ (4)"BRUN MATCH,A$9400"
15 MTCH = 37888 + 13
20 DIM VB$(9): REM Array mit Variablenbindungen
25 INPUT "Muster: ";MU$
30 INPUT "Matchstring: ";MA$
35 CALL MTCH,MU$,MA$,MR%,VB$
40 PRINT
45 IF NOT (MR%) THEN PRINT "Strings matchen nicht!":
   PRINT : PRINT : GOTO 25
50 REM  MR%=1, Strings MU$ und MA$ matchen
55 PRINT "Strings matchen !!!": PRINT :
   PRINT "Variablenbindungen: ": PRINT
60 FOR I = 0 TO 9: PRINT I;": ";VB$(I): NEXT I
65 PRINT : PRINT : GOTO 25
```

T.MATCH

```
BSAVE MATCH, A$9400,L509
```

```
                    1   ********************************
                    2   *                              *
                    3   * PATTERN-MATCHER FUER APPLE-   *
                    4   *         SOFT-STRINGS         *
                    6   * von:     D. Greiner 1985     *
                   13   *------------------------------*
                   14   *                              *
                   15   * Installation:                *
                   17   * BRUN MATCH, A37888           *
                   18   *                              *
                   19   *------------------------------*
                   20   *                              *
                   21   * Aufrufformat:                *
                   22   *                              *
                   23   * CALL 379Ø1,<MU>,<MT>,<MR>,<A>*
                   24   *                              *
                   25   * <MU> Stringmuster, das Var-  *
                   26   *      iablen enthalten darf   *
                   27   * <MT> String, gegen den <MU>  *
                   28   *      gematcht werden soll    *
                   29   * <MR> Name einer Integer-Var- *
                   30   *      iablen, die das Match-  *
                   31   *      resultat (Ø oder 1) ent-*
                   32   *      hält                    *
                   33   * <A>  Optionales Argument. Na-*
                   34   *      me eines Stringarrays,  *
                   35   *      das nach erfolgreichem  *
                   36   *      Matchen die Variablen-  *
                   37   *      bindungen enthält       *
                   38   *                              *
                   39   ********************************
                   40
                   41            ORG  $9400      ;37888
                   42
                   43   * Applesoft-Routinen
                   44
                   45   CHRGOT   EQU  $B7
                   46   CHRGET   EQU  $B1
                   47   CHKCOM   EQU  $DEBE
                   48   PTRGET   EQU  $DFE3
                   49   GETARYP  EQU  $F7D9
                   50   SYNERR   EQU  $DEC9
                   51   ILQNTERR EQU  $E199
                   52   ERROR    EQU  $D412
                   53
                   54   * Applesoft-Zeigervariable
                   55
                   56   FRETOP   EQU  $6F
                   57   HIMEM    EQU  $73
                   58   VARPNT   EQU  $83
                   59   ARRPNT   EQU  $9B
                   60
                   61   * Programmvariablen
                   62
                   63   L1       EQU  $06
                   64   MU       EQU  $07        ;<MU>
                   65   P1       EQU  $09
                   66   SL       EQU  $19
                   67   ST       EQU  $1A
                   68   VAROFFST EQU  $1C
                   69   SAVESP   EQU  $1D
                   70   L2       EQU  $F9
                   71   MT       EQU  $FA        ;<MT>
                   72   P2       EQU  $FC
                   73   SAVE     EQU  $FD
                   74   NUMSALLD EQU  $FE
                   75
                   76   * Variablenkennzeichnungen
                   77
                   78   SEGVARCH EQU  '*'
                   79   ELVARC   EQU  '/'
                   80   ASEGVARC EQU  '*'
                   81   AELVARC  EQU  '?'
                   82
                   83   * Setzen des HIMEM-Pointers zum
                   84   * Schutz des Hauptprogramms
                   85   * (INIT wird später durch
                   86   *  Strings überschrieben)
                   87
9400: A9 0D        88   INIT     LDA  #ENTRY
9402: 85 73        89            STA  HIMEM
9404: 85 6F        90            STA  FRETOP
9406: A9 94        91            LDA  #ENTRY/256
9408: 85 74        92            STA  HIMEM+1
940A: 85 70        93            STA  FRETOP+1
940C: 60           94            RTS
                   95
                   96   * Einlesen der Argumente und
                   97   * Plausibilitätstests
                   98
940D: BA           99   ENTRY    TSX             ;Rette
940E: 86 1D       100            STX  SAVESP     ;SP
9410: 20 BE DE    101            JSR  CHKCOM     ;Komma!
                  102
                  103   * Lies 1. Argument <MU>, Zeiger
                  104   * auf <MU> steht nachher in
                  105   * (MU,MU+1), Länge in L1
                  106
9413: 20 E3 DF    107            JSR  PTRGET
9416: A0 00       108            LDY  #0
9418: B1 83       109            LDA  (VARPNT),Y
941A: 85 06       110            STA  L1
941C: C8          111            INY
941D: B1 83       112            LDA  (VARPNT),Y
941F: 85 07       113            STA  MU
9421: C8          114            INY
9422: B1 83       115            LDA  (VARPNT),Y
9424: 85 08       116            STA  MU+1
9426: 20 BE DE    117            JSR  CHKCOM     ;Komma!
                  118
                  119   * Lies 2. Argument <MT>, Zeiger
                  120   * auf <MU> steht nachher in
                  121   * (MT,MT+1), Länge in
                  122   * L2
                  123
9429: 20 E3 DF    124            JSR  PTRGET
942C: A0 00       125            LDY  #00
942E: 84 09       126            STY  P1
9430: 84 FC       127            STY  P2
9432: 84 FE       128            STY  NUMSALLD
9434: B1 83       129            LDA  (VARPNT),Y
9436: 85 F9       130            STA  L2
9438: C8          131            INY
9439: B1 83       132            LDA  (VARPNT),Y
943B: 85 FA       133            STA  MT
943D: C8          134            INY
943E: B1 83       135            LDA  (VARPNT),Y
9440: 85 FB       136            STA  MT+1
9442: 20 BE DE    137            JSR  CHKCOM     ;Komma!
                  138
                  139   * Lies 3. Argument <MR>, welches
                  140   * ersatzweise den Wert Ø (<=>
                  141   * kein Match) hat.
                  142
9445: 20 E3 DF    143            JSR  PTRGET
9448: A9 00       144            LDA  #0         ;No Mtch
944A: A8          145            TAY
944B: 91 83       146            STA  (VARPNT),Y
944D: C8          147            INY
944E: 91 83       148            STA  (VARPNT),Y
9450: 20 B7 00    149            JSR  CHRGOT
                  150
                  151   * Optionales Argument <A> ange-
                  152   * geben?
                  153   * Ja: Lies Argument, Setze
                  154   *     (ARRPNT,ARRPNT+1) auf 1.
                  155   *     Arrayelement
                  156   * Nein: Start des Matchers
                  157
9453: C9 2C       158            CMP  #','
9455: D0 36       159            BNE  START
9457: 85 FE       160            STA  NUMSALLD   ;A<>0!
                  161
                  162   * <A>-Argument angegeben ->
                  163   * Numerierte Variable erlaubt!
                  164
9459: 20 B1 00    165            JSR  CHRGET
945C: 20 D9 F7    166            JSR  GETARYP
945F: A0 04       167            LDY  #4
9461: B1 9B       168            LDA  (ARRPNT),Y
9463: C9 01       169            CMP  #1         ;DIM=1?
9465: D0 0C       170            BNE  ILLEGAL
```

```
9467: C8          171           INY
9468: B1 9B       172           LDA  (ARRPNT),Y
946A: D0 0A       173           BNE  OK
946C: C8          174           INY
946D: B1 9B       175           LDA  (ARRPNT),Y
946F: C9 0A       176           CMP  #10
9471: B0 03       177           BCS  OK
9473: 4C 99 E1    178 ILLEGAL   JMP  ILQNTERR
9476: A9 07       179 OK        LDA  #7
9478: 18          180           CLC
9479: 65 9B       181           ADC  ARRPNT
947B: 85 9B       182           STA  ARRPNT
947D: 90 02       183           BCC  UNBIND
947F: E6 9C       184           INC  ARRPNT+1
                  185
                  186 * Entbinden der ersten 10
                  187 * Variablen von <A>. Index=0..9
                  188
9481: A0 00       189 UNBIND    LDY  #0
9483: A2 00       190           LDX  #0
9485: 20 94 95    191 UNBIND1   JSR  UNBNDVBL
9488: E8          192           INX
9489: E0 0A       193           CPX  #10
948B: D0 F8       194           BNE  UNBIND1
                  195
                  196 * Top-Level-Aufruf des
                  197 * Pattern-Matchers
                  198
948D: 20 99 94    199 START     JSR  MATCH
9490: D0 06       200           BNE  NM
9492: A9 01       201           LDA  #1          ;Match
9494: A0 00       202           LDY  #0          ;war
9496: 91 83       203           STA  (VARPNT),Y ;mögl.
9498: 60          204 NM        RTS
                  205
                  206 *********************************
                  207 *                               *
                  208 * MATCH ist der eigentliche     *
                  209 * Pattern-Matcher. MATCH kehrt  *
                  210 * mit Zeroflag=1 zurück, falls  *
                  211 * <MU> und <MT> ab den Positi-  *
                  212 * onen P1 (in <MU>) und P2 (in  *
                  213 * <MT>) matchen                 *
                  214 *                               *
                  215 *********************************
                  216
9499: A4 09       217 MATCH     LDY  P1
949B: C4 06       218           CPY  L1
949D: D0 05       219           BNE  NOTEND
949F: A5 FC       220           LDA  P2
94A1: C5 F9       221           CMP  L2
94A3: 60          222           RTS
                  223
                  224 * Noch nicht am Ende von <MU>.
                  225 * Accu:=Akt. Zeichen von <MU>
                  226
94A4: B1 07       227 NOTEND    LDA  (MU),Y
94A6: C9 23       228           CMP  #SEGVARCH
94A8: F0 54       229           BEQ  SEGMENT
94AA: C9 2A       230           CMP  #ASEGVARC
94AC: D0 05       231           BNE  NOSEGVAR
94AE: E6 09       232           INC  P1
94B0: 4C 73 95    233           JMP  BCKTRCK2
                  234
                  235 * Keine Segmentvariable. Wenn
                  236 * in <MT> bereits am Ende, kein
                  237 * Match
                  238
94B3: A6 FC       239 NOSEGVAR LDX  P2
94B5: E4 F9       240          CPX  L2
94B7: 90 03       241          BCC  TEST
94B9: A9 01       242 NOMATCH  LDA  #1          ;Kein
94BB: 60          243          RTS              ;Match!
                  244
                  245 * Untersuche weiter, ob in <MU>
                  246 * Elementvariable vorliegt.
                  247
94BC: C9 2F       248 TEST      CMP  #ELVARC
94BE: F0 10       249           BEQ  ELEMENT
94C0: C9 3F       250           CMP  #AELVARC
94C2: F0 06       251           BEQ  TEST2
                  252
                  253 * Für Match müssen Zeichen in
                  254 * <MU> und <MT> gleich sein.
                  255
94C4: A4 FC       256 TEST1     LDY  P2
94C6: D1 FA       257           CMP  (MT),Y
94C8: D0 EF       258           BNE  NOMATCH
94CA: E6 09       259 TEST2     INC  P1          ;Check
94CC: E6 FC       260 TEST3     INC  P2          ;next!
94CE: D0 C9       261           BNE  MATCH       ;immer!
                  262
```

```
                  263 * Behandlung von Elementvar.
                  264
94D0: 20 A2 95    265 ELEMENT  JSR  GETOFFST
94D3: F0 0C       266          BEQ  EVNOTBND
                  267
                  268 * Elementvar. ist gebunden. Vgl.
                  269 * erstes Zeichen mit aktuellem
                  270 * Zeichen in <MT>. Für Match
                  271 * müssen beide gleich sein
                  272
94D5: A0 00       273          LDY  #0
94D7: B1 1A       274          LDA  (ST),Y
94D9: A4 FC       275          LDY  P2
94DB: D1 FA       276          CMP  (MT),Y
94DD: F0 EB       277          BEQ  TEST2       ;OK!
94DF: D0 D8       278          BNE  NOMATCH     ;BAD!
                  279
                  280 * Elementvar. ist nicht geb.
                  281 * Binde Variable und rufe
                  282 * MATCH rekursiv mit dieser
                  283 * Bindung auf. Falls Match
                  284 * nicht erfolgreich ist, ent-
                  285 * binde Variable wieder und
                  286 * melde "No Match"
                  287
94E1: A9 01       288 EVNOTBND LDA  #1
94E3: A4 1C       289          LDY  VAROFFST
94E5: 20 DD 95    290          JSR  BINDVBL
94E8: A5 1C       291          LDA  VAROFFST
94EA: 48          292          PHA
94EB: E6 09       293          INC  P1
94ED: E6 FC       294          INC  P2
94EF: 20 EF 95    295          JSR  CHKDEPTH
94F2: 20 99 94    296          JSR  MATCH
94F5: F0 5D       297          BEQ  GOOD1
94F7: 68          298          PLA
94F8: A8          299          TAY
94F9: 20 94 95    300          JSR  UNBNDVBL
94FC: F0 BB       301          BEQ  NOMATCH
                  302
                  303 * Behandlung von num. Segment-
                  304 * variablen
                  305
94FE: 20 A2 95    306 SEGMENT  JSR  GETOFFST
9501: F0 2D       307          BEQ  SVNOTBND
                  308
                  309 * Behandle gebundene Segment-
                  310 * variable. Überprüft wird,
                  311 * ob das Segment der Länge SL
                  312 * ab Position P2 in <MT> gleich
                  313 * dem gebundenen Segmentstring
                  314 * ist. Wenn nicht, liefere "No
                  315 * Match"
                  316
9503: 18          317          CLC
9504: A5 19       318          LDA  SL
9506: 65 FC       319          ADC  P2
9508: F0 02       320          BEQ  WITHIN
950A: B0 AD       321          BCS  NOMATCH
950C: A5 1A       322 WITHIN   LDA  ST
950E: 8D 28 95    323          STA  PL+1
9511: A5 1B       324          LDA  ST+1
9513: 8D 29 95    325          STA  PL+2
9516: A2 00       326          LDX  #0
9518: A4 FC       327          LDY  P2
                  328
                  329 * Vergleichsschleife zum Vergl.
                  330 * obiger Stringsegmente
                  331
951A: E4 19       332 CPLOOP   CPX  SL
951C: D0 07       333          BNE  PL1
951E: 84 FC       334 MM       STY  P2
9520: E6 09       335          INC  P1
9522: 4C 99 94    336          JMP  MATCH
9525: B1 FA       337 PL1      LDA  (MT),Y
9527: DD FF FF    338 PL       CMP  $FFFF,X
952A: D0 8D       339          BNE  NOMATCH
952C: E8          340 ON       INX
952D: C8          341          INY
952E: D0 EA       342          BNE  CPLOOP
                  343
                  344 * Behandlung nichtgebundener
                  345 * num. Segmentvar. Prinzip:
                  346 *  B A C K T R A C K I N G
                  347
9530: A2 00       348 SVNOTBND LDX  #0
9532: A4 1C       349          LDY  VAROFFST
9534: 98          350 BCKTRCK1 TYA
9535: 48          351          PHA
9536: 8A          352          TXA
9537: 20 DD 95    353          JSR  BINDVBL
                  354
```

```
                        355  * Rette auf Stack (wegen
                        356  * rek. Aufruf v. MATCH):
                        357  * P1: Position in <MU>
                        358  * P2: Position in <MT>
                        359  * X-Reg.: Segmentlänge
                        360
953A: A5 09             361          LDA  P1
953C: 48                362          PHA
953D: A5 FC             363          LDA  P2
953F: 48                364          PHA
9540: 8A                365          TXA
9541: 48                366          PHA
9542: 18                367          CLC
9543: 65 FC             368          ADC  P2
9545: 85 FC             369          STA  P2
9547: E6 09             370          INC  P1
9549: 20 EF 95          371          JSR  CHKDEPTH
954C: 20 99 94          372          JSR  MATCH
954F: D0 07             373          BNE  BT
                        374
                        375  * Match!
                        376
9551: 68                377          PLA
9552: 68                378          PLA
9553: 68                379  GOOD    PLA
9554: 68                380  GOOD1   PLA
9555: A9 00             381          LDA  #0          ;Z=1!
9557: 60                382          RTS
                        383
                        384  * Rek. Aufruf lieferte
                        385  * "No Match" -> Stelle
                        386  * alte Werte von P1,
                        387  * P2 und X-Reg. wieder
                        388  * her, probiert, falls
                        389  * möglich, ein läng-
                        390  * eres Segment (X-Reg.
                        391  * erhöhen) ab Pos.
                        392  * P2 in <MT>
                        393
9558: 68                394  BT      PLA
9559: AA                395          TAX
955A: 68                396          PLA
955B: 85 FC             397          STA  P2
955D: 68                398          PLA
955E: 85 09             399          STA  P1
9560: 68                400          PLA
                        401
                        402  * Y-Reg erhält Offset
                        403  * der Segment-Variablen
                        404
9561: A8                405          TAY
9562: E8                406          INX
9563: 8A                407          TXA
9564: 18                408          CLC
9565: 65 FC             409          ADC  P2
9567: C5 F9             410          CMP  L2
9569: 90 C9             411          BCC  BCKTRCK1
956B: F0 C7             412          BEQ  BCKTRCK1
956D: 20 94 95          413          JSR  UNBNDVBL
                        414
                        415  * Alle Möglichkeiten für
                        416  * Segment in <MT> er-
                        417  * schöpft. Kein Erfolg->
                        418  * liefere "No Match"
                        419
9570: A9 01             420          LDA  #1          ;Z=0!
9572: 60                421          RTS
                        422
                        423  * Backtracking zur Behandlung
                        424  * anonymer Segmentvariablen
                        425  * Vgl. Code für Behandlung
                        426  * numerierter Segmentvar.
                        427
9573: A5 09             428  BCKTRCK2 LDA  P1
9575: 48                429          PHA
9576: A5 FC             430          LDA  P2
9578: 48                431          PHA
9579: 20 EF 95          432          JSR  CHKDEPTH
957C: 20 99 94          433          JSR  MATCH
957F: F0 D2             434          BEQ  GOOD
9581: 68                435          PLA
9582: 85 FC             436          STA  P2
9584: 68                437          PLA
9585: 85 09             438          STA  P1
9587: E6 FC             439          INC  P2
9589: A6 FC             440          LDX  P2
958B: E4 F9             441          CPX  L2
958D: 90 E4             442          BCC  BCKTRCK2
958F: F0 E2             443          BEQ  BCKTRCK2
9591: A9 01             444  NO      LDA  #1          ;Z=0!
9593: 60                445          RTS
```

```
                        446
                        447  * Entbinde die Variable mit
                        448  * Offset Y in <A>
                        449
9594: A9 00             450  UNBNDVBL LDA  #0
9596: 91 9B             451           STA  (ARRPNT),Y
9598: C8                452           INY
9599: 91 9B             453           STA  (ARRPNT),Y
959B: C8                454           INY
959C: 91 9B             455           STA  (ARRPNT),Y
959E: C8                456           INY
959F: A9 00             457           LDA  #0
95A1: 60                458           RTS
                        459
                        460  * Liefert in VAROFFST den
                        461  * Offset der numerierten Var.,
                        462  * auf die P1 zeigt. (ST,ST+1)
                        463  * zeigt auf den Stringtext
                        464  * dieser Variablen, SL gibt
                        465  * die Länge des Stringtexts
                        466  * an. Falls die Variable unge-
                        467  * bunden ist, kehrt GETOFFST
                        468  * mit Zeroflag=1 zurück.
                        469
95A2: A5 FE             470  GETOFFST LDA  NUMSALLD
95A4: D0 06             471           BNE  ALLOWED
95A6: A6 1D             472           LDX  SAVESP
95A8: 9A                473           TXS
95A9: 4C C9 DE          474           JMP  SYNERR
95AC: E6 09             475  ALLOWED  INC  P1
95AE: A4 09             476           LDY  P1
95B0: C4 06             477           CPY  L1
95B2: B0 07             478           BCS  DEFVAL
95B4: B1 07             479           LDA  (MU),Y
95B6: 38                480           SEC
95B7: E9 30             481           SBC  #'0'
95B9: B0 07             482           BCS  GR0
95BB: C6 09             483  DEFVAL   DEC  P1
95BD: A9 00             484           LDA  #0
95BF: 18                485           CLC
95C0: 90 09             486           BCC  SOFFS
95C2: C9 0A             487  GR0      CMP  #10
95C4: B0 F5             488           BCS  DEFVAL
95C6: 85 FD             489           STA  SAVE
95C8: 0A                490           ASL
95C9: 65 FD             491           ADC  SAVE
95CB: A8                492  SOFFS    TAY
95CC: 85 1C             493           STA  VAROFFST
95CE: B1 9B             494           LDA  (ARRPNT),Y
95D0: 85 19             495           STA  SL
95D2: C8                496           INY
95D3: B1 9B             497           LDA  (ARRPNT),Y
95D5: 85 1A             498           STA  ST
95D7: C8                499           INY
95D8: B1 9B             500           LDA  (ARRPNT),Y
95DA: 85 1B             501           STA  ST+1
95DC: 60                502  EXIT     RTS
                        503
                        504  * Weise an die Variable im
                        505  * Array <A> mit Offset Y das
                        506  * Segment ab Pos. P2 in <MT>
                        507  * mit Länge=<ACCU> zu
                        508
95DD: 91 9B             509  BINDVBL  STA  (ARRPNT),Y
95DF: C8                510           INY
95E0: 18                511           CLC
95E1: A5 FC             512           LDA  P2
95E3: 65 FA             513           ADC  MT
95E5: 91 9B             514           STA  (ARRPNT),Y
95E7: A9 00             515           LDA  #0
95E9: 65 FB             516           ADC  MT+1
95EB: C8                517           INY
95EC: 91 9B             518           STA  (ARRPNT),Y
95EE: 60                519           RTS
                        520
                        521  * Überprüfe, ob noch genügend
                        522  * Platz für rekursiven Aufruf
                        523  * auf dem Stack vorhanden ist
                        524
95EF: BA                525  CHKDEPTH TSX
95F0: E0 0A             526           CPX  #10
95F2: B0 08             527           BCS  TIEFEOK
95F4: A6 1D             528           LDX  SAVESP
95F6: 9A                529           TXS
95F7: A2 BF             530           LDX  #191
                        531
                        532  * X-Reg enthält Code für
                        533  * "FORMULA TOO COMPLEX"
                        534
95F9: 4C 12 D4          535           JMP  ERROR
95FC: 60                536  TIEFEOK  RTS
```

Long-Integer-Routinen in UCSD-Pascal

von Thomas Schneider

1. Anwendungsgebiet

Das UCSD-Programm LONGINT unterstützt die Ein- und Ausgabe von Festkommazahlen auf der Basis des im UCSD-System implementierten Datentyps Long Integer.
Das Programm bewährt sich vor allem bei Anwendungen, bei denen es auf die Genauigkeit von Beträgen ankommt, die bei Real-Zahlen nicht gegeben ist. Es ist deshalb im gesamten kaufmännischen Bereich von großem praktischem Nutzen. Bei der Eingabe wurden umfangreiche Plausibilitätskontrollen vorgesehen, so daß damit auch Programme für absolute EDV-Laien erstellt werden können.

2. Programmaufbau

Die in dieser Unit zusammengefaßten Routinen ermöglichen die Ein-/Ausgabe von sog. Long-Integer-Zahlen. Sie sind lauffähig unter dem Apple UCSD-Pascal-System, Version 1.1.
Long-Integer-Zahlen haben im Gegensatz zu den normalen Integer-Zahlen im Pascal-System eine maximale Größe von 36 Dezimalziffern und ermöglichen damit genauere Rechnungen als Real-Zahlen – allerdings nur im ganzzahligen Bereich. Verwendet man aber Festkommazahlen, die bei der Addition und Subtraktion denselben Gesetzen wie die ganzen Zahlen gehorchen, so ist es möglich, intern mit der kleinsten vorkommenden Einheit zu rechnen, wenn nur die Ein/Ausgabe im gewohnten Festkommaformat erfolgt. So ist es insbesondere im kaufmännischen Bereich empfehlenswert, intern mit Pfennigen zu rechnen, die Beträge jedoch in gewohnter Weise in DM und Pfennigen auszugeben, z.B. DM 1,24.
Dies ist genau die Funktion, die durch die Routinen realisiert wird.

Entwickelt wurden die Routinen bei der Implementierung eines Programms zur Kontoverwaltung, nachdem klar wurde, daß normale Integer-Zahlen zu klein (mit DM 325,75 kann ich gerade 1 1/2 Monate BAFöG ausrechnen) und Real-Zahlen zu ungenau sind.
Die Routinen wurden in einer Unit organisiert, um sie nach Einbinden in die Systembibliothek SYSTEM.LIBRARY immer verfügbar zu haben. Natürlich ist es auch möglich, sie direkt in ein Programm einzubauen. Dazu muß man beachten, daß folgende Bezeichner global vereinbart und initialisiert werden:

```
CONST long     : maximale Größe der
                 Integerzahlen (max. 36)
TYPE int_long = integer[long];
VAR bell,bs    : char;
                 {Piepston und Backspace}
    nachkomma  : integer; {Anzahl der
                 Nachkommastellen}
```

3. Die Unterprogramme

Die Unit besteht im wesentlichen aus 3 Unterprogrammen:
– *Procedure set_nachkomma:* legt die Anzahl der Nachkommastellen fest.
– *Procedure write_long:* gibt eine Zahl auf der Konsole aus. Die Zahl wird mit der Anzahl der angegebenen Nachkommastellen rechtsbündig in einem Ausgabefeld mit „feld" Stellen ausgegeben. Wird „feld" zu klein angegeben, werden statt dessen Sternchen ausgegeben.
– *Procedure read_long:* Dies ist die umfangreichste Prozedur, weil sie eine ganze Reihe von Plausibilitäts- und Fehlerkontrollen enthält.

Der Initialisierungsteil umfaßt lediglich die Initialisierung der Variablen bs und bell (chr(8) und chr(7)) und den Defaultwert für die Nachkommastellen (2). Als Trennzeichen für die Nachkommastellen kann ein Punkt oder ein Komma eingegeben werden. Die Ausgabe erfolgt jedoch immer mit einem Punkt.
Je nach Anwendungsfall können auch verschiedene Änderungen eingebaut werden:
Wird die Ausgabe auf ein anderes Gerät als die Konsole gewünscht (z.B. auf Diskette), kann die Parameterliste der Prozedur write_long um einen Parameter „out" vom Typ „text" erweitert und die Write-Anweisung geändert werden.
Die Überprüfung der Nachkommastellen kann dahingehend abgeändert werden, daß überzählige Nachkommastellen einfach abgeschnitten werden oder daß gerundet wird.
Bei Bedarf können Routinen für Multiplikation und Division eingefügt werden.
Auf eine Kleinigkeit möchte ich noch besonders hinweisen: In der Prozedur read_string muß die Range-Check-Option des Compilers ausgeschaltet werden, da auf die einzelnen Elemente des Strings indiziert zugegriffen wird. Durch das Abschalten der Option wird eine Überprüfung der Indizes auf ihre Gültigkeit verhindert. Dies wird durch die Routine selbst erledigt.

Kurzhinweise

1. Zweck: Ein- und Ausgabe von Long-Integer-Zahlen
2. Konfiguration:
II+/e/c, UCSD-Pascal 1.1
3. Test:
E(xecute Demo
4. Sammeldisk:
DEMO.TEXT (Testprogramm)
LONGINT.TEXT (Unit-Quelltext)
5. Wie man Units erstellt, wird im Pascal-Kompaktkurs, Teil 2 (Heft 12/86) beschrieben.

DEMO.TEXT

```
program int;
uses long_integer;
var long_i: integer[long];
    n     : integer;
    ch    : char;
begin
repeat
   write ('Anzahl der Nachkommastellen: ');
   readln (n);
   set_nachkomma (n);
   write ('Bitte eine Zahl eingeben: ');
   read_long (long_i);
   writeln;
   write ('Die Zahl lautet:       ');
   write_long (long_i,15);
   writeln;
   writeln;
   writeln;
   write ('Noch einmal? (j/n)');read (ch);
   writeln;
   until ch in ['N','n'];
end.
```

LONGINT.TEXT

```
{$S+}
unit longinteger; intrinsic code 23 data 24;
interface
const long = 10;
type int_long = integer[long];

procedure read_long (var int: int_long);
procedure write_long (int: int_long; feld: integer);
procedure set_nachkomma (i: integer);


implementation
var bell,bs : char;
    nachkomma : integer;

procedure read_long;
const longl = 36; {maximale Zahlenlaenge = 36 digits}
type int_string = string[longl];
var ok,neg: boolean;
    is : int_string;
    fak,z,laenge_is,i,point_pos : integer;

procedure read_string (var is : int_string);
var ch: char;
    ziffer_anzahl,i : integer;
    legal : set of char;
    ziffern : set of '0'..'9';
    dez_zeichen : set of ',','.';
    point : boolean;
begin
if nachkomma = 0
   then dez_zeichen := [] {Keine Nachkommastellen}
   else dez_zeichen := [',','.'];
ziffern := ['0'..'9'];
legal := ziffern + ['-','+'] + dez_zeichen;{erlaubte Zeichen}
i := 0;             {Index fuer String}
ziffer_anzahl := 0; {zaehlt die eingegebenen Ziffern}
point := false;     {Merker fuer Dez.-Punkt/Komma}

{$R-}
{Range-Check abschalten}
{notwendig bei der Indizierung des Strings}
repeat
  read (keyboard,ch); {Zeichen einlesen, aber nicht ausgeben}
  if ch in legal
     then begin
        write (ch);
        if ch = bs then begin
              write (' ',bs);
              if is[i] = '.' then point := false;
              if is[i] in ziffern then
                ziffer_anzahl := ziffer_anzahl-1;
              i := i-1;
              end
        else begin
              i := i+1;
              {evtl. Dez.-Komma in Dez.-Punkt umwandeln:}
              if ch = ',' then ch := '.';
              if ch = '.' then point := true;
              if ch in ziffern then
                ziffer_anzahl := ziffer_anzahl+1;
              is [i] := ch;
              end;

{Menge der zulaessigen Zeichen fuer die}
{naechste Eingabe bestimmen:}
        if ziffer_anzahl >= longl
        then legal := [bs]
        else
          if i = 0
          then legal := ziffern + ['-','+'] + dez_zeichen
          else
            if point
            then legal := ziffern + [bs]
            else legal := ziffern + [bs] + dez_zeichen;
        end
     else {Falls illegales Zeichen gegeben wurde: Piepston}
          if not eoln (keyboard) then write (bell);
  until eoln (keyboard);

if i = 0
then is := '0'          {Es wurde nur 'RETURN' eingegeben}
else is [0] := chr (i); {Laenge eintragen}
{$R+}
end;

begin
repeat
   ok := true;
   read_string (is);
   laenge_is := length (is);

   {Position des Dezimalpunktes bestimmen und,}
   {falls vorhanden, entfernen}
   point_pos := pos('.',is);
   if point_pos <> 0 then delete (is,point_pos,1);
   if point_pos = 0 then point_pos := laenge_is;
      {Falls kein Punkt: Punkt am Ende}

   {Anzahl der anzuhaengenden Nullen bestimmen}
   fak := nachkomma - (laenge_is - point_pos);

   {Fehler, falls zuviele Nachkommastellen eingegeben wurden}
   {oder die Anzahl der Ziffern nach dem Einfuegen der}
   {Nullen zu gross wuerde.}
   if (fak < 0) or (length(is) + fak > long) then begin
      ok := false;
      write (bell);
      {Cursor zurueck auf den Anfang und Eingabe loeschen}
      for i := 1 to laenge_is do write (bs,' ',bs);
      end;
   until ok;

neg := false;
int := 0;

{erforderliche Nullen anhaengen:}
for i := 1 to fak do is := concat (is,'0');

{Vorzeichen bestimmen und ggf. entfernen:}
if is[1] in ['+','-'] then begin
   if is[1] = '-' then neg := true;
   delete (is,1,1);
   end;
{jetzt besteht der String nur noch aus Ziffern!}

{String in Integerzahl umwandeln:}
i := 0;
laenge_is := length(is);
{Laenge nach diesen Manipulationen neu bestimmen}
while i < laenge_is do begin
   i := i+1;
   z := ord(is[i]) - ord('0');
   int := int*10 + z;
   end;
if neg and (int <> 0) then int := -int;
end;

procedure write_long;
var is : string[38]; {max. 36 digits+Vorzeichen+Dezimalpunkt}
    laenge_is,i : integer;
begin
str (int,is); {Integerzahl in String umwandeln}
laenge_is := length(is);

{Falls erforderlich Nullen einfuegen}
if (int < 0) and (laenge_is-1 <= nachkomma) then
   for i := laenge_is-1 to nachkomma do insert ('0',is,2)
else if laenge_is <= nachkomma then
   for i := laenge_is to nachkomma do is := concat ('0',is);

{Falls Nachkommastellen erforderlich: Dezimalpunkt einfuegen}
if nachkomma <> 0 then insert ('.',is,length(is)-nachkomma+1);

{Ausgabe; Falls Zahl zu lang fuer das spez. Ausgabefeld:}
{Entsprechend viele Sternchen ausgeben.}
if length(is) > feld then
  is := copy('**************************************',1,feld);
write (is:feld);
end;

procedure set_nachkomma;
begin
  nachkomma := i;
end;
begin
  {Initialisierung}
  bs := chr (8);    {Backspace}
  bell := chr (7);  {Piepston}
  nachkomma := 2;   {Defaultwert}
end.
```

Readin-Prozedur für Ganzzahlen

Eine Utility für USCD-Pascal

von Norbert M. Doerner

Ausgangslage

Eine der unangenehmsten Eigenschaften des Apple-Pascal-Betriebssystems ist wohl die reichlich unfreundliche Reaktion auf einen falschen Tastendruck bei der Eingabe von Integer-Zahlen. Tippt man beispielsweise auf „A", verabschiedet sich das Programm mit „I/O-Error: bad input format", und die bereits verarbeiteten Daten sind verloren. Da mir das auch schon häufig passiert ist, machte ich mich an die Arbeit und entwickelte die hier abgedruckte Prozedur.

Prozedur Readin

Die Prozedur besteht aus drei Teilen:
1. Der momentane Wert der einzulesenden Variablen wird initialisiert und ausgegeben. Dadurch kann dieser unverändert übernommen werden, eine Möglichkeit, die die Read-Prozedur nicht bietet.
2. Nun erfolgt die eigentliche Eingabe. Zusätzlich zur normalen Korrekturmöglichkeit mit dem Linkspfeil sind zwei weitere Tasten belegt:
– Die Delete-Taste macht „reinen Tisch". Alle bisher auf dem Bildschirm ausgegebenen Zahlen werden gelöscht, bleiben aber im Puffer. Dies ersetzt das mehrmalige Drücken der Linkspfeil-Taste.
– Die Rechtspfeil-Taste bewegt den Cursor um jeweils eine Stelle nach rechts, der entsprechende Wert wird übernommen. Beendet wird die Eingabe mit dem Drükken der Leertaste oder Return. Alle Zahlen, die links vom Cursor stehen, werden übernommen, es sei denn, der Cursor befindet sich an derselben Stelle wie zu Beginn der Eingabe; in diesem Fall wird der Wert der Variablen nicht verändert.
3. Der Puffer wird in eine Zahl verwandelt und ausgegeben, denn wenn z.B. im Puffer fünfmal eine 9 steht, heißt das noch lange nicht, daß die Zahl dann 99 999 ist. Maxint setzt hier Grenzen.

Damit die Readin-Prozedur jedem Programm zur Verfügung steht, sollte sie in eine Library-Unit eingebaut werden. Dazu muß man den Testteil aus dem Listing entfernen.

Kurzhinweise

1. Zweck:
Prozedur zum komfortablen Einlesen von Integer-Werten.
2. Konfiguration:
Apple II+/e/c; Apple-Pascal 1.1 oder 1.2
3. Test:
E(xecute Readin
4. Sammeldisk:
READIN.TEXT
5. Sonstiges:
READIN.TEXT muß zunächst mit GETDOS (von Sammeldisk #15) auf Ihre Pascal-Arbeitsdiskette konvertiert und dann compiliert werden.

READIN.TEXT
(Readin-Prozedur mit Testprogramm)

```
program test;
var x:integer;
{.....................................................}
Procedure Readin (var wert : integer);
{von Norbert M. Doerner, 1985}
const       max   =   5;
       backspace  =   8;
          return  =  13;
           right  =  21;
         clearEOL =  29;
           space  =  32;
             del  = 127;
type    digit = Ø..9;
var    puffer : packed array [1..max] of digit;
       stelle : integer;
      ORDwert : integer;
            I : integer;
           I2 : integer;
      puffer2 : string [6];
      negativ : boolean;
           ch : char;
```

```
begin
  {Teil 1}
  write (chr (clearEOL), '  ', wert);
  str (wert, puffer2);
  negativ := (puffer2 [1] = '-');
  if negativ then delete (puffer2,1,1);
  stelle := length (puffer2);
  for I2 := 1 to stelle do
    puffer [stelle] := ord (puffer2 [I2]) - ord ('Ø');
  for I := stelle + 1 to max do
    if stelle < max then puffer [I] := Ø;
  wert := abs (wert);
  for I := 1 to stelle do
    write (chr (backspace));
  stelle:=1;

  {Teil 2}
  repeat
    read (keyboard, ch);
    ORDwert := ord (ch);
    if (stelle = 1) and (ORDwert = 45) then
      begin
        negativ := true;
        write (ch)
      end;
    if (ORDwert = backspace) and (stelle > 1) then
      begin
        write (ch);
```

```
        stelle := stelle - 1
      end;
    if (ORDwert = right) and (stelle <= max) then
      begin
        write (puffer [stelle]);
        stelle := stelle + 1
      end;
    if ORDwert = del then
      begin
        for I := 1 to stelle - 1 do
          write (chr (backspace));
        if negativ then write (chr (backspace));
        write (chr (clearEOL));
        negativ := false;
        stelle := 1
      end;
    if (stelle <= max) and (ORDwert-ord ('Ø') in [Ø..9]) then
      begin
        write (ch);
        puffer [stelle] := ORDwert - ord ('Ø');
        stelle := stelle + 1
      end
  until ORDwert in [space,return];

  {Teil 3}
  if stelle > 1 then
    begin
      ORDwert := 1;
      wert := Ø;
      for I:=stelle - 1 downto 1 do
        begin
          wert := wert + puffer [I] * ORDwert;
          ORDwert := ORDwert * 1Ø
        end
    end;
  if negativ then wert := - wert;
  for ORDwert := 1 to stelle - 1 do
    write (chr (backspace));
  if negativ then write (chr (backspace));
  write (chr (clearEOL), wert, ' ')
end;
{.....................................................}
begin
  writeln (chr (12), 'Readin-Testprogramm');
  writeln ('von Norbert M. Doerner 1985');
  x:=19;
  repeat
    write ('Geben Sie eine Zahl ein! (Ende mit 13)');
    readin (x);
    writeln (x)
  until x=13;
  write (chr (12))
end.
```

UCSD-Pascal-Bildschirm-Dump für Ehring-V80-Karte

von Dr. U. Hesse

Angeregt durch verschiedene SCREEN80-Programme (für Pascal auf Apple IIe/c in Peeker, 12/85, S. 48) habe ich ein entsprechendes Programm für den Apple II+ in Verbindung mit der V80-Karte der Firma Ehring entwickelt, das in UCSD-Pascal den Ausdruck des 80-Zeichenbildschirms auf dem Star-Drucker DP510 erlaubt. Das Programm dürfte jedoch auch mit jedem anderen Drucker funktionieren, da die Zeichen über den normalen WRITE-Befehl ausgegeben werden.

Das Programm befindet sich unter dem Namen **V80.TEXT** auf der Peeker-Sammeldisk und muß zunächst mit GETDOS auf Ihre Pascal-Arbeitsdiskette konvertiert und dann compiliert werden.

V80.TEXT

```
PROGRAM V8Ø;
VAR
 I,J : INTEGER;
{.....................................}
PROCEDURE HARDCOPY;
TYPE
 TRIXARRAY = PACKED ARRAY[Ø..1]
             OF Ø..255;
VAR
 INH   : ARRAY [1..2ØØØ] OF Ø..255;
 TRIX  : RECORD
           CASE BOOLEAN OF
            FALSE: (ADDRESS:INTEGER);
            TRUE : (POINTER:↑TRIXARRAY)
         END;

PROCEDURE POKE(ADDR,VALUE:INTEGER);
BEGIN
 WITH TRIX DO
 BEGIN
 ADDRESS:=ADDR;
 POINTER↑[Ø]:=VALUE
 END
END;


FUNCTION PEEK(ADDR:INTEGER):INTEGER;
BEGIN
 WITH TRIX DO
 BEGIN
   ADDRESS:=ADDR;
   PEEK:=POINTER↑[Ø]
 END
END;

PROCEDURE RUECK;
VAR
 D : TEXT;
 J : INTEGER;
BEGIN
 REWRITE (D,'PRINTER:');
 FOR J := 1 TO 2ØØØ DO
 BEGIN
   WRITE(D,CHR(INH[J]));
   IF J MOD 8Ø = Ø THEN WRITELN(D)
 END;
 CLOSE(D)
END;


PROCEDURE LIES;
VAR
 I,SEITE,ADRESSE,ADR,X,Y : INTEGER;
BEGIN
 I := 1;
 FOR Y := 1 TO 25 DO
   FOR X := Ø TO 79 DO
   BEGIN
    ADR := (X+Y*8Ø+(PEEK(1787)-15)*16)
           MOD 2Ø48;
    SEITE := ADR DIV 512;
    POKE(-15615,Ø);
    ADRESSE := (-162Ø8+SEITE*4);
    POKE(ADRESSE,Ø);
    INH[I]:= PEEK(ADR MOD 512-13312);
    I := I+1
   END;
END;

BEGIN {HARDCOPY}
 LIES;
 RUECK
END;
{.....................................}
BEGIN {TEST}
 FOR J := 1 TO 24 DO
 BEGIN
   WRITE(J:2);
   FOR I := 35 TO 112 DO WRITE(CHR(I))
 END;
 HARDCOPY
END.
```

ProDOS-RAM-Disk-Driver

für die OHO-512K-RAM-Karte

von Stefan Hußfeldt

Für die OHO-512K-RAM-Karte existieren Init- und Treiber-Programme für die Betriebssysteme DOS, UCSD Pascal und CP/M. Da ich die Karte aber auch als RAM-Disk unter ProDOS benutzen wollte, schrieb ich das Programm OHO.RAM-DISK. Es handelt sich dabei um eine Kombination aus Init- und Treiber-Programm, d.h. daß dieses Programm die RAM-Disk sowohl initialisiert als auch an ProDOS anschließt. Das Ankoppeln an ProDOS wird nur durch das Eintragen diverser Parameter in die ProDOS-Global-Page vollzogen. Damit dürfte die RAM-Disk unter allen ProDOS-Versionen lauffähig sein. Erfolgreich getestet habe ich das Programm auf einem Apple II+ mit 64K RAM unter den ProDOS Versionen 1.0.1, 1.0.2 und 1.1.1.

1. Die RAM-Karte

Die maximale Speicherkapazität der OHO-Karte beträgt 512K RAM; geliefert wird die Karte jedoch nur mit 256K. Das Aufrüsten auf die maximale Kapazität beschränkt sich jedoch auf den Kauf von acht Speicher-ICs vom Typ 41256, die dann in die freien Sockel auf der Karte gesteckt werden. Bei meiner Version der RAM-Karte muß jetzt nur noch ein Jumper umgesteckt werden, damit die vollen 512K RAM adressiert werden können.
Neben diesem eigentlichen RAM verfügt die Karte auch noch über 256 Bytes RAM im Bereich $Cs00-$CsFF (s = Slot). Dort wird normalerweise das jeweilige RAM-Disk-Treiber-Programm untergebracht.
Um nun Daten von und zur RAM-Karte zu transportieren, besitzt diese ein 19-Bit-Adreß- und ein 8-Bit-Datenregister. Die Registeradressen lauten wie folgt:
$C0x0 = Low-Byte-Adreßreg.
$C0x1 = Middle-Byte-Adreßreg.
$C0x2 = High-Byte-Adreßreg. und
$C0x3 = Datenregister,
wobei x=s+$8 (s = Slot).

Im folgenden gehen wir davon aus, daß sich die Karte in Slot 5 befindet. Es gelten dann die Adressen $C0D0-$C0D3.

Durch das 19-Bit-Adreßregister lassen sich genau 2 hoch 19 = 524288 = 512K Adressen ($000000-$07FFFF) ansprechen. Die 19 Bits sind auf zwei 8-Bit-Register (Low- und Middle-Byte) und ein 3-Bit-Register (High-Byte) verteilt.
Wollen wir nun zum Beispiel in die Adresse $012345 (HHMMLL) das Daten-Byte $FF poken, müssen wir erst einmal das High-Byte der Adresse ($01) nach $C0D2, das Middle-Byte ($23) nach $C0D1 und das Low-Byte ($45) nach $C0D0 poken. Jetzt muß nur noch das Daten-Byte in Adresse $C0D3 (Datenregister) geschrieben werden. Somit steht in Adresse $012345 der RAM-Karte das Byte $FF.
Wenn wir im umgekehrten Fall ein Daten-Byte von der Karte lesen wollen, müssen wir zuerst wieder die Adresse in das 19-Bit-Adreßregister poken und können dann den Wert aus dem Datenregister ($C0D3) lesen. Man vergleiche hierzu das analoge Verfahren bei der 1-Megabyte-AP33-RAM-Karte, die ein 20-Bit-Adreßregister verwendet (s. Schaubild in Peeker, Heft 7/85, S. 10).

2. Das Programm

2.1. RAM-Disk-Init

Gestartet wird das Programm mit BRUN OHO.RAMDISK. Zuerst wird festgestellt, ob die RAM-Karte schon initialisiert ist. Dies geschieht, indem die ersten drei Bytes des Treiber-Programmes, welches ab $C500 im Speicher liegt (RAM-Karte in Slot 5), auf die Bytefolge „4C72C5" hin überprüft werden. Ist das Ergebnis positiv (= RAM-Karte initialisiert), überzeugt sich das Programm, ob ProDOS von der Existenz der RAM-Disk „weiß", d.h. ob die RAM-Disk in der ProDOS-Global-Page eingetragen ist. Ist dies nicht der Fall, wird die RAM-Karte als RAM-Disk in Slot 5, Drive 1, Volume: /RAM/ (1024 Blöcke) angeschlossen und das Programm verlassen.
Ist das Ergebnis der Überprüfung negativ, übernimmt das Programm das Initialisieren der Karte. Dazu wird zuerst einmal der RAM-Bereich der Karte für die Blöcke 0-6 der RAM-Disk gelöscht (Adreßbereich: $00000-$00DFF). Von der Maßnahme, die ganze Karte zu löschen, habe ich abgesehen, da dies nicht notwendig ist. (Sie haben es sicher gemerkt: Auf der Karte sind auch die Blöcke 0 und 1 existent. Sie enthalten bei physischen Disketten den Urlader. Auf der RAM-Disk stehen sie zur freien Verfügung und können über das MLI bearbeitet werden.)

Als nächstes wird der Volume-Directory-Kopf angelegt (Block 2), dann die Blöcke 3-5 als Non-Key-Blocks des Volume-Directory initialisiert, die Volume-Bit-Map (Block 6) erstellt und zum Schluß der RAM-Disk-Driver nach Adresse $C500 kopiert und die RAM-Disk an ProDOS angeschlossen. Dazu müssen folgende Parameter in die ProDOS-Global-Page eingetragen werden: Disk-Driver-Vektor (= Beginn des Treiberprogrammes), Device-Anzahl (Anzahl der „Geräte" um eins erhöhen) und Device-Bit-Muster (Geräte-Erkennungs-Bits). Wer näheres darüber und über das „Aussehen" der Blöcke 2-6 wissen möchte, der lese dies in „Apple ProDOS für Aufsteiger, Band 1" nach.
Nach einem Neubooten kann die RAM-Disk am einfachsten mit BRUN OHO.RAMDISK wieder an ProDOS angeschlossen werden. Die Daten auf der RAM-Disk bleiben dabei natürlich erhalten. Zwei weitere Möglichkeiten, die Karte wieder anzuschließen, sind:
1. Nach CALL -151 im Monitor $C500G eingeben,
2. Unter dem BASIC.SYSTEM CALL 49152+256*5 eintippen.

2.2. RAM-Disk-Treiber

Die eigentliche Aufgabe eines RAM-Disk-Treibers ist es, Daten von und zur RAM-Karte zu transferieren. Damit das Programm die richtigen Daten überträgt, sind ihm einige Werte zu übergeben. Vor einem Sprung zum RAM-Disk-Treiber hat das MLI (Machine Language Interface) in der Zeropage folgende Werte abgelegt:
$0042: Befehl (0=Seek, 1=Read, 2=Write, 3=Format)
$0043: Unit-Nummer ($50 für Slot 5, Drive 1)
$0044: Low- und High-Byte des 512-Byte-I/O-Puffer
$0046: Low- und High-Byte der Blocknummer.

Als Befehle sind bei einer RAM-Disk nur Read und Write zulässig. Bei Aufruf von Seek wird der Treiber mit CLC und Accu = $00 (kein Fehler) und bei Format mit SEC und Accu = $27 (I/O-Fehler) verlassen.
Ein Vergleich der vom MLI übergebenen Unit-Nummer mit der eigenen Unit-Nummer ist hier nicht notwendig.
Die Adresse des I/O-Puffers wird schon vom MLI auf ihre Richtigkeit hin überprüft, kann also vom Treiber ohne Test übernommen werden.
Auch wird der Wert für die Blocknummer vor Aufruf des Treibers vom MLI berechnet und ist somit als richtig anzusehen.
Bei einem Zugriff auf die RAM-Disk mit Read oder Write wird nun als nächstes aus der Blocknummer der entsprechende RAM-Bereich auf der Karte berechnet. Dies geschieht folgendermaßen:

```
Blocknummer        Adreßbereich
$0000       ->     $000000-$0001FF
$0001       ->     $000200-$0003FF
$0002       ->     $000400-$0005FF
  :         ->          :
$03FF       ->     $07FE00-$07FFFF
```

Jetzt wird bei einem Read der entsprechende Adreßbereich der RAM-Karte in den I/O-Puffer übertragen. Bei Write kopiert der Treiber den I/O-Puffer in den berechneten Adreßbereich der RAM-Karte. In beiden Fällen wird der Treiber mit CLC und Accu = $00 (kein Fehler) verlassen.
Die Übertragungsrate beträgt 51.2K/s bei Read und 53.7K/s bei Write.

3. Hinweise und Änderungen

```
OHO.RAMDISK      -> für Slot 5; 512K
OHO.RAMDISK.ACC  -> für LC;     512K
OHO.RAMDISK.256  -> für Slot 5, 256K
OHO.RD.ACC.256   -> für LC;     256K
RENAME.VOL.RD    -> zum Umbenennen
```

Das vorliegende Programm OHO.RAMDISK geht davon aus, daß sich die OHO-RAM-Karte in Slot 5 befindet. Soll die Karte in einem anderen Slot arbeiten, müssen folgende Zeilen im Quellcode T.OHO.RAMDISK geändert werden: 16 SLOT, 17 SLOTCs, 28 SLDR (s. Zeile 19-26), 266 Device-Bit-Muster (s. Zeile 257-264). Für eine RAM-Disk in Slot 4, Drive 2 lauten die Zeilen folgendermaßen: 16 SLOT EQU $40, 17 SLOTCs EQU $C400, 28 SLDR EQU $BF28, 266 LDA #%11001111.

Wer der RAM-Disk einen anderen Volume-Namen geben möchte, kann dies mit Hilfe des BASIC-Programms RENAME.VOL.RD tun.

Ist die RAM-Karte nur mit 256K RAM bestückt, müssen im Quelltext zwei Zeilen geändert werden: Zeile 67 in HEX 0D000006000002 und Zeile 131 in LDY #$3F. (Hex-Dump: $203A:02 und 2096:3F) Auf der RAM-Disk stehen jetzt 505 Blöcke zur freien Verfügung. Die 256K-Varianten befinden sich auf der Sammeldisk als Objektcodes (OHO.RAMDISK.256 und OHO.RD.ACC.256).

Das Programm gibt es in zwei Versionen:
Die Version OHO.RAMDISK läuft sowohl auf dem Apple II+ (64K) als auch auf dem Apple IIe mit und ohne 64K-RAM-Karte. Für den Betrieb auf dem Apple IIe mit 128K RAM muß lediglich der Volume-Name der OHO-RAM-Disk geändert werden, da die 64K-RAM-Karte, falls diese als RAM-Disk arbeitet, schon den Namen „/RAM“ hat (s.o.).
Die Version OHO.RAMDISK.ACC legt das Treiber-Programm nicht bei $C503, sondern bei Adresse $FF00 ab. Diese Version läuft ebenfalls auf beiden Rechnertypen, jedoch ist folgendes zu beachten: Beim Apple IIe mit 128K RAM liegt ab $FF00 der RAM-Disk-Treiber für die 64K-Karte. Wer aber auf die Benutzung der 64K-Karte als RAM-Disk verzichten kann, sollte diese Version benutzen. Der Vorteil: Besitzer einer Accelerator-Karte werden eine Geschwindigkeitssteigerung bei der Datenübertragung von und zur OHO.RAM-Disk feststellen. Dies liegt daran, daß jetzt nur noch wenige Adressen im Bereich von $C000-$CFFF liegen; denn wird eine dieser Adressen angesprochen, taktet die Accelerator-Karte von 3,5 auf 1 MHz herunter.

Noch ein Tip für diejenigen, welche die RAM-Karte unter KYAN PASCAL benutzen möchten. Damit die OHO-RAM-Disk (/RAM) nach dem Start des KIX.SYSTEMs automatisch mit den Files des Subdirectories BIN der KYAN-Arbeitsdiskette geladen wird, muß Bit 4 der Adresse MACHID ($BF98) in der ProDOS-Global-Page auf 1 gesetzt werden.

Kurzhinweise

1. Zweck:
ProDOS-RAM-Disk-Treiber für OHO-RAM-Karten
2. Konfiguration:
Apple II+/e mit OHO-Karte (512K oder 256K) in Slot 5; ProDOS
3. Test:
BRUN OHO.RAMDISK
4. Sammeldisk:
T.OHO.RAMDISK (Quelltext)
OHO.RAMDISK (512K, Slot 5)
T.OHO.RAMDISK.ACC (Quelltext)
OHO.RAMDISK.ACC (512K, Slot 5, LC-Version)
OHO.RAMDISK.256 (256K, Slot 5)
OHO.RD.ACC.256 (256K, Slot 5, LC-Version)
RENAME.VOL.RD
5. Sonstiges:
Die Dateien müssen zunächst mit GET-DOS oder CONVERT von der Sammeldisk auf Ihre ProDOS-Arbeitsdiskette konvertiert werden.

RENAME.VOL.RD

```
10 D$ = CHR$ (4)
13 PRINT D$;"BLOAD OHO.RAMDISK"
16 FOR I = 1 TO 15:VOLL$ =
   VOLL$ + CHR$ (00): NEXT
19 FOR I = 1 TO 15:NAME$ =
   NAME$ + CHR$ ( PEEK (I + 8212) + 128):
   NEXT
22 HOME
25 PRINT CHR$ (12);
28 PRINT "JETZIGER VOLUME-NAME: ";NAME$
31 INPUT "NEUER    VOLUME-NAME: ";NEU$:
   LANG = LEN (NEU$)
34 IF LEN (NEU$) > 15 THEN PRINT :
   PRINT "NAME ZU LANG !";: GOSUB 79:
   GOTO 25
37 NEU$ = NEU$ + MID$ (VOLL$, LEN (NEU$)
   + 1,15 - LEN (NEU$))
40 FOR I = 1 TO 15
43 TEST$ = MID$ (NEU$,I,1)
46 IF (TEST$ > = "1" AND
   TEST$ < = "9") THEN PRINT : PRINT
   "NAME MUSS MIT EINEM VERSAL BEGINNEN!";:
   GOSUB 79: GOTO 25
49 IF (TEST$ < "." AND TEST$ <> CHR$ (0))
   OR TEST$ = "/" OR TEST$ > "Z" OR
   (TEST$ > "9" AND TEST$ < "A") THEN
   PRINT: PRINT "DAS ZEICHEN '";TEST$;
   "' IST NICHT ERLAUBT !";: GOSUB 79:
   GOTO 25
52 NEXT
55 FOR I = 1 TO 15
58 POKE I + 8212, ASC ( MID$ (NEU$,I,1))
61 NEXT
64 POKE 8212,240 + LANG
67 PRINT D$;"UNLOCK OHO.RAMDISK"
70 PRINT D$;"BSAVE OHO.RAMDISK,
   A$2000,L$14F"
73 PRINT D$;"LOCK OHO.RAMDISK"
76 END
79 PRINT
82 PRINT CHR$ (7);
   "BITTE EINE TASTE DRüCKEN -> ";:
   GET KEY$
85 PRINT : PRINT
88 RETURN
```

T.OHO.RAMDISK
(Big-Mac-Quelltext)

```
1   *
2   *         OHO.RAMDISK
3   *         ═══════════
4   *
5   * Das Programm initialisiert unter
6   * ProDOS die OHO-512K-RAM-Karte
7   * als RAM-Disk:
8   *
9   * Volume: /RAM/
10  * Slot 5, Drive 1
11  * 1017 freie Blöcke
13  *
14            ORG  $2000
15  *
16  SLOT      EQU  $50        ;$s0
17  SLOTCs    EQU  $C500      ;$Cs00, s=Slot
18  *
19  * Disk-Driver-Vektor
20  *
21  * $BF10+2*s für Drive 1
22  * $BF20+2*s für Drive 2 (s=Slot)
23  *
24  * z.B.: S4,D2: -> $BF20+2*4=$BF28
25  *
26  * SLDR    EQU  $BF28      ;S4,D2
27  *
28  SLDR      EQU  $BF1A      ;S5,D1
29  *
30  DEVNUM    EQU  $BF31
31  DEVICE    EQU  $BF32
32  *
33  * Adressen der RAM-Karte
34  *
35  LBYTE     EQU  $C080+SLOT
36  MBYTE     EQU  $C081+SLOT
37  HBYTE     EQU  $C082+SLOT
38  DBYTE     EQU  $C083+SLOT
39  *
40  *═══════════════════════════*
41  *
42  *         RAM-Disk-Init
43  *         -------------
44  *
45  * RAM-Disk schon initialisiert?
46  *
47            LDX  #02
48  LOOP      LDA  INITEND+1,X
49            CMP  DRIVER,X
50            BNE  INIT       ;Nein -> Init
51            DEX
52            BPL  LOOP
53  *
54            JMP  CONNECT?
55  *
56  * 43 Bytes Volume-Directory-Kopf
57  *
58  VDK       HEX  00000300
59  *
60            HEX  F3                    ;$Fx
61  *                                    ;x=Länge
62            HEX  52414D0000000000000 ;RAM.....
63            HEX  00000000000000000   ;.......
64  *
65            HEX  00000000000000000
66            HEX  000000000100C327
67            HEX  0D0000060000004
68  *
69  * RAM-Disk wird initialisiert
70  *
71  INIT      LDY  #00
72            TYA
73            TAX
74            STA  HBYTE
75            STA  MBYTE
76  *
77  * Blöcke 0-6 werden gelöscht
78  *
79  LOOP0     STY  LBYTE
80            STA  DBYTE
81            INY
82            BNE  LOOP0
83            INX
84            STX  MBYTE
85            CPX  #$0E       ;alle Blöcke?
86            BNE  LOOP0
87  *
88  * Volume-Directory-Kopf (Block 2)
89  * anlegen.
90  *
91            LDA  #04
92            STA  MBYTE
93            LDY  #42
94  *
95  LOOP1     STY  LBYTE
96            LDA  VDK,Y
97            STA  DBYTE
98            DEY
99            BPL  LOOP1
100 *
101 * Blöcke 3-5 (Volume-Directory
102 * Non-Key-Blocks) formattieren.
103 *
104           LDX  #03
105 LOOP2     TXA
106           TAY
107           ASL
108           STA  MBYTE
109           LDA  #00
110           STA  LBYTE
111           DEX
112           STX  DBYTE      ;Rückw.-Zeiger
113           LDA  #02
114           STA  LBYTE
115           INY
116           STY  DBYTE      ;Vorwä.-Zeiger
117           INX
118           INX
119           CPX  #06
120           BNE  LOOP2
121           LDA  #00        ;letzt. Vorwä.-
122           STA  DBYTE      ;Zeiger=00
123 *
124 * Volume-Bit-Map (Block 6) anlegen
125 * belegte Blöcke: 0-6
126 * freie   Blöcke: 7-1023
127 *
128           LDA  #$0C
129           STA  MBYTE
130           LDA  #$FF
131           LDY  #$7F
132 *
133 LOOP3     STY  LBYTE
134           STA  DBYTE
135           DEY
136           BNE  LOOP3
137           STY  LBYTE
138           INY
139           STY  DBYTE
140 *
141 * RAM-Disk-Driver wird nach Adresse
142 * $Cs00 kopiert. (s=Slot)
143 *
144           LDX  #DRIVEND-DRIVER
145 LOOP4     LDA  INITEND+1,X
146           STA  DRIVER,X
147           DEX
148           CPX  #$FF
149           BNE  LOOP4
150 *
151           JSR  CONNECT?
152 *
153 INITEND   RTS
155 *
156 *         RAM-Disk-Driver
157 *         ---------------
158 *
159           ORG  SLOTCs
160 *
161 COMMAND   EQU  $42
162 PAGES     EQU  $43        ;wird gerettet
163 PUFFERL   EQU  $44
164 PUFFERH   EQU  $45
165 BLOCKL    EQU  $46
166 BLOCKH    EQU  $47
167 *
168 *
169 DRIVER    JMP  CONNECT?
170 *
171 DRENTRY   LDA  BLOCKL
172           ASL             ;*2,BLOCKL>$7F
173           STA  MBYTE      ;-> C=1
174           TAX
175           LDA  BLOCKH
176           ROL             ;*2+C
177           STA  HBYTE
178 *
179           LDA  COMMAND
180           BEQ  NOERROR    ;00=Seek
181           CMP  #03
182           BEQ  IOERROR    ;03=Format
183           LSR             ;Read -> C=1
184           LDA  PAGES
185           PHA
186           LDA  #02
187           STA  PAGES
188           LDY  #00
189           BCS  READ1
190 *
191 * Schreibvorgang
192 *
193           LDA  PUFFERL
194           STA  WRPAGE+1
195           LDA  PUFFERH
196           STA  WRPAGE+2
197 WRITE     STY  LBYTE
198 WRPAGE    LDA  $FFFF,Y    ;wird gepokt
199           STA  DBYTE
200           INY
201           BNE  WRITE
202           INC  WRPAGE+2
203           INX
204           STX  MBYTE
205           DEC  PAGES      ;2.Page
206           BNE  WRITE
207           BEQ  RESTORE
208 *
209 * Lesevorgang
210 *
211 READ1    LDA  PUFFERL
212           STA  RDPAGE+1
213           LDA  PUFFERH
214           STA  RDPAGE+2
215 READ      STY  LBYTE
216           LDA  DBYTE
217 RDPAGE    STA  $FFFF,Y    ;wird gepokt
218           INY
219           BNE  READ
220           INC  RDPAGE+2
221           INX
222           STX  MBYTE
223           DEC  PAGES      ;2. Page
224           BNE  READ
225 *
226 RESTORE   PLA
227           STA  PAGES
228 *
229 NOERROR   TYA             ;A=Y=0=Okay
230           CLC             ;kein Fehler
231           RTS
232 *
233 IOERROR   LDA  #$27       ;I/O-Error
234           SEC             ;Fehler
235           RTS
236 *
237 * RAM-Disk angeschlossen?
238 *
239 CONNECT?  LDA  SLDR
240           CMP  #<DRENTRY
241           BNE  CONNECT
242           LDA  SLDR+1
243           CMP  #>DRENTRY
244           BNE  CONNECT
245           RTS
246 *
247 * RAM-Disk anschließen
248 *
249 CONNECT   LDA  #<DRENTRY  ;Disk-Driver-
250           STA  SLDR       ;Vektor in
251           LDA  #>DRENTRY  ;ProDOS-G-P
252           STA  SLDR+1     ;eintragen
253 *
254           INC  DEVNUM     ;um ein Drive
255           LDX  DEVNUM     ;erhöhen
256 *
257 * Device-Bit-Muster
258 *
259 *           Bit 76543210
260 *               0        Drive 1
261 *               1        Drive 2
262 *                110     Slot 6
263 *                101     Slot 5
264 *                   1111 RAM-Karte
265 *
266           LDA  #%01011111 ;S5,D1
267           STA  DEVICE,X
268 *
269 DRIVEND   RTS
```

OHO.RAMDISK.ACC
(Hex-Dump)

```
BSAVE OHO.RAMDISK.ACC,A$2000,L350

$2000: A2 04 BD 00 C5 CD 39 21
$2008: D0 31 CA 10 F5 4C 00 C5
$2010: 00 00 03 00 F3 52 41 4D
$2018: 00 00 00 00 00 00 00 00
$2020: 00 00 00 00 00 00 00 00
$2028: 00 00 00 00 00 00 00 00
$2030: 01 00 C3 27 0D 00 00 06
$2038: 00 00 04 A0 00 98 AA 8D
$2040: D2 C0 8D D1 C0 8C D0 C0
$2048: 8D D3 C0 C8 D0 F7 E8 8E
$2050: D1 C0 E0 0E D0 EF A9 04
$2058: 8D D1 C0 A0 2A 8C D0 C0
$2060: B9 10 20 8D D3 C0 88 10
$2068: F4 A2 03 8A A8 0A 8D D1
$2070: C0 A9 00 8D D0 C0 CA 8E
$2078: D3 C0 A9 02 8D D0 C0 C8
$2080: 8C D3 C0 E8 E8 E0 06 D0
$2088: E2 A9 00 8D D3 C0 A9 0C
$2090: 8D D1 C0 A9 FF A0 7F 8C
$2098: D0 C0 8D D3 C0 88 D0 F7
$20A0: 8C D0 C0 C8 8C D3 C0 A2
$20A8: 24 AD 39 21 9D 00 C5 CA
$20B0: 10 F7 2C 89 C0 2C 89 C0
$20B8: A2 6E BD CA 20 9D 00 FF
$20C0: CA 10 F7 2C 8A C0 20 0F
$20C8: C5 60 A5 46 0A 8D D1 C0
$20D0: AA A5 47 2A 8D D2 C0 A5
$20D8: 42 F0 57 C9 03 F0 56 4A
$20E0: A5 43 48 A9 02 85 43 A0
$20E8: 00 B0 23 A5 44 8D 2F FF
$20F0: A5 45 8D 30 FF 8C D0 C0
$20F8: B9 FF FF 8D D3 C0 C8 D0
$2100: F4 EE 30 FF E8 8E D1 C0
$2108: C6 43 D0 E9 F0 21 A5 44
$2110: 8D 55 FF A5 45 8D 56 FF
$2118: 8C D0 C0 AD D3 C0 99 FF
$2120: FF C8 D0 F4 EE 56 FF E8
$2128: 8E D1 C0 C6 43 D0 E9 68
$2130: 85 43 98 18 60 A9 27 38
$2138: 60 AD 1A BF C9 00 D0 08
$2140: AD 1B BF C9 FF D0 01 60
$2148: A9 00 8D 1A BF A9 FF 8D
$2150: 1B BF EE 31 BF AE 31 BF
$2158: A9 5F 9D 32 BF 60 00 00
```

OHO.RAMDISK
(Hex-Dump)

```
BSAVE OHO.RAMDISK,A$2000,L335

$2000: A2 02 BD B8 20 DD 00 C5
$2008: D0 31 CA 10 F5 4C 72 C5
$2010: 00 00 03 00 F3 52 41 4D
$2018: 00 00 00 00 00 00 00 00
$2020: 00 00 00 00 00 00 00 00
$2028: 00 00 00 00 00 00 00 00
$2030: 01 00 C3 27 0D 00 00 06
$2038: 00 00 04 A0 00 98 AA 8D
$2040: D2 C0 8D D1 C0 8C D0 C0
$2048: 8D D3 C0 C8 D0 F7 E8 8E
$2050: D1 C0 E0 0E D0 EF A9 04
$2058: 8D D1 C0 A0 2A 8C D0 C0
$2060: B9 10 20 8D D3 C0 88 10
$2068: F4 A2 03 8A A8 0A 8D D1
$2070: C0 A9 00 8D D0 C0 CA 8E
$2078: D3 C0 A9 02 8D D0 C0 C8
$2080: 8C D3 C0 E8 E8 E0 06 D0
$2088: E2 A9 00 8D D3 C0 A9 0C
$2090: 8D D1 C0 A9 FF A0 7F 8C
$2098: D0 C0 8D D3 C0 88 D0 F7
$20A0: 8C D0 C0 C8 8C D3 C0 A2
$20A8: 96 BD B8 20 9D 00 C5 CA
$20B0: E0 FF D0 F5 20 72 C5 60
$20B8: 4C 72 C5 A5 46 0A 8D D1
$20C0: C0 AA A5 47 2A 8D D2 C0
$20C8: A5 42 F0 57 C9 03 F0 56
$20D0: 4A A5 43 48 A9 02 85 43
$20D8: A0 00 B0 23 A5 44 8D 32
$20E0: C5 A5 45 8D 33 C5 8C D0
$20E8: C0 B9 FF FF 8D D3 C0 C8
$20F0: D0 F4 EE 33 C5 E8 8E D1
$20F8: C0 C6 43 D0 E9 F0 21 A5
$2100: 44 8D 58 C5 A5 45 8D 59
$2108: C5 8C D0 C0 AD D3 C0 99
$2110: FF FF C8 D0 F4 EE 59 C5
$2118: E8 8E D1 C0 C6 43 D0 E9
$2120: 68 85 43 98 18 60 A9 27
$2128: 38 60 AD 1A BF C9 03 D0
$2130: 08 AD 1B BF C9 C5 D0 01
$2138: 60 A9 03 8D 1A BF A9 C5
$2140: 8D 1B BF EE 31 BF AE 31
$2148: BF A9 5F 9D 32 BF 60 00
```

Sammeldisk #26

```
A 002 PEEKER ═══ HEFT  2/87 ═══
A 002 A-
A 003 TEXT.TEST
B 002 TEXT.LQ800
T 004 T.TEXT.LQ800
A 003 HGR1.TEST
B 002 LQ800.HGR1
T 009 T.LQ800.HGR1
A 003 HGR3.TEST
B 002 LQ800.HGR3
T 010 T.LQ800.HGR3
A 003 HGR3Q.TEST
B 002 LQ800.HGR3Q
T 010 T.LQ800.HGR3Q
A 002 B-
T 011 READIN.TEXT
A 002 C-
T 006 V80.TEXT
A 002 D-
T 003 DEMO.TEXT
T 020 LONGINT.TEXT
A 002 E-
A 035 FX80.PATCHER
T 006 T.FX80.INIT
B 002 FX80.INIT
A 002 F-
T 014 T.BRK
B 002 BRK
T 010 T.BRK.TEST
B 002 BRK.TEST
A 002 G-
A 005 STIEHL
B 030 DB-MEISTER.OBJ
B 011 DB-STARTER.OBJ
B 036 DB-PFLEGER.OBJ
B 002 DB-BILDSCHIRM
A 002 H-
B 077 ASM.SYSTEM
B 002 ASM.GP
A 002 I-
A 003 MATCH.TEST
T 033 T.MATCH
B 004 MATCH
A 002 J-
T 016 T.OHO.RAMDISK
B 003 OHO.RAMDISK
T 016 T.OHO.RAMDISK.ACC
B 003 OHO.RAMDISK.ACC
B 003 OHO.RAMDISK.256
B 003 OHO.RD.ACC.256
A 005 RENAME.VOL.RD
A 002 K-
T 072 INHALT.DISKETTEN
```

Einzelpreis DM 28,–, Fortsetzungspreis DM 20,–

MICOL-BASIC

Eine strukturierte Compiler-Sprache

getestet von Paul Batt

1. Gegenstand

Die kanadische Firma MICOL SYSTEMS hat einen Compiler für Applesoft-Basic auf den Markt gebracht, der gleichzeitig eine Weiterentwicklung von Applesoft zur strukturierten Programmiersprache ist. Er compiliert bestehende Applesoft-Programme unter ProDOS (nicht DOS 3.3); diese laufen dann im Durchschnitt 8- bis 20mal schneller als zuvor. Außerdem lassen sich mit dem MICOL-Editor strukturierte Programme schreiben, die zwar auf Applesoft aufbauen, jedoch eher an Pascal oder LOGO erinnern. MICOL-BASIC ist auf dem Apple II+, dem IIc, dem IIe und dem erweiterten IIe lauffähig und beim kanadischen Hersteller zum Einführungspreis von $49.95 (plus $10.00 Porto und Verpackung) erhältlich (jetzt $89.95).

2. Leistungsfähigkeit

MICOL-BASIC wurde hauptsächlich für Schulen entwickelt. Kinder mögen Basic aus verschiedenen Gründen ganz besonders, Pädagogen hingegen überhaupt nicht. Ihr Hauptvorwurf: Basic verführt zu einem chaotischen Programmierstil.
Der Applesoft-Compiler von MICOL bringt die beiden Standpunkte zur Deckung. Er baut auf dem auf, was Kinder mögen und in der Regel schon können, nämlich Basic, und fügt hinzu, was Pädagogen an Basic vermissen, nämlich eine Struktur. Natürlich können auch erwachsene Basic-Programmierer hiervon profitieren.
Ein einfaches Beispiel soll illustrieren, wie das MICOL-System aus Applesoft eine strukturierte Sprache macht:

Unstrukturiertes Programm in Applesoft:

```
10 LET X = 0
20 LET Y = 10
30 X = X + 1:Y = Y - 1
40 PRINT X,Y
50 IF Y > 0 THEN GOTO 30
60 STOP
```

Dasselbe Programm in strukturiertem MICOL-BASIC:

```
10 LET zähle_bis_zehn  = 0
20 LET zähle_bis_null  = 10
30   REPEAT
40     zähle_bis_zehn  = zähle_bis_zehn  + 1
50     zähle_bis_null  =  zähle_bis_null  - 1
60        PRINT zähle_bis_zehn , zähle_bis_null
70   UNTIL zähle_bis_null = 0
80 STOP
```

Das Beispiel zeigt, daß in MICOL-BASIC Variablen beliebige Längen haben und mit denselben zwei Buchstaben beginnen dürfen, daß es zusätzliche Befehle für strukturiertes Programmieren gibt (hier REPEAT.. UNTIL) und daß Programme durch Einzüge der Zeilen übersichtlicher gestaltet werden.

Außerdem fehlt in der MICOL-Version der Befehl GOTO. GOTO gilt bei den MICOL-Entwicklern als größte „Sünde" von Basic und ist daher in MICOL-BASIC zwar implementiert, es gibt aber die Compiler-Option NOGOTO: Schaltet man sie ein (z.B. als Lehrer in der Schule), dann akzeptiert der Compiler keine GOTOs.

MICOL-BASIC erlaubt modulares Programmieren wie beispielsweise Pascal oder LOGO. Zu diesem Zweck wurde Applesoft um den Befehl GOSUB ROUTINE erweitert.
Folgendes Beispiel soll dies erläutern:

```
100   GOSUB modul_eins
110   GOSUB modul_zwei
120   GOSUB modul_eins
130   END
200     ROUTINE modul_eins
210      PRINT "Wir sind im ersten Modul"
220     RETURN
230       ROUTINE modul_zwei
240        PRINT "Wir sind im zweiten Modul"
250       RETURN
```

GOSUB verlangt also nicht unbedingt eine Zeilennummer, es akzeptiert auch den benutzerdefinierten Namen einer ROUTINE, was das Programmieren besonders angenehm und übersichtlich macht.
Ein typisches MICOL-Programm sieht demnach ähnlich aus wie ein Pascal-Programm: Es besteht aus einem Hauptteil (im Beispiel Zeilen 100-130) und einer Reihe von Routinen, die aus dem Hauptprogramm aufgerufen werden. Dabei können Routinen auch aus anderen Routinen heraus aufgerufen werden.
Weitere MICOL-Befehle, die strukturiertes Programmieren erleichtern, sind die folgenden:
IF..THEN..ELSE, IF..THEN BEGIN..ELSE BEGIN..ENDIF, FOR..TO..UNTIL..NEXT (bricht eine FOR-NEXT-Schleife vorzeitig ab, wenn die Kondition nach UNTIL wahr wird, eine Annehmlichkeit, die es nicht einmal in Pascal gibt), PERFORM..UNTIL (führt eine Routine aus, bis die Kondition nach UNTIL wahr wird), REPEAT. .UNTIL und WHILE..WEND.

In MICOL-BASIC gibt es boolesche Variablen, die mit „!" enden und als TRUE oder FALSE definiert werden. Außerdem können – ähnlich wie in Pascal – Variablen am Anfang des Programms als Integer, Real, String oder Boolean definiert werden, was die Programme übersichtlicher macht.
Leider gibt es auch einen (wenn auch einzigen) schwerwiegenden Mangel: An die Einführung lokaler Variablen in Routinen wurde nicht gedacht, was in großen Programmen natürlich zu Seiteneffekten führen kann und didaktisch schlecht ist.

3. Handhabung

Der MICOL-Compiler ist weitgehend kompatibel mit Applesoft und ProDOS, d.h. alle bestehenden Applesoft- (nicht aber Integer-Basic-) Programme können compiliert werden.
Tut man dies erstmals, erlebt man eine Überraschung: Die Programme rasen in einer Geschwindigkeit los, die man nicht für möglich gehalten hätte. MICOL-compilierte Programme laufen (im Gegensatz zu Applesoft) umso schneller, je weniger Real-Variablen sie benutzen. Schreibt man bestehende Applesoft-Programme entsprechend um, wird die Geschwindigkeit so hoch, daß man sie gelegentlich – z.B. in Spielen – mit „SPEED=" wieder „abbremsen" muß.
Applesoft muß zum Compilieren zuerst in einen ASCII-Text umgewandelt (ein entsprechendes Umwandlungsprogramm wird mitgeliefert) und teilweise leicht abgeändert werden: Beispielsweise muß das Argument von HTAB und VTAB in Klammern gesetzt werden, und der Ausdruck „HPLOT x,y TO a,b" wird nicht akzeptiert, er muß durch „HPLOT x,y:HPLOT TO a,b" ersetzt werden. Außerdem führen Ausdrücke wie „IF A THEN" zu einer Fehlermeldung des Compilers und müssen in „IF A > 0 THEN" umgeschrieben werden.

Umgeschrieben werden müssen auch alle ProDOS-Kommandos wie OPEN, CLOSE, CHAIN usw. Gewissermaßen als Ausgleich für diese Unbequemlichkeit wurden die ProDOS-Kommandos in MICOL-BASIC stark vereinfacht, indem das leidige CONTROL-D-Verfahren wegfällt. Übrigens wurde auch daran gedacht, INPUT STRING$ so zu ändern, daß alle Zeichen erlaubt sind.
Das zu compilierende Programm muß mit einem Kopfteil versehen werden, der mit dem Wort PROGRAM und einem Namen beginnt und von allen Deklarationen, insbesondere DATA- und DIM-Zeilen, gefolgt sein muß.

All diese Änderungen sind zwar einfach, können in einigen Fällen, z.B. wenn viele HPLOT-Befehle umgeschrieben werden müssen, jedoch recht zeitaufwendig sein. Äußerst unglücklich ist auch, daß die Adressen $300 bis $330, wo in Applesoft häufig kleine Maschinenprogramme residieren, nicht mehr benutzbar sind. Sehr umständliches Umschreiben von Programmen ist die Folge. Immerhin ist der Compiler so „freundlich", dem Benutzer recht ausführlich mitzuteilen, was ihm während des Compilierens nicht paßt. Ich pflege Applesoft-Programme zunächst ohne Änderung dem Compiler zu übergeben, der bei einem Fehler stoppt, eine Meldung, wie z.B „- '(' EXPECTED IN LINE NUMBER 10" ausgibt, die fehlerhafte Zeile auf den Schirm holt und den Cursor gleich an die Stelle rückt, die zu korrigieren ist.

Leider kann der Compiler auch „genarrt" werden. Beispielsweise ergibt „PRINT LEFT$(MID$(STRING$,5,20),5)" nur „Schrott", weil LEFT$ und MID$ die gleiche temporäre Adresse benutzen. Gelegentlich gelang es mir auch, mit sehr komplizierten, verschachtelten Applesoft-Ausdrücken den Compiler völlig „aus dem Häuschen" zu bringen – zu über 99 Prozent akzeptiert er jedoch jedes Applesoft-Programm.

4. Nachteil

Programmiert wird mit einem Editor, der für mich die einzige kleinere Enttäuschung an MICOL-BASIC ist. Dies kann allerdings an der Neuheit des Produkts liegen, denn der Editor scheint einfach nicht „fertig" zu sein – zahlreiche Möglichkeiten fehlen. So gibt es zwar automatische Zeilennumerierung, aber keine Umnumerierung. Cut und Paste, Hold und Merge fehlen. Der Cursor ist nicht frei auf dem Schirm bewegbar, sondern nur innerhalb einer Basic-Zeile. Die Delete-Taste löscht von links nach rechts statt umgekehrt. Es kann nicht beliebig hinauf und hinunter gescrollt werden usw.

Wer Kyan-Pascal besitzt, kann statt des MICOL-Editors den Kyan-Editor benutzen, der (abgesehen vom Fehlen der Autonumerierung) sehr viel bequemer ist. Auch alle anderen ASCII-Text erzeugenden Textverarbeitungsprogramme dürften benutzbar sein. Nur auf eines muß geachtet werden: Der MICOL-Editor verändert Text-Files so, daß Text, der mit dem MICOL-Editor geschrieben oder geändert wurde, von anderen Textsystemen nicht mehr verarbeitet werden kann. Meine analytischen Fähigkeiten reichen leider nicht aus, um herauszufinden, woran dies liegt.

5. Fazit

Insgesamt erhält man beim Kauf von MICOL-BASIC für sein Geld ein sehr gutes Produkt. Zur Lieferung gehört ein 250 Seiten umfassendes Handbuch in bester Druckqualität und eine ungeschützte Diskette. Das Handbuch ist sehr ausführlich, es beschreibt jeden Applesoft-Befehl noch einmal im Detail und enthält eine gute Einführung ins strukturierte Programmieren. Besonders lobenswert ist die Tatsache, daß das Handbuch auch blutigste Anfänger berücksichtigt, indem es ausführlich erklärt, was Interpreter, Compiler, Assembler usw. sind.
Meines Erachtens schließt das Produkt eine Lücke zwischen dem relativ primitiven Basic und dem sehr akademischen Pascal, die für manchen Programmierer einfach zu groß ist.

Bücher

Physikalische Experimente mit dem Mikrocomputer

„On-Line"-Messungen mit dem Apple II im Apple-Pascal-System
von K.-D. Tillmann
1986, 362 S., kart., DM 68,–
Vieweg Verlag, Braunschweig
Gliederung
Leistungsfähigkeit des Systems – Die Hauptmenükarte – Schnelle analoge Messungen – Schnelle digitale Messungen – Regeln und Messen: Kennlinien elektronischer Bauteile – Auswertungsprogramme – Hardware – Software – Anhang: Literaturhinweise, Begriffserklärungen, Bezugsquellen, Programmverzeichnisse, Sachwortverzeichnis
Bemerkungen
Mit Hilfe des Computers können problemlos physikalische Versuche bearbeitet werden, die zahlreiche Messungen und umfangreiche Auswertungen erfordern. Diplomphysiker Tillmann behandelt in seinem Buch nur „On-Line"-Messungen und ihre Auswertungen. Zunächst wird die Steuerung des Programms durch Menükarten beschrieben, dann werden Versuche aus verschiedenen physikalischen Bereichen analysiert und die Grundlagen erläutert. Als mathematisches Handwerkszeug werden dabei Differentialgleichungen benutzt. Im nächsten Schritt folgen die Messungen und ihre grafische Auswertung. Das Buch eignet sich als Anleitung zur selbständigen Planung, Durchführung und Auswertung von Experimenten. Im Hardware-Kapitel werden alle getesteten Schaltungen mit Anschlußbelegungen angegeben, die benötigten Bauteile und ihre Bezugsquellen sind im Anhang genannt. Im umfangreichen Software-Teil werden die Quellprogramme der Pascal-Programme, -Units und Assembler-Prozeduren aufgelistet. Damit wird dem Leser eine Anleitung zur Verfügung gestellt, die dem Erstellen der Hard- und Software zum direkten Umsetzen in die Praxis dient. Zur Arbeit mit den Programmen ist folgende Konfiguration nötig: Apple II mit einem oder zwei Diskettenlaufwerken, die im Hardware-Kapitel genannten Interfaces, das Apple-Pascal-System, ggf. eine 80-Zeichenkarte und zur Druckausgabe einen grafikfähigen Drucker oder Plotter.
Mit den Informationen zu den Hardware-Bausteinen und dem für das Durchführen der physikalischen Experimente notwendigen Pascal-Programmpaket wird der Computer schrittweise in eine „On-Line"-Meßstation umgewandelt. Besonders interessant dürfte das Buch für Physiklehrer und Physikstudenten der Anfangssemester sein. Für den Apple II/IIe unter UCSD-Pascal sind die Programme auch auf Diskette zum Preis von DM 89,– vom Vieweg-Verlag zu beziehen.

CP/M – ein Betriebssystem stellt sich vor

von R. Weiß
1986, 178 S., kart., DM 39,–
Dr. Alfred Hüthig Verlag, Heidelberg
Gliederung
Suchhinweise für die wichtigsten Kommandos – Grundsätzliches zu CP/M und zu diesem Buch – Die residenten Kommandos – Die wichtigsten transienten Kommandos – ED, der Editor der CP/M-Familie – DDT, der Debugger der CP/M-Familie – Weitere Hilfsprogramme unter CP/M – Anhangstabellen
Bemerkungen
Die Portabilität von CP/M auf die unterschiedlichsten Rechner und die hohe Verfügbarkeit von Anwendungs-Software für dieses Betriebssystem haben zu dem großen und weltweiten Erfolg von CP/M geführt. Vor allem im Bereich der 8-Bit-Rechner hat sich CP/M durchgesetzt.
Das CP/M-Handbuch von R. Weiß stellt die wichtigsten Kommandos verschiedener Versionen der CP/M-Familie dar und erläutert sie durch Beispiele und Syntaxdiagramme. Die Arbeitsweise und die Kommandos des CP/M-Editors werden in einem eigenen Kapitel ausführlich beschrieben. Eine Übersicht über die wichtigsten Kommandos in den verschiedenen Betriebssystemversionen erleichtert dem CP/M-Anwender die Suche nach detaillierter Information. Tabellen zu den Fehlermeldungen des Editors und der Betriebssyteme CP/M-86, CP/M-80 und CP/M Plus runden das Buch ab und machen es zum geeigneten Nachschlagewerk.

Von Programmen, Computern und Datenträgern

Das solide Basiswissen für den EDV-Einsteiger
von A. Janson
1985, 254 S., 112 Abb., geb., DM 38,–
Franzis Verlag, München
Gliederung
Einführung – Programmierung mit problemorientierten Programmiersprachen – Organisationsformen von Datenverarbeitungsanlagen – Computer-Arithmetik – Computer-Logik – Technische Realisierung eines Computers – Programmierung mit einer maschinenorientierten Sprache – Datenträger und Peripheriegeräte – Datenorganisation – Beispiele für Problemlösungen – Literaturverzeichnis – Sachverzeichnis
Bemerkungen
Dieses Buch macht den Computer-Laien vom Punkt Null an auf leicht verständliche Weise mit den wichtigsten Prinzipien, Werkzeugen und Fachbegriffen der EDV vertraut. Die Computer-Schlagworte, die den EDV-Laien anfangs zu überrollen scheinen, werden klar und logisch erklärt. Dem Leser wird im Selbstudium ein solides Basiswissen vermittelt. Die Prinzipien der Programmierung werden anhand von BASIC- und Pascal-Beispielen erläutert. Angesichts der Stofffülle wird modernen Entwicklungen im Mikrocomputerbereich Priorität eingeräumt, ohne dabei das Grundlagenwissen zu kürzen. Schon nach kurzer Zeit kann der Leser selbständig Lösungsmodelle zu verschiedenen Problemen entwickeln, Programme eingeben und austesten.

Das Modem-Buch

Schnittstellen, Software, Konzeptionen
von A. Schön (Hrsg.)
1986, 227 S., 102 Abb., geb., DM 48,–
Franzis Verlag, München
Gliederung
Grundlagen – Anwendungen – Hardware – Software (für MS-DOS-Rechner, CP/M-Rechner, Apple II, C64, AIM-65/MC-65, mc-68000-Computer) – Bezugsquellen – Literatur
Bemerkungen
Dieses Buch bietet neben allgemeinen Informationen konkrete Anwendungen für die Datenkommunikation. Die notwendigen Grundkenntnisse zur Kommunikation über das öffentliche Telefonnetz und über die erforderlichen Schnittstellen am Computer werden vermittelt. Der Aufbau und die Funktion eines Modems werden am Beispiel des Selbstbaumodems mc detailliert erläutert.
Den Schwerpunkt des Buches bildet das Kapitel über die Kommunikationssoftware für die gängigsten Computertypen. Die Software, die aus Geschwindigkeitsgründen vorwiegend in Assembler geschrieben ist, ist umfangreich dokumentiert, so daß Anpassungen an ähnliche Systeme problemlos möglich sein sollten. Daneben ist auch ein Programmbeispiel für eine Mailbox vorhanden. Den Besitzern von Apple II-, C64- und IBM-Computern werden mit der beschriebenen Hard- und Software fertige und funktionsgerechte Lösungsvorschläge vorgestellt. Nach entsprechender Einarbeitung können sie diese Lösungen eigenen Vorstellungen anpassen. Das Buch verdeutlicht, daß Datenkommunikation auch ohne hohen Aufwand an Hard- und Software zu realisieren ist.

Computergrafik mit dem Mikrocomputer

von R. Grabowski
1985, 215 S., kart., DM 24,80
Teubner Verlag, Stuttgart
Gliederung
Was ist Computergrafik? – Grundsoftware – Koordinatensysteme, Fenster, Grafikprimitive – Grafiksegmente – Randschnitt (Clipping) – Grafikeingabe – Abbildungen – Interaktive Grafikeingabe – Dreidimensionale Grafik – Abbildungen im Raum – Axonometrische Abbildungen – Text in Grafiken – Programmverzeichnis – Literaturverzeichnis
Bemerkungen
Dieses Buch versteht sich als theoretische und praktische Einführung in die Computergrafik. Es wendet sich an Leser, die bereits einige Programmiererfahrung haben. Die theoretischen Aspekte basieren auf der analytischen und darstellenden Geometrie und auf

Elementen der Graphentheorie und setzen daher einige mathematische Kenntnisse voraus (z.B. Vektor- und Matrizenrechnung, Skalarprodukt). Dem praktischen Programmieren dient eine umfangreiche BASIC-Programmsammlung, die auch auf Diskette bezogen werden kann (lauffähig auf Apple II+/e/c, Preis: DM 46,–). Die Programme sind strukturiert aufgebaut und damit gut verständlich; sie können direkt zur Problemlösung eingesetzt werden oder als Vorlage für eigene Lösungsmöglichkeiten in BASIC oder einer strukturierten Sprache wie Pascal dienen. Die erstellten Grafiken werden in den Programmen in Form temporärer, interner Dateien abgespeichert. Diese können – abhängig vom verwendeten Computertyp – dann in externe Dateien umgewandelt werden. Innerhalb der ständig expandierenden Computergrafik beschränkt sich das Buch auf die Erzeugung und Modifikation einfarbiger Grafiken, die auf dem Grundelement „Strecke" basieren. Rastergrafiken werden nicht behandelt. Vom Thema her richtet sich Grabowski vor allem an Programmierer, die mehr Interesse an technischen als an künstlerischen Grafiken haben. Leider läßt die Druckqualität des Buches zu wünschen übrig: Der Text ist in Schreibmaschinenschrift abgedruckt, Abbildungen und mathematische Sonderzeichen sind von Hand gezeichnet, die Programmlistings werden als Ausdrucke eines sehr einfachen Matrixdruckers wiedergegeben. Da das Buch nur ein Band einer ganzen Reihe von Mikrocomputer-Büchern des Verlages ist, sollte man die Anschaffung eines gut lesbaren Druckers erwarten können.

Online-Datenbanken

Zugang zum Wissen der Welt mit Personal Computern
von S. Schubert
1986, 199 S., geb., DM 44,–
Sybex Verlag, Düsseldorf
Gliederung
Einleitung – Online-Datenbanken – Datenbankproduzenten – Datenbankanbieter – Nutzung der öffentlichen Datenübertragungsnetze – Private Übertragungsnetze – Die Recherche – Hardware – Software – Anhang: Glossar, Adressen, Gebührenangaben, Literatur, Register
Bemerkungen
Das Buch macht dem Leser, der keine technischen Vorkenntnisse mitbringen muß, das Gebiet der Online-Datenbanken zugänglich. Dieser Anwendungsbereich wird zukünftig aufgrund des wachsenden Informationsbedarfes eine beträchtliche Rolle bei der Nutzung von Personal Computern spielen. Steffen Schubert stellt dem Leser das gesamte Umfeld der Online-Systeme vor: Er schafft einen Überblick über die verschiedenen Arten von Datenbanken, über Anbieter und die notwendige Hard- und Softwareausstattung. Öffentliche und private Datenübertragungsnetze werden vorgestellt, ihr Aufbau wird erläutert. Das Kapitel Recherche erläutert die Schritte, die zur Auswahl der zutreffenden Datenbank führen. Gebührenverzeichnisse für die Datenübertragung in verschiedenen Netzen der Deutschen Bundespost runden die Informationen ab.

CAD/CAM – Entscheidungshilfen für das Management

von W. Poths und R. Löw
1985, 176 S., 110 Abb., kart., DM 42,–
Dr. Alfred Hüthig Verlag, Heidelberg
Gliederung
Betrieblicher Alltag und Allheilmittel CAX – Das Leistungszentrum Konstruktion – Das Werkzeug CAD/CAM – Die Wirtschaftlichkeit des Werkzeuges CAD/CAM – Der Entscheidungsprozeß zum CAD/CAM-Einsatz – Leitfaden zur Einführung der CA-Techniken – Literatur
Bemerkungen
Viele betriebliche Leistungsprozesse in der Fertigungsindustrie werden bereits seit über 20 Jahren von Computern überwacht und gesteuert. Die Konstruktionsabteilungen, innerbetriebliche Innovationszentren, mußten dagegen bis vor wenigen Jahren mit traditionellen Methoden arbeiten. Mit den geeigneten CAD/CAM-Systemen kann nun die Konstruktion, die im Zusammenwirken mit der Fertigungsplanung die Mehrzahl der betrieblichen Daten festlegt, alle produktspezifischen Daten mit Hilfe des Computers erarbeiten, speichern, verwalten und an die übrigen Stellen der Unternehmen weiterleiten. Neben den finanziellen Aufwendungen hat der Einsatz der CAD/CAM-Systeme weitreichende Einwirkungen auf die gesamte Unternehmensorganisation. Aus diesem Grund muß der Entscheidungsprozeß sorgfältig geplant, durchgeführt und überwacht werden. Dazu trägt dieses Buch in entscheidendem Umfang bei. Nach einer ausführlichen Analyse der Bedeutung und Aufgaben der Konstruktion im betrieblichen Leistungsprozeß erläutern die Autoren die Möglichkeiten des CAD/CAM-Einsatzes und seine vielfältigen Konsequenzen. Der Wirtschaftlichkeitsberechnung ist ein eigenes Kapitel gewidmet; individuell auszufüllende, tabellarische Erfassungsformulare (sog. Entscheidungsmatrizen) bieten konkrete Entscheidungshilfen bei der Frage nach dem Einsatz bzw. der Umstellung auf die neuen CA-Techniken. Ist die Entscheidung positiv ausgefallen, dient Kapitel 5 als Leitfaden für die Probleme, die sich bei der Realisierung ergeben können.

Sinclair QL-Begleiter

von B. Allan
1985, 133 S., kart., DM 35,–
Dr. Alfred Hüthig Verlag, Heidelberg
Gliederung
Einführung zum QL – Wir bauen BASIC-Strukturen – SuperBASIC-Grafik – Ein Turtle-Grafik-System – Überlegungen zur SuperBASIC-Syntax – Der 8049-Einchip-Computer – Der MC 68008: Ein strukturierter Chip – Die Programmierung des MC 68008 – Anhang
Bemerkungen
Der Sinclair-QL-Begleiter bietet mehr als ein Handbuch, das nur in die Arbeit mit diesem 16-Bit-Mikrocomputer einführt. Es befaßt sich schwerpunktmäßig mit der Idee, Programmierung und Anwendung von SuperBASIC und dem MC68008-Prozessor von Motorola, der im Sinclair QL verwendet wird. Anhand von Programmierbeispielen wird die Überlegenheit von SuperBASIC gegenüber anderen BASIC-Dialekten leichtverständlich erläutert. Der QL-Benutzer wird in diese BASIC-Version eingeführt, dabei wird auch die SuperBASIC-Grafik behandelt. Viele Beispiele für wichtige Programmierthemen und nützliche Anwendungen sollen dem Leser die Benutzung der Sprache nahe bringen. Die Beschreibung des INTEL-8049-Einchip-Computers, der im QL Steuerfunktionen übernimmt und besonders des MC-68008-Chips mit seinem Befehlsvorrat und den Adressierungsarten ermöglichen dem Leser, die Fähigkeiten seines Computers durch Assemblerprogrammierung vollständig auszunutzen.

MS-DOS für Insider

Wie der Spezialist mit dem Betriebssystem umgeht
von D. Smode
1987, 115 S., kart., DM 38,–
Franzis Verlag, München
Gliederung
MS-DOS, ein Überblick – Der Aufbau ablauffähiger Dateien und ihre Anbindung an MS-DOS – Die CP/M-ähnlichen Systemaufrufe – Struktur eines MS-DOS-Datenträgers, ein Exkurs – Die Xenix-kompatiblen Systemaufrufe – Funktionen zur Programmverwaltung – Funktionen zur Speicherverwaltung (memory management) – Sonstige Funktionen zur Systemverwaltung – Programmstart aus anderem Programm – Die restlichen MS-DOS-Interrupts
Bemerkungen
Das Buch basiert auf einer Artikelserie über das Betriebssystem MS-DOS, die ab Oktober 1985 in der Zeitschrift „mc" erschien. Wer hier eine Bedienungsanleitung für MS-DOS erwartet, wird auf die entsprechenden Handbücher verwiesen. Das Buch will nämlich dort anfangen, wo die Informationen des Manuals aufhören. Es beschäftigt sich mit Interna des Betriebssystems und beschreibt die Schnittstellen von MS-DOS zur Anwenderseite (system calls). Neben Funktionsbeschreibungen zur Programm-, Speicher- und Systemverwaltung von MS-DOS klärt es grundsätzliche Fragen wie den Unterschied zwischen COM- und EXE-Files, das Verschieben von Speicherblöcken und die Struktur der MS-DOS-Datenträger. Die Parallelen von MS-DOS zu CP/M und Unix/Xenix, die hierarchischen Strukturen der Dateiverzeichnisse und der Aufbau der FAT (File Allocation Table) werden recht ausführlich behandelt. Die nutzbaren MS-DOS-Interrupts werden aufgelistet, die darüber auslösbaren Funktionen besprochen. Die angegebenen Programmlistings sind in Assembler geschrieben.

Apple PASCAL für Fortgeschrittene

Band 1
von M. Juhnke und H. Redlin
1986, 216 S., kart., DM 42,–
Dr. Alfred Hüthig Verlag, Heidelberg
Gliederung
Apple-PASCAL-Grafik – Ein-/Ausgabemöglichkeiten – Systembedienung durch User-Software – Das UNIT-Konzept – Anhänge und Tabellen, Stichwortregister, Verzeichnis der Programme
Bemerkungen
Dieses Buch wendet sich an fortgeschrittene Pascal-Programmierer und erläutert die komplexen Befehle und Programmiertechniken dieser Sprache, die den vielfältigen Einsatz bei der Lösung von Anwenderproblemen ermöglichen. Apple-PASCAL ist die am weitesten verbreitete Version von UCSD-Pascal auf dem Heimcomputer-Sektor, deshalb befassen sich die Autoren schwerpunktmäßig mit Realisationen, die auf dem Apple II laufen. Einige Kapitel lassen sich jedoch auch auf andere Computer mit anderen UCSD-Implementationen anwenden. Der erste Teil des Buches beschäftigt sich mit den umfangreichen Grafikmöglichkeiten von Apple-PASCAL, die dem Anwender mit der Unit TURTLEGRAFICS zur Verfügung stehen. Zur Programmierung bewegter Grafik für Spiele oder Simulationen wird der Apple-PASCAL-Befehl DRAWBLOCK erläutert. Als Anwendungsbeispiel dient das Spiel FACEFIGHT. Hinweise für das Mischen von Text und Grafik und zur Erstellung eines eigenen Zeichensatzes ergänzen die Informationen zum Thema Grafik. Der zweite Teil des Buches befaßt sich ausführlich mit den schnellen und komfortablen Ein-/Ausgabemöglichkeiten von Apple-PASCAL. Schwerpunktmäßig behandelt wird der Aufbau und die Datenstruktur der UCSD- und DOS-Disketten. Ein Konvertierungsprogramm ermöglicht die Verarbeitung von DOS-Disketten unter Apple-PASCAL. Im dritten Abschnitt wird die automatische Bedienung des Systems über EXEC-Files und die Unit CHAINSTUFF behandelt. Das Unit-Konzept und die Modularisierung von Programmen sind Thema des vierten Kapitels, ebenso die Erstellung von INTRINSIC-Units und ihre Einbindung in die SYSTEM.LIBRARY. Zahlreiche kommentierte Beispielprogramme und zusätzliche Utilities für eigene Anwendungen und die Anhangskapitel zum Befehlsatz, ASCII-Zeichensatz, den Fehlermeldungen und Compiler-Steueranweisungen runden diesen ersten Band ab.

CAD/CAM – Anwendungen im Anlagenbau

von H. Richter
1986, 91 S., kart., DM 32,–
Dr. Alfred Hüthig Verlag, Heidelberg
Gliederung
CAD im Anlagenbau – Datenstrukturen – Organisation des CAD-Einsatzes – Die 2D-Welt des Anlagenbaus – Die 3D-Welt des Anlagenbaus – Zeichenausgabe (Plotten) – Zeichnungsarchivierung und CAD – Schlußbetrachtung
Bemerkungen
Dieses Buch spricht den technisch interessierten Leser und den erfahrenen Anlagenbauer gleichermaßen an, indem es Einblick in die Besonderheiten und Spezialitäten des CAD-Einsatzes im Anlagenbau gewährt. Dem CAD-Anfänger stellt es die Möglichkeiten dieser neuen Technik vor, dem CAD-Erfahrenen hilft es, eigene CAD-Anwendungen kritisch zu bewerten und auftretende Schwierigkeiten durch optimierte Anpassungen auszubessern. Das Buch liefert keine allgemeingültigen Lösungsvorschläge für CAD-Aufgaben, es versucht vielmehr, dem Leser anhand von ausgewählten, exemplarischen Lösungen ein Grundgerüst zu vermitteln, das nach eigenen Erfordernissen modifiziert werden kann. Die ausgewählten Beispiele vermitteln auch nur einen Eindruck von den vielfältigen Einsatzmöglichkeiten von CAD im Anlagenbau, ohne Anspruch auf Vollständigkeit zu erheben.

Softwareführer '87 für Personal-Computer

Alles über Programme, Anwendungsbereiche, Bezugsquellen
von G. Rolle (Hrsg.)
3., erw. Auflage 1986, 736 S., kart., DM 28,–
Dr. Rossipaul Verlag, München
Gliederung
Allgemeine Informationen – Branchenneutrale Programme – Branchenspezifische Programme – Technisch-wissenschaftliche Programme – Systemsoftware – Lehr- und Lernprogramme – Hobby, Sport, Spiel – Alphabetisches Anbieterverzeichnis – Alphabetisches Programmverzeichnis – Hardware-Register
Bemerkungen
Der Softwareführer enthält Beschreibungen und Kurzdaten von über 2700 Programmen für Personal Computer von ca. 500 Anbietern aus Deutschland, Österreich und der Schweiz (Stand: September/Oktober 1986). Zu jedem Programm werden die wichtigsten Daten kurz genannt: Zweck, Umfang, Programmanbieter und Bezugsquelle, erforderliche Massenspeicher, Hardware, Betriebssystem, RAM-Kapazität, Programmiersprache, Preis, Schulungen, Erstinstallationsdatum, Wartung/Pflege. Alle Informationen über die besprochenen Produkte stammen von den Soft- und Hardwareanbietern selbst, nicht von einem unabhängigen Tester. Die Softwareprodukte sind nach Funktionen (Adreßverwaltung, Textverarbeitung, Finanzbuchhaltung, Mathematik, Chemie etc.) und nach Branchen geordnet. Register erleichtern die Suche nach einem bestimmten Produkt. Das kurze Einführungskapitel bietet sehr allgemein gehaltene Hinweise zu den Themen Hard- und Softwaremarkt und zum EDV-Einsatz im Betrieb. Dieser redaktionelle Teil enthält nur recht oberflächliche Informationen und fällt zudem durch zahlreiche Rechtschreibfehler unangenehm auf. Für Entscheidungshilfen oder detaillierte Informationen muß man auf speziellere Literatur zurückgreifen. Kurz und gut: Der Softwareführer '87 ist für all diejenigen ein geeignetes Nachschlagewerk, die sich rasch einen vergleichenden Überblick über das Softwareangebot verschaffen möchten, um dann gezielte Informationen über einzelne Produkte vom jeweiligen Hersteller anzufordern.

Einführung in PASCAL und Turbo Pascal

von R. Zaks
1986, 464 S., geb., DM 48,–
Sybex Verlag, Düsseldorf
Gliederung
Grundlagen – Programmieren mit Pascal – Skalare Typen und Operationen – Ausdrücke und Anweisungen – Ein- und Ausgabe – Kontrollstrukturen – Prozeduren und Funktionen – Datentypen – Felder – Verbunde und Varianten – Dateien – Mengen – Zeiger und Listen – Programmentwicklung – Anhang, Literatur, Lösungen zu ausgewählten Übungen – Stichwortverzeichnis
Bemerkungen
Autor Rodnay Zaks gehört zu den Wegbereitern der industriellen und kommerziellen Anwendung von Mikroprozessoren und hat bereits viele Bücher über Computer geschrieben. Seiner Erfahrung verdankt der Leser ein sauber gegliedertes, leichtverständlich geschriebenes Lehrbuch für die Programmiersprachen UCSD (Standard) und Turbo Pascal. Dieses Buch ist die deutsche Ausgabe des amerikanischen Standardwerks „Introduction to PASCAL including UCSD PASCAL". Es wurde um die Besonderheiten von Turbo Pascal erweitert, einer schnellen Pascal-Version, die inzwischen auf dem Mikrocomputermarkt weit verbreitet ist. Die einzelnen Kapitel führen mit steigendem Schwierigkeitsgrad von den allgemeinen Konzepten der Sprache hin zu komplexen Datenstrukturen. In den ersten sechs Kapiteln werden grundlegende Definitionen behandelt, die zum Verständnis von Pascal notwendig sind. Darauf folgend werden besondere Techniken und Datenstrukturen zur Entwicklung komplexerer Programme erläutert. Der Vertiefung des theoretischen Wissens dienen Anwendungsaufgaben am Ende jedes Kapitels. Ein umfangreicher Anhang listet alle Symbole, reservierten Wörter und Syntaxregeln von Pascal auf. Dieser Anhang ist auch als Nachschlagewerk geeignet. Das Buch ist ein gelungenes Lehrbuch für Pascal-Neulinge und -Fortgeschrittene.

Damit sind Sie komplett!

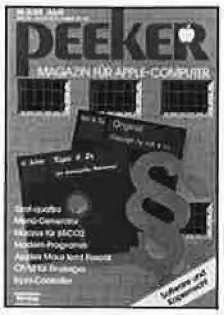
Bestellen Sie jetzt Ihre fehlenden »**peeker**-Hefte«. (Seite einfach kopieren, Heftnummer ankreuzen und Anzahl dazuschreiben)

Preis: DM 6,50 plus Versandspesen. Einfach jetzt bestellen, bevor diese Ausgaben vergriffen sind. Am besten bestellen Sie Ihre praktische »**peeker-Sammelbox**« gleich dazu (Preis DM 16,50 plus Versandspesen).

2/84 ☐

1+2/85 ☐

3/85 ☐



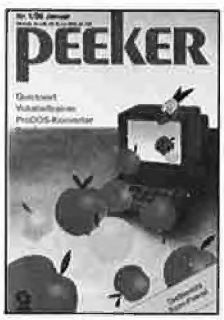
4/85 ☐

6/85 ☐

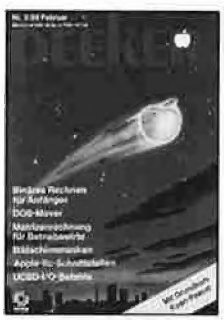
7/85 ☐

9/85 ☐

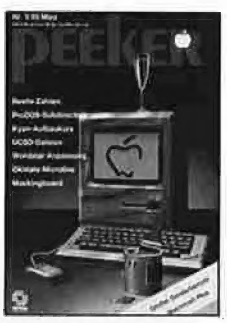
12/85 ☐



1/86 ☐

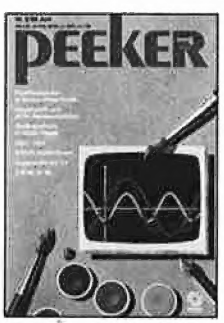


2/86 ☐



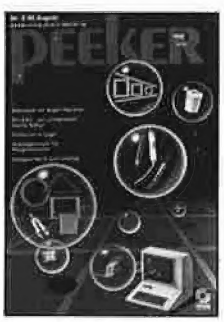
3/86 ☐

5/86 ☐

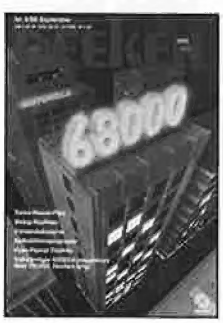


6/86 ☐

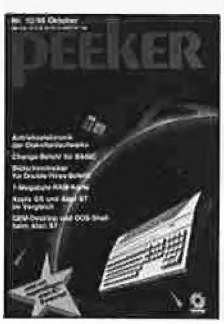
7/86 ☐



8/86 ☐

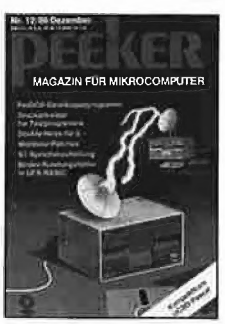


9/86 ☐



10/86 ☐

11/86 ☐



12/86 ☐

Bitte falzen

In DIN-lang Fensterumschlag einkuvertieren.
Kann als Briefdrucksache verschickt werden.

Hüthig
PUBLIKATION

Antwort

Dr. Alfred Hüthig Verlag
»peeker« Leserservice
Postfach 10 28 69

6900 Heidelberg

Name, Vorname

Straße, Postfach

PLZ, Ort

✗

Datum, Unterschrift

3 starke Bücher für Ihren

Atari ST

Dieter und Jürgen Geiß

Logo auf dem Atari ST

1986, 146 S., DM 35, – ,
ISBN 3-7785-1262-5

LOGO ist die erste Sprache auf dem Atari ST. Hier treffen sich die hervorragenden grafischen Fähigkeiten einer Programmiersprache und die überlegenen Leistungen des neuen Rechners. Das Atari-LOGO unter der Benutzeroberfläche des GEM verfügt über den zur Zeit größten LOGO-Sprachumfang und macht vollen Gebrauch von Fenstern, der Maus als Eingabegerät und den sogenannten Drop-Down-Menüs. Dieses Buch zeigt das Planen und Schreiben von faszinierenden und nützlichen Programmen. Das gesamte LOGO-System – nämlich Bedienung und Sprache – wird vorgestellt. Hier stehen die Antworten auf Fragen, die im Original-Handbuch offen geblieben sind. Der Leser lernt die gesamte LOGO-Sprache mit strukturierter Top-Down-Programmierung, Prozeduren, Rekursionen usw. Einige beispielhafte Projekte zeigen, daß LOGO weit mehr ist als eine anschauliche Lernsprache für Kinder.

Hajo Lemcke, Volker Dittmar und Michael Sommer

Programmierlexikon für Atari ST

1986, ca. 500 S., DM 48, – ,
ISBN 3-7785-1412-1

Wie jedes Lexikon ist auch dieses vollständig nach Stichworten sortiert. Im Gegensatz zu einem normalen Lexikon findet der Leser hier jedoch nicht nur eine Beschreibung, sondern gleich eine Programmieranleitung. Ebenso sind mehrere Tabellen enthalten, die das Auffinden weiterer Stichworte erleichtern und zusätzliche Informationen beinhalten. Die meisten anderen Bücher enthalten entweder eine Dokumentation über die GEM-Fähigkeiten des Rechners oder die Beschreibung der einfachen Betriebssystemroutinen. Hier jedoch findet der Leser alles. Es gibt nicht nur Hinweise zur Programmierung von Dialogboxen, Fenstern oder Kommandointerpreten, sondern es werden auch alle systeminternen Fragen beantwortet. Dies umfaßt sowohl die Programmierung der im Rechner benutzten Chips, als auch eine Beschreibung der Schnittstellen und deren Benutzung. Es wird auf alle grafischen Möglichkeiten des ST eingegangen. Gleichgültig, ob nach den deutschen oder nach den englischen Begriffen gesucht wird, es sind alle vorhanden und verweisen gegebenenfalls aufeinander.

Software-Entwicklung auf dem Atari ST

1986, 388 S., DM 54, – , ISBN 3-7785-1339-7

In diesem Buch findet der ernsthafte Programmierer alles was er braucht, um gute und professionelle Software auf dem Atari ST zu entwickeln.
Der vollständige Arbeitsablauf sowohl in einer C- als auch in einer PASCAL-Umgebung wird beschrieben, die notwendigen BATCH-Programme zum Compilieren, Assemblieren und Linken sind gelistet.
Das Kapitel 3 beschäftigt sich mit der Entwicklung von reinen TOS-Programmen. An dieser Stelle werden sowohl das Betriebssystem und der Aufruf aller GEMDOS-, BIOS- und XBIOS-Funktionen als auch die Bedeutung der System-Variablen erklärt.
Kapitel 4 ist das Herzstück des Buches: die GEM-Programmierung. Alle Funktionen der beiden großen GEM-Bibliotheken (VDI, AES) werden behandelt.
Zwei komplette Sitzungen mit dem Recourse-Construction-Set werden in Kapitel 5 dargestellt.
Im Kapitel 6 werden dem Leser an zwei vollständigen, sehr sauber programmierten und kommentierten Beispielen mit einigen hundert Zeilen fast alle Probleme vor Augen geführt und gelöst, die bei der Fensterprogrammierung auftreten. Diese Programme können als Muster für eigene Applikationen oder Desk-Accessories benutzt werden.

☐ D. u. J. Geiß, **Logo auf dem Atari ST,**
ISBN 3-7785-1262-5, DM 35, –

☐ Lemcke, Dittmar und Sommer, **Programmierlexikon für den Atari ST,**
ISBN 3-7785-1412-1, DM 48, –

☐ D. u. J. Geiß, **Software-Entwicklung auf dem Atari ST,**
ISBN 3-7785-1339-7, DM 54, –

BESTELLCOUPON

Gewünschte Bücher bitte ankreuzen und an Dr. Alfred Hüthig Verlag, Postfach 102869, 6900 Heidelberg, schicken.

Name _____

Straße _____

Ort _____

Datum _____ Unterschrift _____

Hüthig