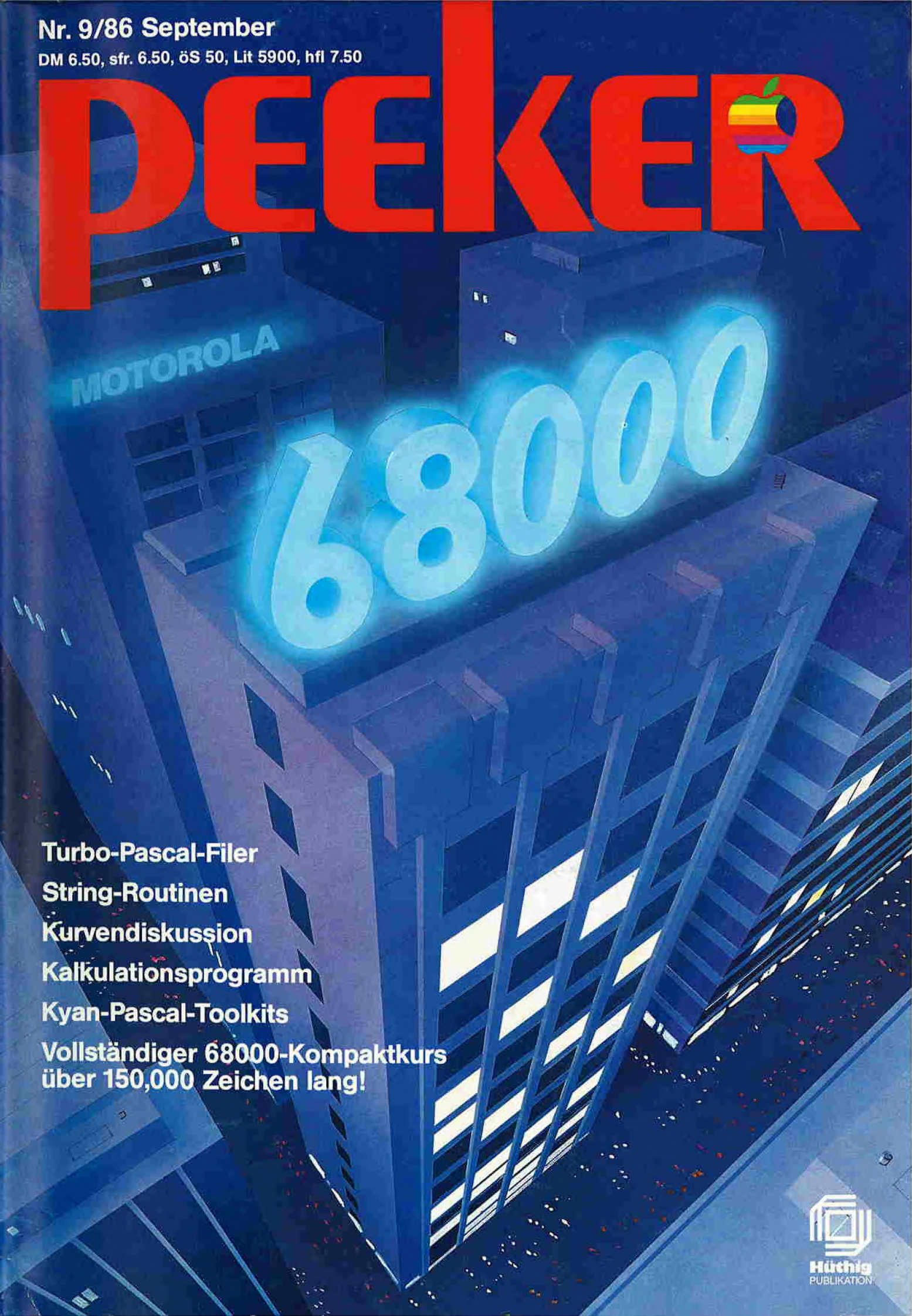
Nr. 9/86 September
DM 6.50, sfr. 6.50, öS 50, Lit 5900, hfl 7.50

PEEKER



Turbo-Pascal-Filer
String-Routinen
Kurvendiskussion
Kalkulationsprogramm
Kyan-Pascal-Toolkits
Vollständiger 68000-Kompaktkurs
über 150,000 Zeichen lang!

Hüthig
PUBLIKATION

Wichtige Information für unsere peeker-Leser

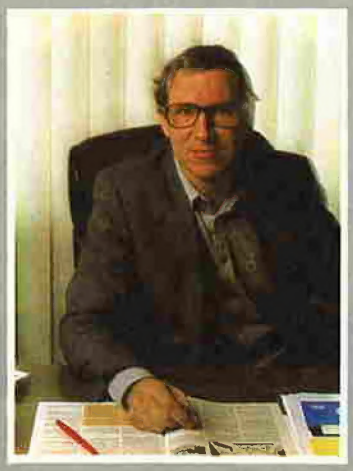
Sie erhalten Ihren »**peeker**«

- bei Ihrem **Zeitschriftenhändler am Bahnhof**
- bei allen **Apple-Händlern**
- und natürlich **beim Verlag.**

Denn der sicherste Weg, keine Ausgabe des »**peeker**« zu versäumen, ist noch immer das Abonnement zum Jahresvorzugspreis. Dabei sparen Sie bares Geld. Beachten Sie in diesem Zusammenhang unsere Anzeige auf Seite 9, oder rufen Sie uns an:
Telefon 0 62 21/489-281 (Peeker-Leserservice).

Hüthig PUBLIKATION

Dr. Alfred Hüthig Verlag, Im Weiher 10, 6900 Heidelberg
BTX ✱51851#



EDITORIAL

Leserumfrage

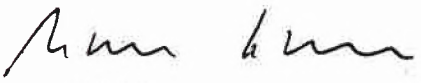
Unsere diesjährige Leserumfrage, für deren rege Beteiligung wir uns bei Ihnen bedanken, hat eine deutliche Wende in bezug auf Fremdgeräte zutage gefördert. Dies ist für uns Veranlassung genug, den Peeker behutsam gemäß Ihrem Leservotum zu erweitern. Zunächst jedoch zu den Apple-Besitzern:

Apple-Geräte im weiteren Sinne (II+/e/c, original und kompatibel) sind mit je knapp 90 % bei den Anwendern (A) und Programmierern (P) vertreten. Darunter gibt es inzwischen allein 47 % (A) bzw. 42 % (P) Original-IIe/c-Computer. Der Anteil der Original-Apple-II+-Geräte (A: 14 %, P: 15 %) und auch der II+-Kompatiblen ist damit stark zurückgegangen, während die Zahl der IIe/c-Kompatiblen insgesamt nur mäßig gewachsen ist (A: 5 %, P: 5 %). Allerdings ist die Relation der IIe/c-Kompatiblen zu den Original-IIe/c-Geräten mit 1 : 9 nicht unerheblich. Der Mac-Anteil ist mit 2 % (A) und 0,5 % (P) nach wie vor gering. Die Fremdgeräte (Atari, IBM) belaufen sich demgegenüber immerhin auf 8 %.

Wichtiger als die Altgeräte-Verteilung ist für uns Ihr Interesse an zukünftigen Apple-Computern und Fremdgeräten. Wir hatten bei der Umfrage Apple-II-Nachfolger, Mac-Nachfolger, Atari-ST und IBM-PC zur Auswahl gestellt. Wer auf der Karte nichts angekreuzt hatte (A: 7 %, P: 6 %), wurde automatisch den reinen Apple-II-Anhängern zugerechnet. Für den zukünftigen Mac haben nur 5 % (A und P) optiert. Dies war zu erwarten. Verwunderlich ist dagegen, daß sich nur 68 % (A) bzw. 64 % (P) für den Apple-II-Nachfolger interessieren, denn immerhin sind knapp 90 % der Peeker-Leser Besitzer von Apple-II-Altgeräten. Dafür liegt die Zahl der Atari/IBM-Interessierten bei sage und schreibe 25 % (A) bzw. 29 % (P), also bei über einem Viertel der Leser.

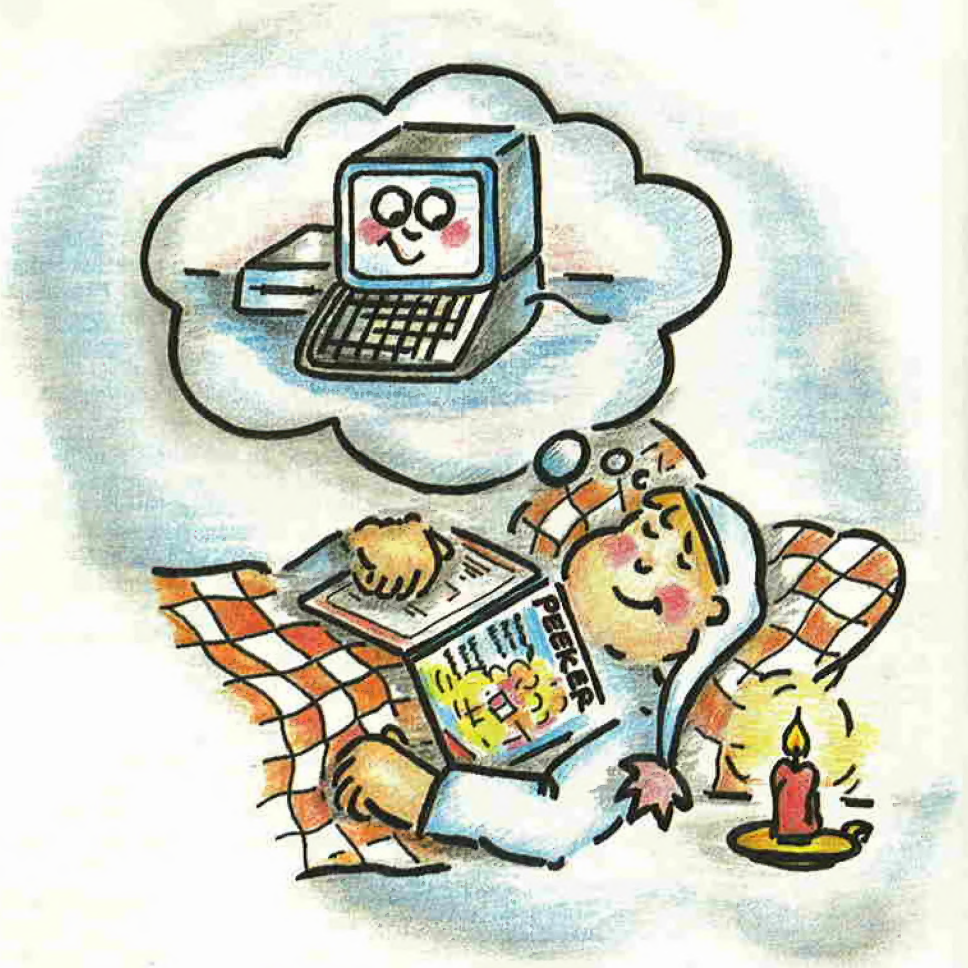
Die Zeitschrift „Computer Persönlich" (Heft 15/1986, S. 32) schätzt, daß es in der Bundesrepublik insgesamt 10000 Mac- und 65000 Atari-ST-Geräte gibt. Wir vermuten, daß von den Apple-II-Typen noch insgesamt gut 100000 Geräte benutzt werden, doch ist der Apple-II-Markt stark rückläufig. Dies ist verständlich, denn für den Apple IIe/c wurde schon seit über zwei Jahren keine Werbung mehr getrieben. Zum Apple-II-Nachfolger, den ich bereits vor einiger Zeit getestet habe, möchte ich an dieser Stelle noch kein Urteil abgeben, doch ohne massive Werbung wird dieses Gerät die verlorenen Märkte nicht mehr zurückerobern können. Für viele wird deshalb der Atari ST zum wahren Apple-II-Nachfolger werden. Dieser ganzen Entwicklung, die sich zugleich in unserem Leservotum widerspiegelt, werden wir zukünftig Rechnung tragen müssen.

Obwohl wir bereits bei unserer letztjährigen Umfrage (vgl. Heft 9/86) eine redaktionelle Öffnung des Peekers angedeutet und deshalb auch seit Anfang 1986 den Untertitel „Magazin für Apple-Computer" entfernt haben, ist der Peeker bis zum heutigen Tage eine lupenreine Apple-Zeitschrift geblieben. Angesichts der großen Zahl der Atari-Interessenten unter unseren Lesern können wir jedoch nicht mehr länger zögern. Ab dem nächsten Heft werden wir deshalb den Peeker behutsam in Richtung Atari 520/1040 ST erweitern. Dagegen scheint uns die Berichterstattung über den IBM-PC im Moment nicht opportun zu sein. Der „Peeker" war und ist eine Zeitschrift für „Peekende", und verständlicherweise findet man unter den Atari-Interessenten erheblich mehr Programmierer als Anwender.



Ulrich Stiehl

INHALT



Impressum

Peeker
3. Jahrgang 1986
ISSN 0176–9200

Verleger und Herausgeber:
Dipl.-Kfm. Holger Hüthig
Geschäftsführung Zeitschriften:
Heinz Melcher
Chefredakteur: Ulrich Stiehl (us)
Redaktion: Dagmar Berberich

Telefonnummern:
Zentrale: 06221/489-1
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 102869, 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.

Hüthig
PUBLIKATION

PEEKER

Heft 9/1986

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

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 72,–, einschl. MwSt und Versandkosten.
Jahresabonnement Ausland DM 72,– plus DM 18,– Versandkosten.
Einzelheft DM 6,50
Vertrieb Handel:
MZV – Moderner Zeitschriften Vertrieb GmbH
Breslauer Str. 5, Postfach 1123,
8057 Eching b. München,
Tel. 089 / 31 90 06 13, Telex 0 522 656
Vertriebsleitung:
Walter Menzel, Tel. (0 62 21) 48 92 80

Bankverbindungen:
Zahlungen: an den Dr. Alfred Hüthig Verlag GmbH, D-6900 Heidelberg 1: Postgirokonten: BRD: Ludwigshafen 4799-673, BLZ 545 100 67; Österreich: Wien 7555 88; Schweiz: Basel 40-24417; Niederlande: Den Haag 1 457 28; Italien: Mailand 5 968 92 08; Belgien: Brüssel 1084 1261; 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

68000-Kompaktkurs

Mit umfangreichen Befehlstabellen

von Pit Capitain

1. Einleitung

In diesem Artikel soll einer der modernen, leistungsfähigen 16-Bit-Mikroprozessoren vorgestellt werden, nämlich der MC 68000 von Motorola. Es ist aus Platzgründen leider nicht möglich, hier alle Eigenschaften dieses Prozessors zu beschreiben. Deshalb können nur einzelne Punkte herausgegriffen werden. Der Artikel wendet sich an Leser, die schon einen Mikroprozessor kennen und etwas über die Programmierung in Maschinensprache wissen, also nicht an den blutigen Laien. Um einen Vergleich mit 8-Bit-Mikroprozessoren anstellen zu können, wird in diesem Artikel stellvertretend immer wieder auf den 6502 verwiesen.
Der Mikroprozessor MC 68000 (im folgenden nur „68000" genannt) kam 1978 auf den Markt. Seitdem hat er eine recht breite Verwendung gefunden. Er wird zum Beispiel in Computern der Firmen Apple (Lisa, Macintosh), Atari (520 ST) und Commodore (Amiga) eingesetzt, um nur einige der bekannteren Mikrocomputer zu nennen.

Bei dem 6502-8-Bit-Prozessor umfaßt der Datenbus 8 Bits und der Adreßbus 16 Bits. 8-Bit-Datenbus besagt hier, daß (a) beim Input/Output und (b) bei den internen Operationen 8 Bits „auf einen Schlag" verarbeitet werden können. 16-Bit-Adreßbus besagt hier, daß $2 \uparrow 16$ (= 65536) Adressen angesprochen werden können. Analog würde bei einem 16-Bit-Prozessor der Datenbus 16 Bits und der Adreßbus 32 Bits umfassen. Dies stimmt jedoch nicht genau für den 68000. Er kann intern sogar Daten mit einer Breite von 32 Bits verarbeiten. Da sein Adreßbus jedoch normalerweise „nur" 24 Bits breit ist, lassen sich „nur" $2 \uparrow 24 =$ 16777216 Bytes (16M) direkt adressieren.
Als weitere Vorzüge gegenüber 8-Bit-Prozessoren lassen sich anführen: relativ hohe Taktfrequenz von bis zu 12,5 MHz (zumeist 8 MHz); erweiterter Befehlssatz mit zum Teil sehr leistungsfähigen Instruktionen, die zudem noch recht einfach zu erlernen sind; viele Adressierungsmöglichkeiten usw.
Der 68000 ist darüber hinaus so entworfen worden, daß höhere Programmiersprachen (z.B. Pascal) relativ leicht in Maschinensprache übersetzt werden können. Außerdem hat er zahlreiche Eigenschaften, die für einen Einsatz in größeren Rechnersystemen benötigt werden.
Alles in allem handelt es sich bei dem 68000 also um einen sehr leistungsfähigen Mikroprozessor, der nun in den nächsten Kapiteln etwas ausführlicher vorgestellt werden soll.

2. Registersatz

In diesem Abschnitt werden die Register des 68000 besprochen. Gegenüber dem 6502 mit seinen 6 Registern (A, X, Y, SP, PC, SR) gibt es beim 68000 ganze **20** Register, von denen alle bis auf eines sogar 32 Bit breit sind.
Abb. 1 zeigt diese Register. Wie man in der Abbildung erkennen kann, gibt es unterschiedliche Gruppen. Diese werden nun von oben nach unten beschrieben.

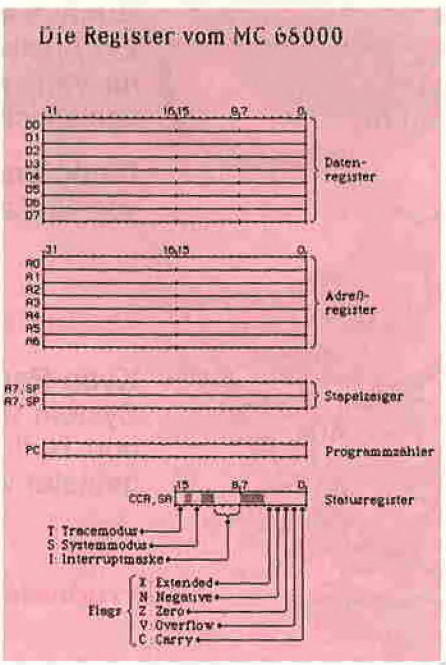


Abb. 1

Das wichtigste Register ist beim 6502 der Akkumulator (A), über den so gut wie alles abgewikkelt wird. Diese Rolle spielen beim 68000 die acht **Datenregister** (D0 bis D7). Wie der Name schon sagt, werden hier allgemeine *Daten* gespeichert und verarbeitet. Die acht Register sind untereinander völlig gleichwertig. Man kann z.B. in allen acht Datenregistern addieren, dividieren usw. Dem Programmierer stehen also praktisch acht Akkus zur Verfügung.
Wie man in der Abb. 1 sieht, sind die Datenregister 32 Bit breit. Nun muß man nicht bei jedem Befehl alle 32 Bits verarbeiten. Der Programmierer kann vielmehr wählen, ob er nur die unteren 8 Bits, die unteren 16 Bits oder alle 32 Bits verändern will.

Neben den Datenregistern gibt es 7 sog. **Adreßregister**, die ebenfalls 32 Bit breit sind. Beim 6502 gibt es kein vergleichbares Register. Es ist aber nicht schwer, die Bedeutung dieser Register zu erraten. Wie der Name schon sagt, werden hier *Adressen* gespeichert. Man kann dies vielleicht am ehesten mit der Zero-Page vom 6502 vergleichen, wo man in zwei aufeinanderfolgenden Bytes (z.B. in $00F0 und $00F1) *eine Adresse* abspeichern kann. Die Daten, die unter dieser Adresse abgespeichert sind, werden dann z.B. mit „LDA ($00F0),Y" in den Akku geladen. In einem späteren Abschnitt werden wir sehen, daß man mit den Adreßregistern dasselbe, aber auch noch viel mehr machen kann.
Der **Stapelzeiger** (Stackpointer) ist ebenfalls ein Adreßregister (A7). Er enthält nämlich die Adresse des letzten Elements, das auf dem Stack abgelegt wurde. Beim 68000 kann der Stack (fast) überall im Speicher untergebracht werden, liegt also nicht in einem festen Bereich (beim 6502: von $0100 bis $01FF).
Um nun alles noch komplizierter zu machen, gibt es beim 68000 zwei Stackpointer, die beide sogar denselben Namen haben (A7 oder SP). Dazu nur eine kurze Erklärung: Bei Computern gibt es immer zwei Programme, die praktisch „gleichzeitig" ablaufen: zum einen das Anwenderprogramm (z.B. ein BASIC-Programm) und zum anderen das Betriebssystem (z.B. DOS 3.3), das den Rechner steuert. Bei größeren Rechnern ist das Zusammenwirken dieser beiden Programme oft sehr komplex. Das Ganze wird aber dann einfacher, wenn diese beiden Programme strikt voneinander getrennt arbeiten. Der 68000 unterstützt das alles dadurch, daß er zwei sogenannte *Betriebsarten* hat. Er läuft entweder im „Benutzermodus" (Anwenderprogramm) oder im „Systemmodus" (Betriebssystem), und in jeder dieser Betriebsarten gibt es einen eigenen Stackpointer. Der Programmierer „sieht" aber nur immer *einen* von beiden.
Wer vom 6502 auf den 68000 umsteigt und seine ersten Programme entwirft, wird sich etwas verloren vorkommen unter einer solchen Masse von Registern. Aber es wird einem schnell so ergehen wie mit den Arbeitsspeichern der Computer. Waren früher 8K ein schier riesiger Bereich, kann man sich heute eigentlich gar nicht mehr vorstellen, wie man mit weniger als 64K auskommen konnte. Genauso ergeht es einem schon nach kurzer Zeit mit den vielen Registern beim 68000. Heute habe ich tatsächlich Mühe, ein Assemblerprogramm für den 6502 zu Ende zu bringen, so beschwerlich ist das Arbeiten mit den wenigen Registern.

Doch zurück zur Beschreibung der anderen Register des 68000. Wie beim 6502 gibt es einen **Programmzähler** (PC), der hier natürlich auch 32 Bit breit ist. Das einzige Register, das „nur" 16 Bits umfaßt, ist das **Statusregister** (SR). Hier wird wie beim 6502 der „Zustand" festgehalten, in dem sich der Prozessor gerade befindet.

In der unteren Hälfte des Registers befinden sich die Flags, die man auch vom 6502 her gewohnt ist. Es fehlt das D-Flag des 6502. Dafür gibt es ein neues X-Flag (extend), das ähnlich wie das C-Flag arbeitet. Diese untere Hälfte des Statusregisters kann auch nur für sich *alleine* benutzt werden. Die Bezeichnung dafür lautet dann „CCR", was die Abkürzung für „Condition-Code-Register" ist. Wer sich näher für das Statusregister interessiert, sieht in Abb. 1 die Bedeutung der einzelnen Bits.

3. Der erste Befehl
(mit 9898 Opcodes)

Nach dieser ganzen Theorie kommt nun der erste konkrete Befehl, den wir uns näher anschauen wollen. Es handelt sich um den Befehl **MOVE**. Dieser Befehl transportiert Daten von einer „Quelle" zu einem „Ziel". Für die Quelle und das Ziel gibt es sehr viele verschiedene Möglichkeiten, z.B. kann die Quelle ein Datenregister und das Ziel eine Adresse im Speicher sein.

Der Befehl „MOVE" lautet eigentlich „MOVE.s ea1,ea2". Was das ‚s' bedeutet,werden wir gleich sehen. Die Buchstaben ‚ea' sind die Abkürzung für „Effektive Adresse". Dieser Begriff ist beim 68000 sehr wichtig. Deshalb wird er hier und im nächsten Abschnitt ausführlich erklärt.

Der Befehl „MOVE.s ea1,ea2" macht folgendes: Er holt Daten von der Adresse ‚ea1' und speichert sie unter der Adresse ‚ea2'. Es muß sich hier nicht immer um tatsächliche *Adressen* im Speicher handeln. Man kann z.B. auch Daten von Register D0 ins Register D3 bewegen. Diese Register haben aber keine „Adresse" im eigentlichen Sinn. Deshalb spricht man beim 68000 von einer „Effektiven Adresse".

Fazit: Die „Effektive Adresse" (ea) gibt also an, *wo* Daten zu finden sind oder *wo* Daten abgespeichert werden können.

Dazu nun gleich ein Beispiel: Wie lautet der Befehl, der Daten aus dem Speicher mit der Adresse $07A700 in das Register D3 lädt?

Er heißt „MOVE.s $07A700,D3". Man muß also nur die Zeichen ‚ea1' und ‚ea2' durch konkrete Angaben ersetzen; dann hat man den Befehl, wie ihn der Assembler versteht. Genauso ist es mit dem Zeichen ‚s':
‚s' bedeutet „Size" (Größe). Damit wird angegeben, wie groß die Daten sind, die transportiert werden sollen. Für ‚s' kann man die folgenden Buchstaben einsetzen:

‚B' für die Größe „Byte" (8 Bits)
‚W' für die Größe „Word" (16 Bits)
‚L' für die Größe „Long" (32 Bits)

Was macht also der Befehl „MOVE.L A3,D7"? Er kopiert die 32 Bits (‚L' = Long) aus dem Adreßregister A3 in das Datenregister D7. Die Z80-Programmierer werden sich schon geärgert haben: *Vor* dem Komma steht die Quelle, *hinter* dem Komma das Ziel. Es geht also „von links nach rechts", genau umgekehrt wie beim Z80.

Wie man schon aus diesen kleinen Beispielen sehen kann, gibt es sehr viele „MOVE"-Befehle. Man kann ja für ‚s', ‚ea1' und ‚ea2' viele unterschiedliche Sachen einsetzen. Alles in allem gibt es für den „MOVE"-Befehl 9898 verschiedene Opcodes! Das zeigt, daß es beim 68000 sehr schwierig ist, aus einem „Monitor" in Hexcode einfach „drauflos" zu programmieren, müßte man sich doch allein für den „MOVE"-Befehl fast 10000 Opcodes merken. Im Assembler dagegen schreibt man nur z.B. „MOVE.L SP,A2" (Stapelzeiger in A2 kopieren) und hat mit dem einen Befehlswort „MOVE" schon alle 9898 Opcodes erledigt.

Besonders zu erwähnen ist an dieser Stelle noch, daß man z.B. mit dem Befehl „MOVE.W $07A700,$07F700" Daten (hier 16 Bits, ‚W' = Word) im Speicher verschieben kann, ohne auch nur ein einziges Register zu benutzen! Für die Profis: Beim 68000 gibt es also *Zwei-Adreß*-Befehle. Beim 6502 oder Z80 geht so etwas nur über ein Hilfsregister, dessen Inhalt vorher eventuell noch gerettet werden muß.

4. Adressierungsarten

Wie wir schon im vorigen Abschnitt gelernt haben, kann man mit der „Effektiven Adresse" (ea) angeben, *wo* Daten zu finden sind. Dafür gibt es sehr viele unterschiedliche Möglichkeiten, die man „Adressierungsarten" nennt.
Der 6502-Programmierer kennt so etwas auch schon: Man kann mit dem Befehl „LDA" ein Byte in den Akku laden. Die Adresse für dieses Byte kann auf unterschiedliche Arten bestimmt werden, z.B. „Zero Page"-Adressierung: LDA $F0 (Adresse = $00F0) oder Adressierungsart „absolut indiziert": LDA $C080,X (Adresse = $C080 + Inhalt von X).

In diesem Abschnitt werden die Adressierungsarten des 68000 vorgestellt. Dabei werden wir auch einiges über die Opcodes der Befehle erfahren. Dies alles wollen wir wieder am „MOVE"-Befehl klarmachen.

Als Beispiel nehmen wir an, wir wollten ein Word (16 Bits) in das Register D3 laden. Der Befehl dazu lautet im Assembler „MOVE.W ea,D3". Das Zeichen ‚ea' gibt die „Effektive Adresse" an, an der das Word zu finden ist. Bevor wir nun darauf eingehen, was man alles für die Zeichen ‚ea' konkret einsetzen kann, sei zunächst der Opcode für den Befehl „MOVE.W ea,D3" angegeben. Er lautet (in Binär-, Oktal- und Hex-Code = %-o-$):
%0011011000MMMRRR = o0330MR = $36XX
Die Ziffern, die mit ‚M', ‚R' und ‚X' bezeichnet sind, sind der *Code* für die Effektive Adresse der Daten. Je nachdem, welche Adressierungsart gewählt wird, müssen für ‚M', ‚R' und ‚X' andere Werte eingesetzt werden. (Der Buchstabe ‚M' steht für „Modus", das ‚R' bedeutet „Registernummer".) Wer die Opcodes wirklich verstehen möchte, der suche sich in der **Tabelle 4** im Anhang den Block, der mit „MOVE.s ea,Dm" beginnt, und dort die Zeile, bei der in der Spalte ‚s' der Buchstabe ‚W' steht. In dieser Zeile befinden sich u.a. die Opcodes für den Befehl „MOVE.W ea,Dm". Die Möglichkeiten für ‚ea' stehen am Kopf der Tabelle in den Spaltenüberschriften.

Doch nun zu den einzelnen Adressierungsarten: Nach der Bezeichnung (Überschrift) stehen in Klammern die Schreibweise für den Assembler, die Werte, die im Code für ‚M' und ‚R' einzusetzen sind, sowie ein kleines Beispiel.
Zur Verdeutlichung der Adressierungsarten sei auf die **Tabelle 1** verwiesen, in der alle Adressierungsarten und ihre Wirkungen zusammengefaßt sind.

4.1. Datenregister direkt
(Dn; M=0, R=n)
Beispiel: MOVE.W **D1**,D3

Bei dieser Adressierungsart stehen die Daten für die Operation *direkt* in einem der acht Datenregister (hier in D1). Mit dem Befehl im Beispiel wird also ein Word (16 Bits) von D1 nach D3 übertragen.

Anmerkung: Es werden immer die *unteren* 16 Bits benutzt. Die *obere* Hälfte der Register wird hier nicht verändert.

Wie lautet nun der Opcode für diesen Befehl? In der Überschrift oben steht die Angabe „M=0, R=n". Wir haben im Beispiel D1 als Quelle gewählt; deshalb gilt hier M=0 und R=1 („R" = Registernummer). Der Opcode sieht dann also wie folgt aus:
%0011011000000001 = o033001 = $3601

4.2. Adreßregister direkt
(An; M=1, R=n)
Beispiel: MOVE.W **A2**,D3

Genauso, wie man ein Word von einem *Daten*register in D3 transportieren kann, kann man das Word auch aus einem der *Adreß*register holen, wie es der Befehl im Beispiel zeigt. Die Effektive Adresse (ea) wird also mit ‚A2' angegeben. Der Opcode ist hier (M=1, R=2):
%0011011000001010 = o033012 = $360A

Spätestens hier sieht man schon, daß es leider nicht so gut ist, die Opcodes wie sonst üblich in *Hex*-Schreibweise anzugeben. Da die bitmäßige Notation zu umständlich ist, eignet sich die *oktale* Schreibweise für die meisten Befehle am besten.

Die Felder im Code der ea (‚M' und ‚R') sind jeweils 3 Bit lang. Also kann man ihre Werte direkt aus dem *oktal* codierten Opcode ablesen: Die hinteren beiden Oktalziffern sind ‚12'. Dementsprechend handelt es sich hier um die Adressierungsart „Adreßregister direkt" (M = 1), wobei das Register A2 (R = 2) angesprochen ist.

Die Opcodes im Anhang sind alle in oktaler Schreibweise angegeben.

4.3. Adreßregister indirekt

((An); M=2, R=n)
Beispiel: MOVE.W **(A6)**,D3

Normalerweise enthalten Adreßregister bei einer Operation nicht die Daten selbst, sondern nur deren *Adresse*. Der Befehl im Beispiel lädt in D3 ein Word von der Adresse, die in A6 angegeben ist. Ein kleines Beispiel soll die Wirkung verdeutlichen (vgl. auch Tab. 1):

Inhalt von D3 sei $FEDCBA98,
Inhalt von A6 sei $0007A700,
ab Adresse $07A700 sei gespeichert: $11,$22,$33, usw.
Dann lautet nach dem Befehl „MOVE.W (A6),D3" der Inhalt von D3 ‚$FEDC1122'. Alles klar?
Wie der Opcode zu dieser Adressierungsart lautet (zumindest in oktaler Schreibweise), dürfte nun nicht mehr schwer zu erraten sein. (Er lautet o033026!)
Die nächsten Adressierungsarten sind etwas komplexer:

4.4. Adreßregister indirekt mit Postinkrement

((An)+; M=3, R=n)
Beispiel: MOVE.W **(A7)+**,D3

Hier erfolgt die Adressierung der Daten wie bei der indirekten Adressierung, die gerade eben beschrieben wurde. Das heißt, die Adresse der Daten steht in dem angegebenen Adreßregister (hier in A7). Nachdem die Daten aber geholt wurden, wird das entsprechende Adreßregister *erhöht*! Man kann sich das am besten am Beispiel des Stapels (Stack) klarmachen (siehe auch **Abb. 3**):

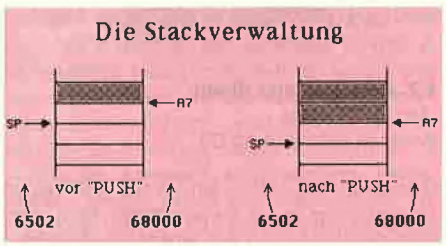


Abb. 3

Wenn beim 6502 ein Byte vom Stack in den Akku geladen wird (PLA), wird *zuerst* der Stackpointer um 1 erhöht. *Dann* wird das Byte mit der Adresse $0100 + (SP) in den Akku geholt. Der Stack beim 68000 arbeitet fast genauso, nur in einer anderen Reihenfolge: *Zuerst* werden die Daten von der Adresse geholt, auf die der Stackpointer zeigt, und *danach* erst wird der SP erhöht. Beim 6502 zeigt der SP also immer auf den nächsten freien Platz im Stack, beim 68000 auf die zuletzt in den Stack „gepushten" Daten (vgl. Abb. 3).
Der Befehl im Beispiel leistet praktisch dasselbe wie beim 6502 der Befehl „PLA". Er holt ein Word vom Stack in ein Register und korrigiert den Stapelzeiger entsprechend.

Wie man Daten aus dem Stack holt, haben wir also schon gelernt. Aber wie schreibt man Daten auf den Stack? Man muß den Vorgang einfach genau umgekehrt ablaufen lassen. Beim 68000 heißt das: Zuerst muß der Stackpointer *erniedrigt* werden, und dann können die Daten dorthin geschrieben werden, wo der Stackpointer nun hinzeigt. Genau dies leistet die nächste Adressierungsart:

4.5. Adreßregister indirekt mit Predekrement

(-(An); M=4, R=n)
Beispiel: MOVE.W D3,**-(A7)**

Dieser Befehl schreibt ein Word aus Register D3 auf den Stack. Der Stackpointer wird *vorher* entsprechend korrigiert.
An diesem Beispiel kann man sich zwei Dinge verdeutlichen: Zum einen kann man mit dem Stapelzeiger (SP oder A7) genauso arbeiten wie mit jedem anderen Adreßregister. Zum anderen kann sich der Programmierer einen *eigenen* Stack einrichten, indem er für seinen eigenen Stack ein eigenes Adreßregister als Stackpointer verwendet (z.B. A4). Dort kann er dann mit den beiden zuletzt beschriebenen Adressierungsarten Daten hineinschreiben und wieder herausholen. (Natürlich wird bei Unterprogrammaufrufen nach wie vor der „normale" Stapelzeiger A7 verwendet, um Rücksprungadressen zu speichern!)
Noch eine letzte Bemerkung zu den beiden letzten Adressierungsarten: Haben Sie sich überlegt, um *wieviel* das Adreßregister erhöht bzw. erniedrigt wird? Das hängt von der Größe der Daten ab, die bei der Operation beteiligt sind. Bei Byte-Operationen (1 Byte) ist der Wert 1, bei Word-Operationen (2 Bytes) ist er 2, und bei Long-Operationen (4 Bytes) demnach 4. In den beiden obigen Beispielen würde also Register A7 um 2 erhöht oder erniedrigt (Operationsgröße = Word = 2 Bytes).
Die nächste Adressierungsart ist in einer ähnlichen Form auch beim 6502 vorhanden:

4.6. Adreßregister indirekt mit Verschiebung

(d(An); M=5, R=n); d=displacement
Beispiel: MOVE.W **8(A2)**,D3

Hier wird die Adresse für den Operanden wie folgt bestimmt: Zu der Adresse, die im Register An (im Beispiel A2) steht, wird der Wert ‚d' (im Beispiel 8) dazuaddiert. Steht in A2 beispielsweise die Adresse $07A812, so wird durch den Befehl „MOVE.W 8(A2),D3" das Word mit der Adresse $07A81A in Register D3 geladen.

Der Wert für ‚d' ist eine vorzeichenbehaftete 16-Bit-Zahl, liegt also im Bereich von -32768 bis 32767. Dieser Wert steht im Programmspeicher nach dem Opcode für den Befehl. Der Befehl „MOVE.W 8(A2),D3" würde (oktal) wie folgt codiert werden:
o033052 (Opcode für den Befehl. M=5, R=2)
o000010 (Verschiebung ‚d' = 8 in oktaler Schreibweise)

Wie beim 6502 sind also auch beim 68000 die Befehle unterschiedlich lang, je nachdem, welche Adressierungsart(en) verwendet werden.
Wie weiter oben schon erwähnt wurde, kann man beim 6502 Adressen in 2 Bytes der Zero-Page abspeichern (z.B. in $00F0 und $00F1 die Adresse $C080 im Format Low/High). Wenn man anschließend in das Register Y den Wert für ‚d' (die Verschiebung = Offset) schreibt (z.B. 8), so kann man mit „LDA ($F0),Y" das Byte mit der Adresse $C080 + 8 = $C088 in den Akku laden. Dies funktioniert dann so, wie es der oben beschriebenen Adressierungsart entspricht.

Stimmt das? Beim 68000 ist die „Verschiebung" ein *fester* Wert, der im Programm steht. Und beim 6502? Da handelt es sich doch offensichtlich um einen *variablen* Wert, der im Register Y steht. Das wäre also ein Rückschritt, wenn es beim 68000 nicht auch so etwas gäbe.

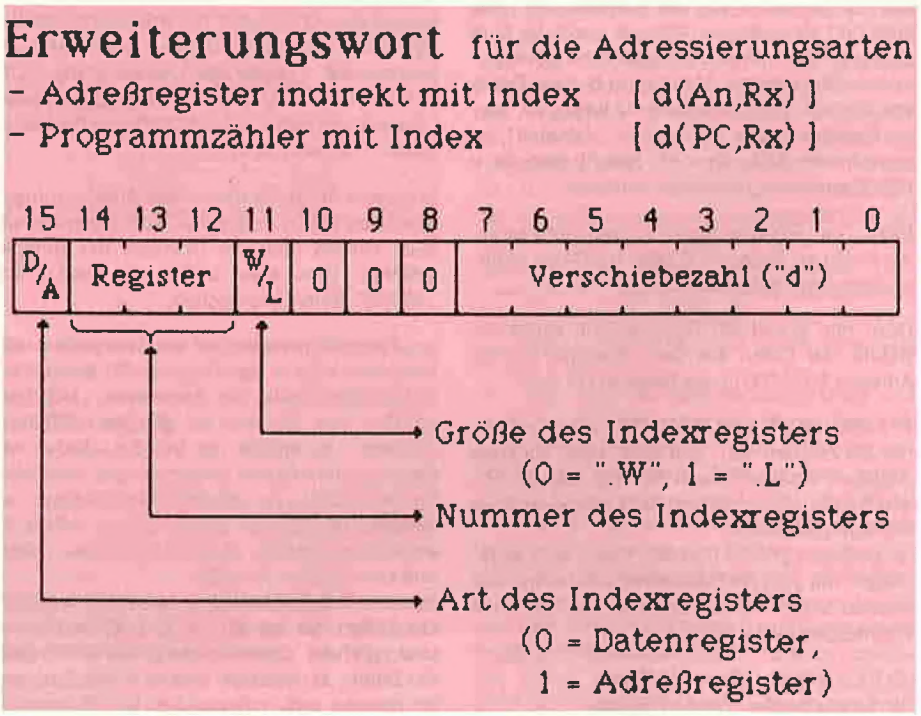


Abb. 2

»peeker« schenkt Ihnen...

. . . den praktischen Disk-Locher mit dem Sie die Speicherkapazität Ihrer Disketten im Nu verdoppeln können!

Überzeugen Sie jetzt einfach Ihre Freunde und Bekannten von den Vorzügen eines »peeker«-Abonnements«. **Sie bekommen als Dank für einen neuen Abonnenten diesen superpraktischen Disk-Locher!**

Füllen Sie den Bestellcoupon aus und ab zur Post!

Bestellcoupon

Ich habe den neuen Abonnenten geworben und erhalte kostenlos den Disk-Locher.

Name, Vorname

Straße, Postfach

PLZ, Ort

X

Datum, Unterschrift

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

Name, Vorname

Straße, Postfach

PLZ, Ort

X

Datum, Unterschrift

Gewünschte Zahlungsweise
☐ gegen Rechnung
☐ bargeldlos durch Bankeinzug

Konto-Nr. Bankleitzahl

Geldinstitut

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

X

2. Unterschrift

Peeker 9/86

Coupon ausschneiden und einsenden an:

»peeker«
Abonnementservice
Im Weiher 10
6900 Heidelberg 1

Hüthig
PUBLIKATION

4.7. Adreßregister indirekt mit Index

(d(An,Rx.W) oder d(An,Rx.L); M=6, R=n)
Beispiele: MOVE.W **-2(A0,D2.W)**,D3
oder MOVE.W **0(A0,A1.L)**,D3

Bei dieser Adressierungsart wird die Adresse wie folgt errechnet:
Die Adresse in An (hier in A0, z.B. $07A700)
+ die Verschiebezahl ‚d' (hier -2)
+ der Inhalt von Rx (hier von D2, z.B. $1000)
= Effektive Adresse (z.B. $07B6FE)
Zu der Adresse im Adreßregister An wird also eine feste Verschiebezahl ‚d' (hier nur 8 Bits) addiert, und dazu wird noch einmal der Inhalt von einem anderen Register ‚Rx' (16 oder 32 Bits) addiert. Auch hier wird nach dem Opcode für den Befehl ein Word (16 Bits) benötigt, um die notwendige Information für die Adressierung zu liefern. Das Format für dieses „Erweiterungswort" sieht man in der **Abb. 2**.

Die Befehle aus unserem Beispiel oben würden also wie folgt codiert:

```
| o033060 | o033060 | (Opcode)
| o020376 | o114000 | (Erweiterungswort)
```

Überlegen Sie einmal, welche Codierung welchem der beiden Beispiele entspricht!

Wenn Sie das alles ungefähr verstanden haben, dann ist das Schlimmste geschafft. Die nachfolgenden Adressierungsarten bringen nichts Neues mehr, sind aber trotzdem recht nützlich. Im Unterschied zu den bisherigen Adressierungsarten dient die Registernummer ‚R' nun nicht mehr dazu, die verwendeten Register zu bestimmen, sondern nur noch dazu, die Adressierungsarten weiter zu unterscheiden. Dies ist nötig, weil im „Modus"-Feld ‚M' nur noch der Wert 7 übrig ist.

4.8. Absolut kurze Adresse

(xxx; M=7, R=0)
Beispiel: MOVE.W **$0B00**,D3

Dies ist endlich einmal ein alter Bekannter, beim 6502 die Zero-Page-Adressierung. Die Adresse wird nach dem Befehls-Opcode direkt angegeben, und zwar mit *weniger* Bytes, als für eine vollständige Adresse eigentlich benötigt würden. Daher der Name „kurze" Adresse.
Der Opcode für den Beispiel-Befehl lautet (hintereinander in Oktalcode): o033070, o005400.

4.9. Absolute Doppelwort-Adresse

(xxxxxx; M=7, R=1)
Beispiel: MOVE.W **$07A700**,D3

Eigentlich muß man hierzu nichts weiter sagen, sogar die Bezeichnung der Adressierungsart stimmt (fast) mit der beim 6502 überein. Eine Bemerkung aber zur Angabe der Adresse: Beim 68000 gibt es (zum Glück) nicht mehr die Reihenfolge Low-Byte/High-Byte wie beim 6502, sondern genau umgekehrt. Der Befehl oben lautet also codiert (hier ausnahmsweise auch in Hexcode):

```
o033071, o000007, o123400
 $3639,    $0007,   $A700
```

Man kann die Adresse also direkt von links nach rechts „lesen", ohne etwas herumdrehen zu müssen.

4.10. Programmzähler mit Verschiebung

(d(PC); M=7, R=2)
Beispiel: MOVE.W **0(PC)**,D3

Diese Adressierungsart arbeitet genau wie die mit dem Namen „Adreßregister indirekt mit Verschiebung", nur wird anstelle des Adreßregisters der *Programmzähler* (PC) verwendet. Die Verschiebung ist hier ebenfalls 16 Bit lang.
Der 6502-Programmierer kennt auch diese Adressierungsart! Sie heißt dort „relative Adressierung" und kommt nur bei den bedingten Sprüngen (z.B. bei „BCS") vor. Nach dem Befehls-Opcode kommt also die Verschiebezahl, die zum Inhalt des Befehlszählers addiert wird. Beim 6502 kann dann nur zu dieser Adresse *gesprungen* werden; beim 68000 kann man von dort z.B. auch Daten laden, wie es bei dem obigen Befehl der Fall ist. Er lautet codiert o033072, o000000 und ist eine etwas umständliche Art und Weise, die unteren 16 Bits von D3 zu löschen (auf 0 zu setzen).
Das ist tatsächlich die Wirkung dieses Befehls! Denn nach der Entschlüsselung des Befehls steht der Programmzähler auf dem Erweiterungswort, zeigt also auf das Wort o000000. Da die Verschiebung hier 0 ist, ist dies gleichzeitig die Adresse für das Wort, das in D3 geladen werden soll. Deshalb ist nach diesem Befehl D3 auf $xxxx0000 (nur 16 Bits!).
Man kann hier noch einen Punkt besonders erwähnen: Wir haben gesehen, daß der 6502 ebenfalls diese Adressierungsart kennt. Er kann sie aber nur in ganz bestimmten Befehlen verwenden (bei bedingten Verzweigungen). Genauso verhält es sich beim 68000. Auch hier kann man nicht bei jedem Befehl jede Adressierungsart verwenden, sondern nur ganz bestimmte. Welche das im einzelnen sind, kann man den Befehlstabellen im Anhang entnehmen.

4.11. Programmzähler mit Index

(d(PC,Rx.W) oder
d(PC,Rx.L); M=7, R=3)
Beispiel: MOVE.W **$40(PC,D0.W)**,D3
oder MOVE.W **-$40(PC,D7.L)**,D3

Auch diese Adressierungsart ist von der Arbeitsweise her bekannt: Es handelt sich um die Adressierungsart mit dem Namen „Adreßregister indirekt mit Index". Hier wird wiederum anstelle des Adreßregisters der *Programmzähler* zur Bestimmung der Basisadresse benutzt. Das Erweiterungswort, das nach dem Befehls-Opcode folgt, hat dasselbe Format wie bei der oben beschriebenen Adressierung mit dem Adreßregister. Das Format ist in Abb. 2 gezeigt.

4.12. Direkte Daten

(#xxx; M=7, R=4)
Beispiel: MOVE.W # **$270F**,D3

Dies ist wieder eine Adressierungsart, die auch der 6502 zu bieten hat. Hier stehen die Daten *direkt* im Programm nach dem Opcode für den Befehl. Der Code für den obigen Befehl lautet (wieder mit der zusätzlichen Hex-Codierung):

```
o033074, o023417
 $363C,    $270F
```

Auch hier gilt, daß bei den Daten zuerst das *höchstwertige* Byte kommt, und am Ende steht das niedrigstwertige Byte. Das ist auch bei Daten der Fall, die 4 Byte lang sind (Long). Bei den Daten, die nur **ein** Byte lang sind, gibt es das folgende Problem:
Wie beim „MOVE"-Befehl sind auch die übrigen Opcodes der Befehle alle 16 Bit (also 2 Byte) lang. Aus diesem Grund haben die Entwickler des 68000 festgelegt, daß Opcodes von Befehlen nur bei *geraden* Adressen im Speicher beginnen dürfen. (Also z.B. bei $040030, nicht aber bei $040033).
Damit das immer gewährleistet ist, müssen auch alle Erweiterungsworte, die für die Adressierung benötigt werden, entweder 2 oder 4 Bytes lang sein. Auch wenn man bei der direkten Adressierung mit der Operationsgröße Byte nur **1** Byte für die Daten brauchte, werden im Speicher deshalb trotzdem **2** Bytes für das Erweiterungswort reserviert. Es wird von diesen zwei Bytes das zweite, also das niederwertige Byte benutzt. Beispiel:
Wenn im Speicher der nächste Befehl mit o013074, o177401 codiert ist, so lautet der Befehl „MOVE.B xxx,D3", wobei die direkten Daten ‚xxx' in dem Wort nach dem Opcode stehen, hier also o177401 = **$FF01**. Nach dem Befehl steht in D3 ein Wert von **$xxxxxx01**. Es wird eben nur *ein* Byte verändert („MOVE.B"), und es wird das zweite, *niederwertige* Byte des Erweiterungswortes benutzt (‚01' und nicht ‚FF').
Bei drei Befehlen („ANDI", „EORI" und „ORI"), kann man die Adressierungsart „Direkte Daten" nicht verwenden. Man kann aber mit diesen logischen Befehlen das Bitmuster im Statusregister (SR oder CCR) verändern. Da dies aber nur bei diesen drei Befehlen geht, wurde dafür nicht eine eigene Adressierungsart geschaffen. Man könnte sie dennoch etwa mit

4.13. Statusregister

(SR oder CCR; M=7, R=4)
Beispiel: ANDI.B $FE,**CCR**

beschreiben. Es handelt sich also um denselben Code (M=7, R=4) wie bei der direkten Adressierung. Bei den drei obengenannten Befehlen wird aber in diesem Fall das *Statusregister* adressiert.
Der Befehl im Beispiel verknüpft den Inhalt des CCR (unteres Byte des Statusregisters, beinhaltet die Flags) und das Byte $FE logisch mit UND und speichert das Ergebnis im CCR. Dadurch wird das Carry-Flag (C) auf 0 gesetzt (vgl. Abb. 1).
Das waren jetzt alle Adressierungsarten auf einen Blick. Sie sind in der Tabelle 1 noch einmal zusammengestellt.

Schlußbemerkung

Man kann bei einem Befehl wie etwa „MOVE.s ea1,ea2" für die Quelladresse ‚ea1' und die Zieladresse ‚ea2' viele unterschiedliche Adressierungsarten einsetzen. Wie schon erwähnt,

Semjan presents...

• Champion Karte für Apple //+,e

- CIRTECH Parallele Text- u. Graphik-Druckerkarte komplett mit Kabel.
- Option 64K RAM Zwischenspeicher, 40/80 Zeichen Dump zu jeder Zeit.
- Umfangreiche Kontroll-Steuerbefehle z.B. Zeichensatzwahl, etc.
- Einsatz von DOS 3.3, PRODOS (Appleworks), UCSD PASCAL, CP/M.
- Champion-Karte voll Graphik fähig, Apple //e Graphik.
- Serieller Ausbau möglich, 75 bis 9600 Baud Ein- bzw. Ausgabe.
- Mischbetrieb, parallel und seriell, zu jeder Zeit möglich.
- Drucker: Epson, Centronics, Brother, Itoh, Imagewriter I/II, etc.

K030 //+,e Champion Interface	**DM 222,00**
K031 //+,e Champion für Imagewriter	**DM 299,00**
K033 //+,e Champion Interface 64K RAM	**DM 476,00**

• Uni-Mate UniDisk Software

- Einsatz der UniDisk Laufwerke ab sofort mit DOS 3.3, CP/M 2.20
- CP/M 2.23, PASCAL 1.1 und PASCAL 1.2. Einmalige Installation.
- Einsatz von UniDisk und DISKII Laufwerken möglich.
- DOS 3.3 bootbar, Speicherkapazität 800K, INIT möglich.
- CP/M 2.2 und 2.23 bietet 768K Speicherk.. Formatierung möglich.
- PASCAL bietet 768K Speicherkapazität. Formatierung möglich.

K045 // CIRTECH Uni-Mate Software DM 127,00

• CIRTECH EPROM SYSTEM

- Super Eprom Programmier-System. System komplett mit aller Software.
- Einsatz als Text/Grafik Interface und Eprom-Programmier-System.
- Einsatz als zweifache I/O Karte möglich.
- Einsatz von EPROM: 2716, 2732 2732A, 2764, 27128 und 27256.

K080 //+,e Eprom Programmier-System DM 408,00

Auf alle CIRTECH-Produkte volle 12 Monate Garantie.
Neu: Katalog 2/86 anfordern. Händleranfragen willkommen!
Fragen Sie Ihren Händler nach CIRTECH-Produkten!

M. Semjan Computer Systeme

Postfach 90 01 64, 6000 Frankfurt/Main 90, Tel. 069-70 18 53
Telex 051 933 521 dmbox g. Ref: Box: DM3:SEMCOM, Mailbox-Adresse: DM3 SEMCOM

kann man nicht bei *allen* Befehlen völlig frei eine Adressierungsart wählen. Beispielsweise kann man die logischen Operationen (AND, EOR, OR und NOT) nicht in einem Adreßregister durchführen. Bei diesen Befehlen ist also die Adressierungsart „Adreßregister direkt" nicht erlaubt. (In einem Adreßregister stehen ja Adressen, bei denen i.a. logische Operationen auch nicht sehr sinnvoll sind.)

In der Tabelle mit den Opcodes und Taktzyklen (im Anhang) ist angegeben, welche Adressierungsarten bei den einzelnen Befehlen möglich sind und welche nicht.

5. Datentypen

Bevor nun auf den Befehlssatz vom 68000 näher eingegangen wird, wollen wir noch einen kurzen Blick auf die Operanden werfen, mit denen man arbeiten kann. Wie wir schon bei den Adressierungsarten gesehen haben, gibt es Daten, die 1 (**Byte**), 2 (**Word**) oder sogar 4 Bytes (**Long**) lang sind. Diese Daten sind im Speicher in der Reihenfolge High-Byte, ..., Low-Byte untergebracht.

Daneben können z.B. aber auch **einzelne Bits** bearbeitet werden. Die Numerierung der Bits beispielsweise in einem Datenregister sieht man in der Abb. 1. Bit 0 ist also das niedrigstwertige Bit („ganz rechts").

Außerdem ist der 68000 in der Lage, **BCD**-codierte Zahlen zu verarbeiten („BCD" = Binary Coded Decimal). Der 6502-Programmierer kennt diesen Zahlencode: Wenn das D-Flag im Statusregister auf 1 ist, werden alle Additionen und Subtraktionen in dem sog. „Dezimal-Modus" durchgeführt. Beim 68000 gibt es kein D-Flag. Da ohnehin nur die arithmetischen Befehle davon betroffen sind, gibt es beim 68000 eigene BCD-Befehle zur Addition, Subtraktion und Negation („ABCD", „SBCD" und „NBCD").

Noch ein letztes Wort zu den Adressen, bevor wir ganz auf den Befehlssatz überschwenken. Der Adreßbus beim 68000 ist ja 24 Bit breit. Also benötigt man für Adressen nur jeweils **3** Bytes. Weil der 68000 aber immer gerne mit einer *geraden* Zahl von Bytes arbeitet, sind auch die Adreßregister 32 Bit (**4** Bytes) breit. Wenn man nun auf eine Adresse zugreift, die in einem Adreßregister gespeichert ist (z.B. bei „MOVE.L (A0),D6"), so werden von diesen 32 Bits nur die unteren 24 Bits an den Adreßbus gelegt. Der Inhalt des höchstwertigen Bytes ist also bei der Adressierung gleichgültig; er wird aber bei manchen Operationen mit berücksichtigt, z.B. bei „MOVE.L A0,D1" oder bei „ADD.L $12000000,A3" (Addition).

6. Befehlssatz

Nun kommen wir endlich zum Befehlssatz des Prozessors, der für die meisten Leser am interessantesten sein dürfte. Die einzelnen Befehle des 68000 werden hier nur kurz zusammengefaßt. In der **Tabelle 2** sind die Befehle etwas ausführlicher in alphabetischer Reihenfolge aufgelistet. So mancher Leser wird wohl auf den ersten Blick etwas enttäuscht oder überrascht sein: Es gibt nämlich gar nicht so viele Befehle. Doch alle diese Befehle stehen für eine ganze Gruppe von Operationscodes, so daß sich beim 68000 eine Gesamtzahl von sage und schreibe 39696 (legalen) Opcodes ergibt. Eine Vorstellung dieser Vielzahl von Opcodes vermitteln auch die Befehlstabellen im Anhang zu diesem Artikel.

Weiterhin muß man sagen, daß manche Befehle der Tabelle 2 noch weiter *aufgegliedert* werden können. Der Befehl „ADD" steht beispielsweise für insgesamt 5 Befehle, die allesamt eine Addition bewirken. Wenn die Addition z.B. in einem Adreßregister stattfinden soll, so lautet der Befehl „ADD**A**". Soll eine Addition mit einer direkt angegebenen Zahl ausgeführt werden (Immediate), so lautet der Befehl „ADD**I**". In dieser Art gibt es noch andere Erweiterungen, die man an den Grundbefehl „ADD" anhängen kann. Welche Befehle auf diese Art erweitert werden können, sieht man in der **Tabelle 3**.

Im folgenden werden die Befehle nicht alphabetisch geordnet vorgestellt, sondern in Gruppen eingeteilt. Bei jeder Gruppe werden die dazugehörigen Befehle aufgelistet. Es kann aber nur auf einige Befehle besonders hingewiesen werden. Die Arbeitsweise der anderen Befehle ist in den Tabellen im Anhang angegeben. Manche Befehle sind schon von anderen Mikroprozessoren her bekannt, andere werden weiter unten in kleinen Beispielprogrammen näher erläutert.

6.1. Befehle zum Datentransport

(EXG, LEA, LINK, MOVE, MOVEM, MOVEP, PEA, SWAP, UNLK)

Die am meisten benutzten Befehle sind dafür zuständig, Daten zwischen Registern untereinander, im Speicher oder zwischen Registern und dem Speicher zu transportieren. Der Standardbefehl für diese Operation lautet „MOVE". Allein dieser Befehl ist so vielfältig, daß es insgesamt 9898 Opcodes für diesen einen Befehl gibt.

Daneben kann man z.B. mit „EXG" den Inhalt zweier Register vertauschen, mit „SWAP" die oberen 16 Bits eines Registers mit den unteren 16 Bits vertauschen, Speicherplatz auf dem Stack reservieren (LINK) bzw. wieder freigeben (UNLK).

6.2. Arithmetische Befehle

(ADD, CLR, CMP, DIVS, DIVU, EXT, MULS, MULU, NEG, SUB, TAS, TST)

Neben Addition und Subtraktion von ganzen Zahlen kann der 68000 auch multiplizieren (MULU, MULS) und dividieren (DIVU, DIVS)! Auf diese beiden neuen Befehle wird später noch genauer eingegangen.

Daneben gehören zu dieser Gruppe Befehle zum Vergleichen (CMP), Negieren (NEG) und Löschen (CLR) von Daten.

Wie wir weiter oben schon gesehen haben, gibt es eine ganze Reihe von Additionsbefehlen (ADD, ADDA, ADDQ, ADDI, ADDX), die alle auf dem Grundbefehl „ADD" aufbauen (siehe auch Tabelle 3). Von diesen Befehlen will ich hier noch einen besonders hervorheben, nämlich den Befehl „ADDQ" („Q" = Quick, Schnell).

Mit diesem Befehl kann man direkte Daten aus dem Bereich von 1 bis 8 zu einem Register oder einer Speicherzelle addieren. Dies geht deshalb schneller als mit dem Befehl „ADDI", weil die Zahl (von 1 bis 8) direkt *im Opcode* des Befehls angegeben ist. Sobald der Prozessor also den Befehl entschlüsselt hat, kann auch schon die Addition erfolgen. Beim 6502 gibt es einen ähnlichen Befehl: „INC". Damit kann man aber nur eine 1 addieren. Beim 68000 gibt es also einen Inkrement-Befehl (natürlich auch einen Dekrement-Befehl „SUBQ"), mit dem man Werte von 1 bis 8 direkt addieren kann.

6.3. Logische Befehle

(AND, OR, EOR, NOT)

Hier gibt es im Vergleich zum 6502 nichts Neues. Es gibt also die logischen Operationen UND, ODER, Exklusiv-ODER sowie die Negation.

6.4. Schiebe-Befehle

(ASL, ASR, LSL, LSR, ROL, ROR, ROXL, ROXR)

Anders verhält es sich mit den Schiebe- und Rotierbefehlen, die beim 68000 *vollständig* vorhanden sind. Man kann also sowohl *logisch* als auch *arithmetisch* nach links und nach rechts schieben. Der Unterschied zwischen dem logischen und dem arithmetischen Schieben besteht in der Art, in der die Daten interpretiert werden: Beim logischen Schieben sind die Daten einfach Folgen von den booleschen Werten 0 und 1, beim arithmetischen Schieben dagegen binäre Zahlen mit Vorzeichen. Wie sich dieser Unterschied auswirkt, sieht man in der **Abb. 4**.

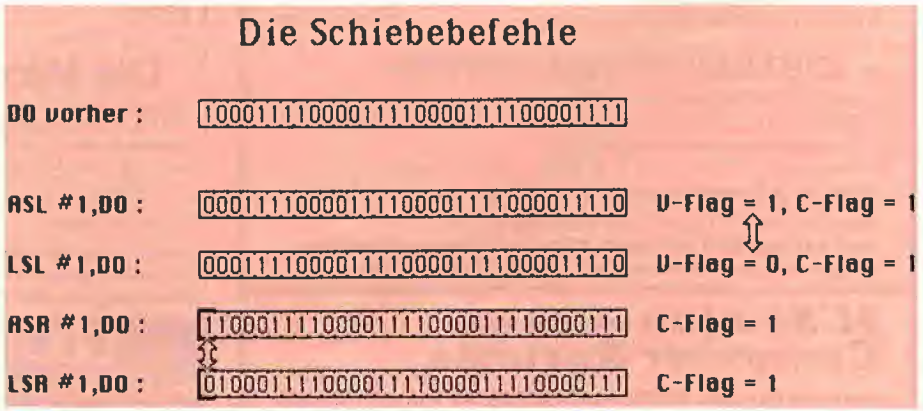


Abb. 4

ZBasic

« Einige Leistungsmerkmale »

- 400-seitiges, ausführliches Handbuch
- Super schnelles Compilieren (s. Benchmarktest)
- Computer unabhängiges Grafik- und File Handling
- neue mächtige Grafikbefehle (CIRCLE,CIRCLE FILL,BOX, BOX FILL,PLOT lines,points etc.,POINT COLOR und FILL)
- Bis zu 54 Stellen Rechengenauigkeit
- Einzelschritt Debugging sowie Fehleranzeige auf der Linie
- Direkte Befehle wie in einem Interpreter
- Nur RUN schreiben um zu compilieren
- Compiliert alte Basic Programme (Gesichert in ASCII)
- Loops: WHILE-WEND,DO-UNTIL,FOR-NEXT-STEP
- Wird geliefert mit: QUICK Sort & SHELL Sort source code

Benchmarktest

	Sieve	Shell
Apple IIe,c		
ZBasic	486	286
Applesoft™	3432	5400
MSDOS™		
ZBasic	13.7	19.2
Turbo™	14.1	28
Basica™	2190	3105
Macintosh™		
ZBasic	7	11
Mbasic™	684	350
Z80™		
ZBasic	23.4	30.8
Mbasic™	2160	4195

Erhältlich für:
Apple IIe/IIc, Mac, Mac Plus, Mac XL, Z80, IBM & Kompatible

Zum sensationellen Preis von nur:

Fr. 220.--
DM 258.--

Generalvertretung
ComputerWorks
Leimenstrasse 49
4051 Basel
Tel: 061 22'50'22

Es gehört inzwischen zum Standard für ein modernes Rechnersystem, mit einer Festplatte ausgerüstet zu sein. Erst dadurch erlangt der Rechner die Qualität in der Datenverarbeitung, wie sie bei professionellen Anwendungen verlangt wird.

Beim Apple wird einfach das Netzteil herausgenommen und dafür das 10/20 MByte MEGA-CORE eingesetzt. Ab dann warten vier Betriebssysteme (DOS, CP/M, UCSD-Pascal, ProDOS) auf Ihr Kommando. Welcher Profirechner kann das schon?

Fragen Sie uns nach Fakten, Preisen, Bezugsquellen und holen Sie sich für 5,-- DM unsere Demo-Diskette.

Ein Produkt von:

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

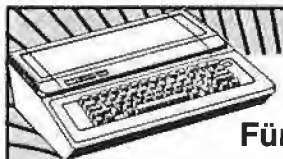
Die Harddiskcontroller-Spezialisten

WARGAMES

Militärische Konfliktsimulationen mit bis zu 100 (!) Stunden Spieldauer. Die Herausforderung an jeden Strategiespieler. Farbinfo für APPLE gegen DM 0,80 Rückporto.

THOMAS MÜLLER
COMPUTER—SERVICE
Postfach 2526 7600 Offenburg

SSI



Für Apple II, IIe

Z-80-Karte	69,–	**80-Zeichen-Karte** mit Softswitch, nur für II Plus kompat.	149,–
Disk-Interface	69,–		
Centronics-Interf. m. Kabel	79,–	**Speech-Karte**	55,–
16-K-Ram-Karte	79,–	**Clock-Karte**	99,–
128-K-RAM-Karte	199,–	**Komp 2E** Apple 2E kompatible. Rechner 64K im 2E-Design, ohne Firmware.	699,–
256-KB-RAM-Karte	299,–		
Motherboard 64K II Plus kompat. 6502, Z80, 64K ohne Firmware	292,–	**80Z + 64K-Karte** für 2E kompatible.	99,–
		Motherboard 2E 2E kompatible ohne Firmware.	299,–

Händleranfragen erwünscht!
Apple-Info 1,– DM (Porto)

Klaus Jeschke
Hard-, Software
Viertstr. 3-13
6233 Kelkheim
☎ (0 61 98) 90 69

Ausgabe und Eingabe mit TYPETERM®

im Slot Ihres

APPLE II/IIe

Das bedeutet: Computer-textverarbeitung von der Schreibmaschinentastatur! Steckerfertig ohne Umbau.

Die neue CE-550!
mit TYPETERM DM 1.398,–

TYPETERM-Interface **DM 479,–**
für alle BROTHER-Typenrad-schreibmaschinen ab AX-30 bis EM-811
(auch für Vorgängermodelle!)
Paketpreis z. B.:

EM-501 mit TYPETERM ..... DM 2136,–
EM-511 mit TYPETERM ..... DM 2412,–
EM-701 mit TYPETERM ..... DM 2468,–

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 0 51 39 - 8 73 93

Ausgabe mit TYPETERM® JUNIOR

im Slot Ihres

APPLE II/IIe

Paketpreis **DM 899,–**
Schreibmaschine AX-10 mit Interface TYPETERM JUNIOR, steckfertig.

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 0 51 39 - 8 73 93

Rotieren kann man wie beim 6502 entweder mit oder ohne dazwischengeschaltetem Carry-Flag. Neu beim 68000 ist, daß man mit einem Befehl um *mehrere* Bits gleichzeitig schieben/rotieren kann!

6.5. Bit-Manipulation-Befehle
(BTST, BSET, BCLR, BCHG)

Mit diesen Befehlen lassen sich einzelne Bits im Speicher oder in einem Register auf 1 oder 0 setzen, umändern oder einfach auf ihren Zustand testen. Beim 6502 gibt es so etwas Ähnliches mit dem Befehl „BIT" (zumindest den Bit-*Test*).

6.6. BCD-Befehle
(ABCD, SBCD, NBCD)

Mit diesen Befehlen können Zahlen, die im BCD-Code codiert sind, addiert, subtrahiert und negiert werden. Es gibt für diese Zahlen aber keine Multiplikation oder Division. Wie weiter oben schon erwähnt, gibt es beim 68000 kein D-Flag, um die Verarbeitung von BCD-codierten Zahlen zu ermöglichen. Dafür stehen nun diese drei eigenen Befehle zur Verfügung.

6.7. Sprungbefehle
(Bcc, DBcc, Scc; BRA, BSR, JMP, JSR; RTR, RTS)

Diese Befehle sind gegenüber dem 6502 stark erweitert worden. Neben den Sprüngen (BRA, JMP), Unterprogrammaufrufen (BSR, JSR) und -rücksprüngen (RTR, RTS) gibt es die bedingten Sprünge. Beim 68000 gibt es insgesamt **16** Bedingungen, die getestet werden können. Soll zum Beispiel gesprungen werden, wenn das Ergebnis einer Operation größer als Null ist, so braucht man dafür nicht mehr wie beim 6502 zwei Befehle, z.B.

```
        :
        BEQ Label
        BPL Größer
Labell  :
        :
        :
sondern es reicht der eine Befehl
        :
        BGT Größer
        :
```

(Branch on Greater Than). Das macht das Programmieren sehr angenehm. Ebenfalls sehr nützlich ist der Befehl „BRA", bei dem auf jeden Fall verzweigt wird. Dadurch erspart man sich solche Konstruktionen wie

```
        :
        CLC
        BCC Label
        :
```

Eine Liste der zur Verfügung stehenden Bedingungen ist im Anhang enthalten. Die relativen Verzweigungen sind beim 68000 nicht mehr auf einen Bereich von 8 Bits beschränkt (-128 bis +127), sondern es stehen 16 Bits für den Offset zur Verfügung.
Sehr gut bewährt hat sich auch die Möglichkeit, Unterprogramme ebenfalls *relativ* zum Programmzähler aufrufen zu können. Der Befehl dazu heißt „BSR" (Branch to SubRoutine). Mit diesem Befehl kann man lange und komplexe Programme schreiben, die völlig beliebig im Speicher *verschiebbar* sind. Alle Sprünge und Unterprogrammaufrufe können nämlich mit den Befehlen „BRA" bzw. „BSR" relativ gemacht werden, so daß sie keine absolute Adresse enthalten.
Ein neuer Befehl ist weiterhin besonders zu erwähnen, nämlich der Befehl „DBcc" (Decrement and Branch **until** cc). Mit diesem Befehl lassen sich sehr schön Schleifen programmieren. Dies wird weiter unten gezeigt.

6.8. Systembefehle
(RESET, RTE, STOP; TRAP, TRAPV, CHK; sowie Befehle mit dem Statusregister)

Zu dieser Gruppe gehören alle Befehle, die das Rechnersystem in einen bestimmten Zustand bringen. Da gibt es zum Beispiel den Befehl „RESET". Dieser Befehl sendet ein Signal an alle angeschlossenen Peripheriegeräte, die dadurch initialisiert (normiert) werden.
In dieser Gruppe sind auch Befehle, die die Flags im Statusregister verändern sowie Befehle, die eine Unterbrechung auslösen. Diese *Unterbrechungen* kennt auch der 6502-Programmierer von dem Befehl „BRK". Dieser Befehl heißt beim 68000 „TRAP". Daneben gibt es die Befehle „TRAPV" und „CHK", die eine Unterbrechung nur dann auslösen, wenn ein Überlauf aufgetreten ist (V-Flag = 1) bzw. wenn der Inhalt eines Registers nicht in einem bestimmten Bereich liegt.
Leider kann hier nicht genauer auf die sehr interessanten Unterbrechungsmöglichkeiten des 68000 eingegangen werden. Die Leser, die dazu mehr erfahren möchten, seien auf die weitere Fachliteratur verwiesen.

Dies war ein kurzer Überblick über die Befehle des 68000. Manche dieser Befehle werden in den weiter unten folgenden Beispielprogrammen näher untersucht. Das ist aber nicht bei allen Befehlen möglich. Im Anhang findet sich dafür eine Liste mit allen Befehlen, deren Opcodes, Taktzyklen und deren Bedeutung.

7. Die Macintosh-Umgebung

Wie weiter oben schon erwähnt, gibt es eine ganze Reihe von neueren Computern, in denen ein 68000-Prozessor eingebaut ist. Mir persönlich stand ein Macintosh (von Apple) zur Verfügung, auf dem auch die Beispielprogramme geschrieben wurden.
Für die Programmierung des Macintosh in Maschinensprache gibt es ein Entwicklungspaket von Apple, in dem u.a. folgendes enthalten ist:

– ein sehr komfortabler und leistungsfähiger Editor zum Erstellen und Editieren der Assembler-Programme
– ein Assembler, dessen Listings leider einen Schönheitsfehler aufweisen, wenn man Vorwärts-Verzweigungen benutzt (s.u.)
– ein Linker, mit dem die übersetzten Programme zu einem Macintosh-Programm zusammengesetzt werden (davon wird gleich noch die Rede sein)
– weitere macintosh-spezifische Programme.

Mit diesen Hilfsmitteln läßt sich sehr komfortabel umgehen, wie man es vom Macintosh her gewohnt ist. In dem mir zur Verfügung stehenden Gerät war zudem eine besondere Speichererweiterung von 2M eingebaut, so daß mit der RAM-Disk auch die Geschwindigkeit beim Programmwechsel (Editor → Assembler → Linker) geradezu beängstigende Dimensionen erreichte.
So weit, so gut. Da es sich bei dem Macintosh aber um einen sehr komplexen Computer handelt, ist das Programmieren in Maschinensprache nicht so einfach, wie man es z.B. vom Apple II her gewohnt ist. Man muß sich jetzt auf einmal um Mäuse, Menüs und Fenster kümmern, was eine nicht unerhebliche Einarbeitungszeit erfordert. Die hier vorgestellten Beispielprogramme wurden als Unterprogramme in ein Testprogramm eingebaut, das in dem Entwicklungspaket enthalten ist. Es wird ein eigenes Menü erzeugt, mit dem die einzelnen Unterprogramme aufgerufen werden können (siehe **Abb. 5**).

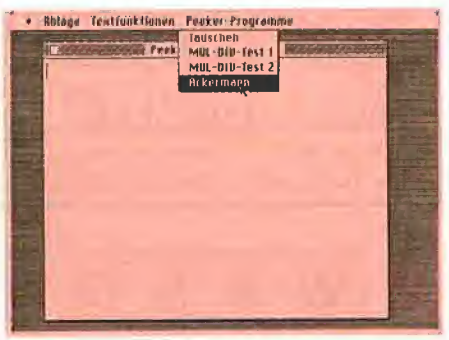


Abb. 5

8. Beispielprogramme

Nun kommen wir endlich zur Sache: Die ersten Programme in 68000-Maschinensprache werden vorgestellt. Es handelt sich dabei natürlich nur um kleinere Beispiele, die ganz bestimmte Punkte verdeutlichen sollen.

8.1. Unterprogramm „Tausch"

Zum Einstimmen zeigt das **Listing 1** ein Programm, mit dem man einen Speicherbereich mit einem anderen (gleichgroßen) Bereich vertauschen kann. Damit man das Ergebnis auch überprüfen kann, wurden die Bereiche so gewählt, daß sie im Bildschirmspeicher des Macintosh arbeiten. Doch zunächst einmal zur Beschreibung des Programms:

Als erstes wird definiert, wo die beiden Bereiche beginnen sollen. Der Bildschirmspeicher des Macintosh beginnt bei der Adresse $07A700. Er hat 342 (= $156) Zeilen mit je 512 (=$200) Pixeln. Der erste Bereich beginnt bei der Zeile $20 (von oben her gerechnet), der zweite bei der Zeile $C0. Die Länge der Bereiche wurde auf $90 Zeilen zu je $40 Bytes gesetzt.
Dann geht das Programm auch schon mit dem ersten unbekannten Befehl los: Er lautet „MOVEM". Mit diesem Befehl kann man mehrere Register gleichzeitig transportieren. Der im

Beispiel verwendete Befehl betrifft die Register D0, D1, A1 und A2. Man könnte den einen Befehl auch durch die folgenden 4 Befehle ersetzen:

```
MOVE.L D0,-(A7)
MOVE.L D1,-(A7)
MOVE.L A1,-(A7)
MOVE.L A2,-(A7)
```

Nun sollte eigentlich klar sein, was diese Befehle machen, denn den „MOVE"- Befehl kennen wir ja schon. A7 ist der Stackpointer, die Adressierungsart ist „Adreßregister mit Predekrement". Mit diesen Befehlen werden demnach nacheinander die Register D0, D1, A1 und A2 auf den Stack geschrieben!
Warum? Unser kleines Unterprogramm benutzt gerade diese 4 Register und verändert deren Inhalt. Es könnte nun so sein, daß das Programm, welches das Unterprogramm „Tausch" aufruft, in diesen Registern gerade etwas gespeichert hat, das nach unserem Unterprogramm weiter verwendet werden soll. Deshalb retten wir am Anfang lieber alle verwendeten Register auf den Stack, von wo sie am Ende auch wieder geladen werden können (s.u.). Dann können wir mit den Registern machen, was wir wollen, und müssen uns nicht um das aufrufende Programm kümmern.

Danach kommt schon wieder ein unbekannter Befehl: „LEA". Er bedeutet „Load Effective Address" und macht nichts anderes, als eine Effektive Adresse in ein Adreßregister zu laden. Was eine Effektive Adresse ist, wurde ja weiter oben erklärt. Nach dem zweiten Befehl steht also in A1 die Adresse vom 1. Bereich, nämlich $07AF00. Der nachfolgende Befehl lädt die Adresse vom 2. Bereich, nämlich $07D700 in das Adreßregister A2.
Nun kommt endlich unser guter alter Bekannter „MOVE". Hier wird eine direkt angegebene Zahl in Register D0 geladen. Aber um was für eine Zahl handelt es sich hierbei? Wir wollen in unserem Programm zwei Speicherbereiche austauschen. Das kann man z.B. Byte für Byte machen. Aber beim 68000 kann man auch 2 Bytes (Word) oder gar 4 Bytes (Long) gleichzeitig transportieren! Das wollen wir hier ausnutzen. In D0 wird also die Länge der Bereiche in „Longs" geladen. Das ist die Länge der Bereiche (in Bytes), geteilt durch 4. (Warum dann noch eine 1 abgezogen werden muß, wird weiter unten erläutert). Dann beginnt mit dem Label ‚TNext' schon die Programmschleife. Zunächst laden wir ein Long (4 Bytes) aus dem Bereich 1 in das Register D1. Wie das geschieht, sollte eigentlich klar sein: In A1 steht die Adresse vom Bereich 1, folglich kann man mit der Adressierungsart „Adreßregister indirekt" ein Long ab dieser Adresse holen.
Danach wird ein Long aus dem Bereich 2 in den Bereich 1 transportiert. Die Quelladresse ‚(A2)' ist klar, A2 zeigt ja auf den Bereich 2. Die Zieladresse ‚(A1)+' gibt zum einen die Adresse vom Bereich 1 an, zum anderen wird nach dem Zugriff das Adreßregister A1 erhöht. Das bedeutet: Nach diesem Befehl steht in A1 die Adresse $07AF04, also die Adresse vom nächsten zu bearbeitenden Long.
Der nächste Befehl transportiert das Long, das wir in D1 gespeichert haben, aus dem Bereich 1 in den Bereich 2. Dabei wird wieder die Adressierungsart „Adreßregister mit Postinkrement" gewählt, so daß nach dem Befehl auch A2 auf die nächste zu bearbeitende Adresse zeigt.

Nun haben wir das erste Long der beiden Bereiche mit Hilfe des Registers D1 ausgetauscht. Außerdem wurden durch die geschickte Wahl der Adressierungsarten die neuen Adressen in A1 und A2 erzeugt (in A1 steht $07AF04, in A2 steht $07D704). Jetzt muß nur noch die Schleife wiederholt werden, bis die Bereiche abgearbeitet sind.

Das wird durch den nächsten Befehl „DBF" erledigt. Es handelt sich hier um einen Befehl der Gruppe „DBcc", wobei die Bedingung ‚cc' mit ‚F' (False) angegeben ist. Diese Befehlsgruppe macht folgendes:

Zunächst wird die Bedingung ‚cc' getestet. Trifft diese Bedingung zu, so macht der Befehl gar nichts (hat also dieselbe Wirkung wie ein „NOP"). Die Bedingung ‚F', die hier gewählt wurde, steht für „False". Diese Bedingung ist nie erfüllt, also geht der Befehl noch weiter.

Wenn nämlich die Bedingung nicht erfüllt ist, so wird das Datenregister, das im Befehl angegeben ist (hier D0), um 1 erniedrigt („Decrement"). Wenn der Wert im Register danach ungleich -1 ist, so wird zu dem angegebenen Label (hier ‚TNext') verzweigt. Wenn sich aber im Register der Wert -1 ergibt ($FFFF), so wird nicht verzweigt; das Programm macht dann mit dem nächsten Befehl weiter.

Mit unserem Befehl wird also Register D0, das die Zahl der Schleifendurchläufe zählt, erniedrigt. Wenn die Bereiche noch nicht abgearbeitet wurden, wenn also D0 noch größer als Null ist, so wird daraufhin die Schleife wiederholt. Das ganze geht so lange, bis D0 am Ende den Wert ‚-1' erreicht. Dann ist die Schleife beendet, und das Programm geht nach dem ‚DBF' weiter. Den Befehl „DBF" kann man im Assembler auch als „DBRA" (Decrement and BRAnch) angeben. Das deutet darauf hin, daß es nur ein Dekrement und ein eventuelles Verzweigen (BRAnch) gibt, ohne daß eine Bedingung ‚cc' getestet wird (‚F' ist ja nie erfüllt). Welche der beiden Schreibweisen verwendet wird, hängt vom persönlichen Geschmack ab.

Wenn am Anfang der Schleife in D0 eine ‚0' gestanden hätte, was wäre dann passiert? Die Schleife wäre dann einmal durchlaufen worden, bis zum „DBF". Dort wird D0 um 1 erniedrigt und erreicht gleich den Endwert von -1. In diesem Fall wird die Schleife also genau einmal durchlaufen. Man sieht, daß man für n Durchläufe der Schleife am Anfang den Schleifenzähler auf n-1 setzen muß. Dies ist der Grund dafür, daß ganz zu Beginn in D0 die Zahl der Longs („Länge/4") *minus 1* geladen wurde.

Nachdem die Schleife vollständig abgearbeitet wurde, sind wir auch schon fertig: Die beiden Bereiche sind ausgetauscht worden. Da wir am Anfang des Unterprogramms die Register D0, D1, A1 und A2 auf den Stack gerettet hatten, müssen wir sie jetzt am Ende der Routine wieder vom Stack herunterholen. Das geht wieder mit dem „MOVEM"-Befehl: Diesmal wird als Quelladresse ‚(A7)+' angegeben, so daß die Register tatsächlich vom Stack geholt werden. Die Register werden dabei automatisch genau in der richtigen Reihenfolge vom Stack geholt, also zuerst A2, dann A1, D1 und am Ende D0.
Das Programm wird ganz normal mit einem „RTS" beendet. Das Ergebnis des Programms sieht man in der **Abb. 6**. Es handelt sich um den Bildschirm aus Abb. 5, wobei die beiden angegebenen Bereiche vertauscht wurden. Ruft man das Programm nochmals auf, so entsteht wieder der Bildschirm aus Abb. 5. Das Programm „Tausch" arbeitet also korrekt.

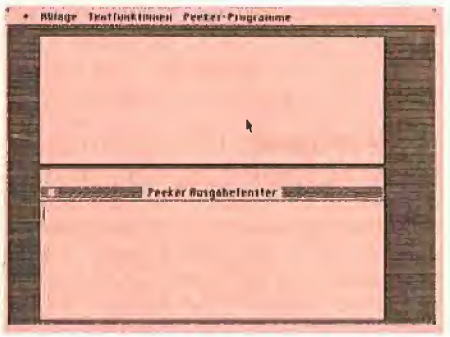


Abb. 6

8.2. MUL/DIV-Test

Mit dem nächsten Programm wollen wir die Befehle zur Multiplikation und Division von ganzen Zahlen näher untersuchen. Der 68000 bietet jeweils zwei Befehle für diese Aufgaben an, zum einen unter Berücksichtigung des Vorzeichens (MULS, DIVS) und zum anderen ohne Vorzeichen (MULU, DIVU). Wir wollen hier nur die Befehle ohne Vorzeichen untersuchen.

Was machen nun diese beiden Befehle? „MULU" multipliziert zwei 16-Bit-Zahlen. Das Ergebnis ist dann eine 32-Bit-Zahl. Bei „DIVU" geht es gerade umgekehrt: Der Dividend ist eine 32-Bit-Zahl, der Divisor eine 16-Bit-Zahl. Beim Ergebnis ist folgendes zu beachten: Der Quotient ist 16 Bit breit und steht am Ende in den unteren 16 Bits des Ergebnisses. Falls bei der Division ein Rest aufgetreten ist, so steht dieser in den oberen 16 Bits des Ergebnisses. Das Ergebnis einer Division ist also 32 Bit lang. Dies wird noch einmal in der **Abb. 7** verdeutlicht.

Mit unserem Unterprogramm wollen wir testen, wie schnell diese eingebauten Befehle sind. Zu diesem Zweck schreiben wir zwei kleine eigene Unterprogramme, die die Multiplikation und die Division „zu Fuß", also im Prinzip nur mit Addition und Subtraktion nachbilden. Anschließend schreiben wir unser Testprogramm „MUL/DIV-Test", bei dem sehr oft multipliziert und dividiert wird. Das Ganze lassen wir dann einmal mit den eingebauten Befehlen und dann mit unseren eigenen Routinen laufen. Dabei stoppen wir jedesmal die Zeit, die insgesamt für das Testprogramm benötigt wird.

Soweit der Überblick über unser Vorhaben. Zunächst schreiben wir unsere Programme für die Multiplikation bzw. Division. Weil man das Ergebnis am Ende besser vergleichen kann, betten wir die 68000-Befehle dazu auch in ein Unterprogramm ein, das dann allerdings nur aus einem Befehl besteht.

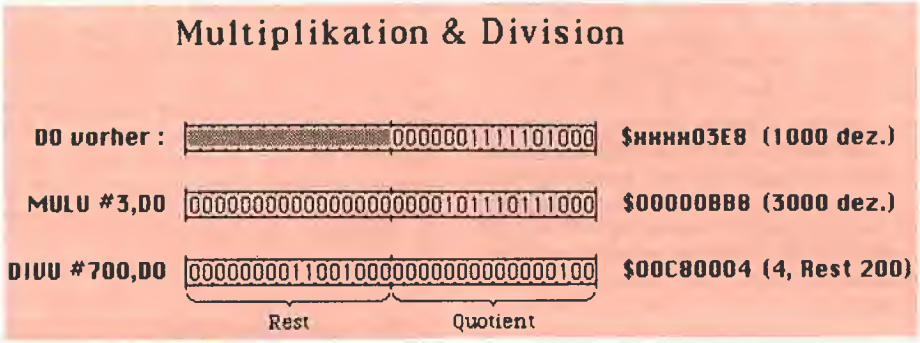


Abb. 7

8.2.1. Multiplikation

Es sollen die beiden 16-Bit-Zahlen in den Registern D0 und D1 multipliziert und das Ergebnis in D0 abgelegt werden. Die beiden Unterprogramme „MUL1" und „MUL2" sind im **Listing 2** abgebildet. „MUL1" benutzt den Multiplikationsbefehl „MULU", die Routine „MUL2" macht dasselbe mit fortlaufender Addition. Es würde hier zu weit führen, die Schritte im einzelnen zu erklären.

8.2.2. Division

Hier wird die 32-Bit-Zahl im Register D0 durch die 16-Bit-Zahl in D1 dividiert. Das Ergebnis steht danach im Register D0 (32 Bits). Die beiden Unterprogramme dazu sind im **Listing 3** gezeigt. „DIV1" benutzt wieder den Divisionsbefehl, „DIV2" macht das mit Subtraktion nach. Auch hier kann nicht auf die einzelnen Schritte der Routine eingegangen werden. Zu erwähnen ist aber das Verhalten bei Überlauf oder Division durch Null. Im letzteren Fall wird eine Unterbrechung (ein „Trap") ausgelöst. Der Prozessor springt dabei zu einer besonderen Routine. Bei einem Überlauf wird das V-Flag im Statusregister gesetzt, die Operanden werden aber nicht verändert. Dieses Verhalten wurde auch in der Routine „DIV2" nachgebildet, obwohl dies beim Testprogramm nicht benötigt wird.

8.2.3. Testprogramm

Nun zum eigentlichen Testprogramm. Normalerweise müßten wir ja zwei Testprogramme schreiben: Eines, das die 68000-Befehle benutzt, und ein anderes mit den nachgebildeten Routinen. Weil diese Programme aber sonst dasselbe leisten, habe ich hier ein einziges Testprogramm geschrieben, das im **Listing 4** zu sehen ist. Das Programm hat zwei unterschiedliche Einsprungstellen „Test1" und „Test2", je nachdem, welche Routinen aufgerufen werden sollen.
An diesen Einsprungstellen geht es zunächst einmal auf bekannte Art und Weise los: Die benutzten Register werden mit dem „MOVEM"-Befehl auf den Stack gerettet. Es sind hier die Register D0, D1, A2 und A3.
Dann kommt der ebenfalls schon bekannte „LEA"-Befehl. Hier wird in das Adreßregister A2 die Adresse der Multiplikationsroutine geladen, in A3 die Adresse der Divisionsroutine. Jedesmal ist die Effektive Adresse mit der Adressierungsart „Programmzähler mit Index" vom Assembler benutzt worden. Wie weiter oben im Abschnitt über die Adressierungsarten schon gesagt, läßt sich diese Adressierungsart mit der „relativen" Adressierung beim 6502 vergleichen. Es wird also in einem Erweiterungswort der Offset zu der gewünschten Adresse festgehalten. Wenn man sich nun im Programm-Listing die Adressen der einzelnen Routinen anschaut, so sieht man, daß zuerst die Routinen „MUL1" und „MUL2", dann „DIV1" und „DIV2", und am Ende das Testprogramm „MUL/DIV-Test" kommt. Deshalb sind die vier Erweiterungsworte mit den Offsets, die ab Adresse $037E, $0382, $038E und $0392 zu finden sind, alle negativ.
Am Ende dieser Sequenz zeigen also die Adreßregister auf die Unterprogramme, die wir zum Test ausführen wollen.
Die Routine „Test1" springt dann mit dem Befehl „BRA" zum Label ‚TestX'. An dieser Stelle geht es gemeinsam weiter. Zunächst wird in D1 ein Startwert von $8000 eingetragen. Später sollen alle Zahlen bis zu diesem Wert mit sich selbst multipliziert, das Ergebnis dann wieder durch diese Zahl geteilt werden. Wir rechnen also zunächst ‚x * x', danach ‚(x ↑ 2) / x', wobei x von $8000 abwärts bis zur 1 laufen soll.
Der nächste Befehl schreibt das Word 8 auf den Stack. Danach folgt der Makro-Befehl „_SysBeep". Hiermit wird beim Macintosh ein Piepser auf dem Lautsprecher ausgegeben. Wie lange dieser Piepser dauern soll, muß vorher auf den Stack geschrieben werden. Bei uns ist die Dauer des Tons ‚8 Einheiten'. Dieser Wert wird von dem Programm, welches den Ton erzeugt, wieder vom Stack gelöscht. Wir können also ganz normal weitermachen, so als ob wir nichts am Stack verändert hätten. Mit diesem Piepser soll der Start des Testprogramms angezeigt werden, damit wir nachher ab hier die Zeit stoppen können, die die Testprogramme benötigen.
Nach dem Piepser kommt die Programmschleife „TestLoop". Zunächst wird der augenblickliche Wert von ‚x' aus D1 in D0 kopiert. Falls er 0 ist, sind wir fertig, und das Programm verzweigt zum Label ‚TestEnd'.
Ansonsten wird nun die Multiplikationsroutine aufgerufen, die uns ‚x * x', also ‚x ↑ 2' in D0 zurückliefern soll. Die Adresse der Routine haben wir nämlich in dem Register A2 gespeichert. Der Befehl „JSR (A2)" springt an die Adresse, die in A2 angegeben ist, also entweder zu „MUL1" oder zu „MUL2", je nachdem, ob wir das Programm über „Test1" oder über „Test2" begonnen haben. Man sieht hier sehr schön, wie elegant sich mit dem 68000 programmieren läßt.
Nach der Multiplikation steht also in D0 ‚x ↑ 2', in D1 steht immer noch ‚x'. Jetzt wird analog die Divisionsroutine aufgerufen, deren Adresse in A3 steht. Das Ergebnis steht danach in D0. Wie müßte denn das Ergebnis aussehen (32 Bits!)? Die Rechnung lautete ‚(x ↑ 2) / x'. Der Quotient (in den unteren 16 Bits von D0) müßte also ‚x' sein, der Rest der Division (in den oberen 16 Bits) Null. Wenn man alle 32 Bits betrachtet, so müßte in D0 also der Wert ‚x' stehen. Das kann man leicht überprüfen: In D1 steht nämlich noch der Wert von ‚x'. Wenn man nun alle 32 Bits von D0 mit denen von D1 vergleicht (mit „CMP.L"), so weiß man, ob die Rechnung korrekt erfolgt ist.

Wie soll es jetzt weitergehen? Bei einem Fehler, wenn also der Vergleich ergeben hat, daß D0 <> D1 ist, soll abgebrochen werden. Ansonsten soll D1 erniedrigt werden, damit mit dem neuen Wert für ‚x' weitergerechnet werden kann. Mit diesem Wert soll dann die Schleife wiederholt werden.
Alles dies kann man mit einem Befehl erledigen: Er lautet „DBNE". Man kann dies als Abkürzung für „Decrement and Branch until Not Equal" auffassen. Was der Befehl macht, wissen wir schon aus dem ersten Beispielprogramm. Er testet, ob die Bedingung „Not Equal" erfüllt ist. Wenn das der Fall ist, ist ein Fehler aufgetreten. In diesem Fall macht der Befehl nichts („NOP"). Sonst (wenn also D0 und D1 gleich waren) wird D1 um 1 erniedrigt. Wenn dabei eine Zahl ungleich -1 entsteht, wird zum Label ‚TestLoop' gesprungen. Weil dies auch der Fall ist, wenn D1 zu Null wird, haben wir diesen Fall zu Beginn der Schleife abgetestet, denn sonst würde durch ‚x', also durch Null dividiert.
Wenn also das Programm nach dem „DBNE"-Befehl angekommen ist, ist ein Fehler aufgetreten. In diesem Fall soll ein langer Piepser ausgegeben werden. Zu diesem Zweck schreiben wir den Wert $40 auf den Stack. Mit dem nächsten Sprungbefehl „BRA" springen wir zum Label ‚TestBeep', wo dann der Ton ausgegeben und das Programm beendet wird.

Wenn aber die Programmschleife korrekt arbeitet, bis D1 den Wert Null erreicht hat, so springt das Programm zum Label ‚TestEnd'. Hier soll ein kurzer Piepser ausgegeben werden zum Zeichen dafür, daß alles korrekt abgelaufen ist und daß die Zeitmessung nun abgebrochen werden kann. Die Dauer des Tons, d.h. 8, wird auf den Stack geschrieben.
Ab dem Label ‚TestBeep' müßte nun alles klar sein. Wie oben beschrieben, wird der Ton ausgegeben. Anschließend werden die Register wieder vom Stack geholt, und das Unterprogramm „Test1" bzw. „Test2" wird verlassen.

Wie sehen nun die Ergebnisse aus, die wir mit unseren Programmen messen können? Das Resultat ist:

```
 1,6 sec für "Test1", und
1Ø,8 sec für "Test2".
```

Man sieht also, daß die eingebauten 68000-Befehle schneller arbeiten als die Routinen „zu Fuß". Abgesehen davon kann man die Befehle auch viel flexibler und komfortabler einsetzen

als die eigenen Routinen. (Es kann natürlich auch so sein, daß meine Routinen nicht optimal sind. Vielleicht kennt oder findet ein Leser schnellere Algorithmen.)

An unserem Beispiel kann man auch noch festhalten, wie elegant sich durch Befehle wie „DBNE" oder „LEA" und „JSR (An)" programmieren läßt.
Als Überleitung zu unserem letzten Beispielprogramm sollten Sie sich einmal fragen, wie man Parameter an Unterprogramme übergeben kann, z.B. an eine Routine, die zwei Zahlen multipliziert, oder an eine Routine, die einen Ton mit bestimmter Dauer ausgibt.
Bei unserer Routine „MUL2" wurden beide zu multiplizierenden Zahlen in den Registern D0 und D1 übergeben, und die Routine gab das Ergebnis ebenfalls im Register D0 zurück. Man kann also Parameter für Unterprogramme in Registern ablegen. Was macht man aber, wenn man nun so viele und so große Parameter hat, daß dadurch die meisten Register belegt sind, obwohl man sie für andere Zwecke brauchte?
Eine in vielen Fällen elegantere Methode wird uns bei der Macintosh-Routine „SysBeep" demonstriert: Der Parameter (die Dauer des Tons) wird auf den Stack geschrieben. Von dort holt sich die Routine die Zahl und wertet sie entsprechend aus. Diese zweite Möglichkeit der Parameterübergabe wird beim Mac vorwiegend verwendet. Man schreibt also einfach die Parameter auf den Stack (natürlich in der richtigen Reihenfolge) und ruft dann das Unterprogramm auf. Das Unterprogramm kann dann die Parameter, die es benötigt, nach Belieben verarbeiten.
Diese zweite Methode wird auch benutzt, wenn Pascalübersetzer eine Pascal-Prozedur in P-Code oder in Maschinensprache übersetzen. Wie das im Prinzip vor sich geht, wollen wir in unserem letzten Beispiel verdeutlichen.

8.3. Ackermann-Funktion

Wir wollen hier die sog. „Ackermann"-Funktion berechnen. Es handelt sich dabei um eine höchst rekursive Funktion, die im Informatik-Unterricht gerne als Übungsbeispiel herangezogen wird. Diese Funktion hat als Parameter zwei natürliche Zahlen ‚x' und ‚y' und berechnet daraus die Ackermann-Funktion ‚A (x,y)'. Dabei ist ‚A' wie folgt definiert:

```
A (Ø,y) := y + 1
A (x,Ø) := A (x-1,1) für x>Ø
A (x,y) := A (x-1, A (x,y-1)) für x,y>Ø
```

Wie man sieht, läßt sich schon für kleine Parameter das Ergebnis nur mit Mühe berechnen. Deshalb soll nun eine Prozedur geschrieben werden, die den Wert ‚A (x,y)' für gegebene Zahlen ‚x' und ‚y' berechnet.
Zu diesem Zweck schreiben wir erst einmal eine solche Prozedur in der Programmiersprache Pascal. Diese Prozedur können wir dann recht problemlos in Maschinensprache übersetzen, da der 68000 gut für solche Aufgaben vorbereitet ist.
Die Pascal-Funktion „Ack" ist schnell geschrieben. **Listing 5** zeigt eine Möglichkeit, sie zu implementieren. Wie kommt man nun von der Pascal-Funktion zu dem Unterprogramm in Maschinensprache? Wir werden hier Schritt für Schritt vorgehen.
Zunächst einmal erwartet die Funktion „Ack" zwei ganze Zahlen als Parameter und gibt ihrerseits das Ergebnis als ganze Zahl zurück. Wie wir uns schon überlegt haben, kann man solche Parameter elegant auf dem Stack übergeben. Wir vereinbaren also, daß das Unterprogramm „Ackermann" wie folgt aufgerufen werden muß:

```
CLR.L   -(A7)
 (Platz machen für Ergebnis)
MOVE.L  "x",-(A7)
 (ersten Parameter auf Stack schreiben)
MOVE.L  "y",-(A7)
 (zweiten Parameter ebenso behandeln)
JSR     Ackermann
 (Funktion aufrufen)
```

Es ist vielleicht im Moment noch nicht ganz einzusehen, warum zuerst ein Long auf dem Stack reserviert werden muß, um das Ergebnis dort zu speichern. Dies wird aber im weiteren Verlauf der Erklärung deutlicher werden.

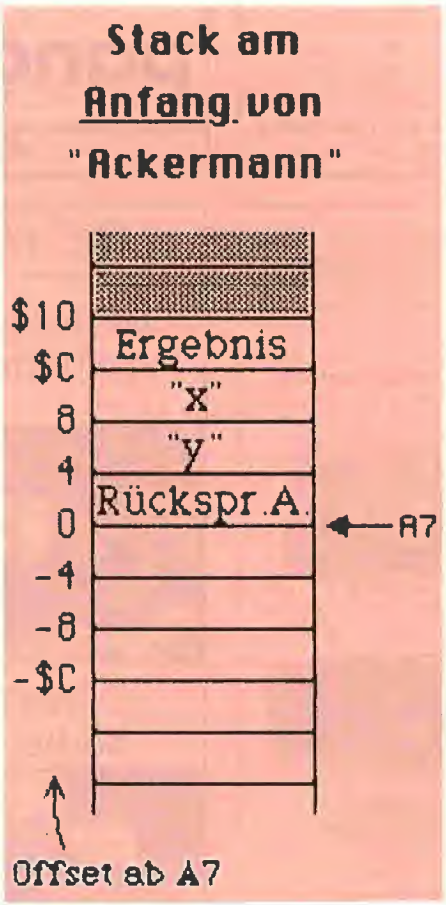


Abb. 8

Wenn nun das Unterprogramm „Ackermann" aufgerufen wurde, sieht der Stack wie in der **Abb. 8** aus. Nun kommt in der Pascal-Funktion „Ack" die Definition der Variablen xMin1 und yMin1. Das heißt, es wird Speicherplatz reserviert, den man unter dem Namen ‚xMin1' ansprechen kann. Weiterhin gilt dieser Name bei Pascal nur in der Funktion selbst, nicht in der äußeren Umgebung. Man spricht hier von „lokalen" Variablen.
Um sich einen solchen Platz für lokale Variablen einrichten zu können, gibt es beim 68000 den Befehl „LINK". Wir betrachten hier beispielsweise den Befehl „LINK A6,-8", der auch in unserem Programm benutzt wird. Man könnte diesen Befehl auch durch die folgenden 3 bekannten Befehle ersetzen:

```
MOVE.L  A6,-(A7)
 (A6 auf Stack retten)
MOVEA.L A7,A6
 (alten SP (A7) in A6 kopieren)
ADDA.L #-8,A7
 (SP um 8 Bytes nach unten setzen)
```

Zunächst einmal wird also der Inhalt vom Adreßregister A6 auf den Stack gerettet. Anschließend wird der momentane Stand des Stapelzeigers in dieses Adreßregister A6 kopiert. Am Ende schließlich wird der Stackpointer um 8 erniedrigt. Das Ergebnis dieses Befehls ist in der **Abb. 9** zu sehen. Wie man leicht erkennen kann, sind nun „unterhalb von A6" 8 Bytes auf dem Stack frei. Wir haben also dort Platz für 2 Longs gemacht. Wie kann man nun auf diese beiden Longs zugreifen? Nichts leichter als das: Wenn man z.B. beide Longs auf Null setzen möchte, so muß man nur

```
CLR.L   -4(A6)
CLR.L   -8(A6)
```

programmieren. Dadurch werden beide Longs gelöscht. Dies wird alles noch klarer, wenn man nachher unser Programm „Ackermann" anschaut.

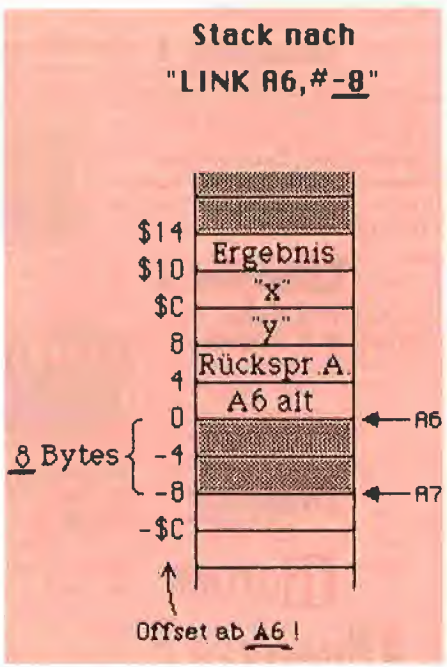


Abb. 9

Wie man Platz für lokale Variablen auf dem Stack reserviert, haben wir nun gesehen. Wie bekommt man am Ende seines Unterprogramms den Stack aber wieder in Ordnung?

Auch dafür gibt es einen Befehl: „UNLK A6". Er dreht die Wirkung des „LINK"-Befehl praktisch um. Die Wirkung:

```
MOVEA.L A6,A7
 (SP wieder auf den alten Stand bringen)
MOVEA.L (A7)+,A6
 (alten Inhalt von A6 wiederholen)
```

Zuerst wird der alte Stand des Stackpointers, der im Register A6 festgehalten wurde, wieder in A7 geholt. Dadurch ist der Platz für die lokalen Variablen praktisch wieder frei. Nun muß nur noch der alte Wert von A6 wieder vom Stack geholt werden, und schon ist alles wie früher (wie in Abb. 8).
Mit diesem Vorwissen kann nun das Unterprogramm „Ackermann" schnell geschrieben werden. Als Voraussetzung kann man annehmen, daß zu Beginn der Routine der Stack wie in Abb. 8 aussieht. Dann geht das Programm in **Listing 6** los:
Zunächst wird mit dem „LINK"-Befehl Platz für die beiden Longs ‚xMin1' und ‚yMin1' gemacht, wie es oben beschrieben wurde. Der Stack sieht jetzt wie in Abb. 9 aus. Anschließend könnten mit einem „MOVEM"-Befehl wie gewohnt die verwendeten Register gerettet werden. Da wir aber in unserem Unterprogramm keine (!) Register verwenden, brauchen wir auch keinen „MOVEM"-Befehl.

Nun geht es mit dem „Rumpf" der Pascal-Funktion „Ack" weiter: Dort wird getestet, ob der Parameter ‚x' auf Null war. Wo steht denn nun aber dieser Parameter? Ein Blick auf Abb. 9 zeigt: Man kann ihn $C Bytes über A6 finden! Damit wir uns das nicht jedesmal merken müssen, folgen nun im 68000-Programm einige Vereinbarungen, wie der Stack aussehen und wo man die einzelnen Variablen finden kann:
Der Parameter ‚x' beispielsweise liegt $C Bytes über A6. Mit dem Befehl „x EQU $C(A6)" wird nun festgehalten, daß ‚x' eben $C Bytes über A6 zu finden ist. Der Assembler setzt nun jedesmal, wenn wir ‚x' schreiben, den String „$C(A6)" ein. Genauso verfahren wir mit dem zweiten Parameter ‚y' und mit den beiden lokalen Variablen ‚xMin1' und ‚yMin1'. Jetzt fehlt nur noch das Ergebnis, dessen Ort wir uns unter dem Namen „Result" merken. Durch diese Vereinbarungen läßt sich nun sehr elegant programmieren, wie man gleich sehen wird.
Jetzt programmieren wir den Test, in dem überprüft wird, ob ‚x' auf Null ist: Mit dem Befehl „TST.L x" wird der Parameter ‚x' mit Null verglichen. Wenn ‚x' ungleich Null ist, so wird zum Label ‚xGT0' gesprungen. Ansonsten ist das Funktionsergebnis ‚y+1'. Das wird mit den nächsten beiden Befehlen bewerkstelligt, die eigentlich klar sein sollten. Damit können wir nun zum Ende der Funktion, zum Label ‚Ack-End' springen.
Wenn ‚x' größer als Null ist, kommt das Programm zum Label ‚xGT0'. Dies entspricht der Stelle nach dem ersten „else" im Pascal-Programm. Dort geht es weiter mit der Berechnung von ‚xMin1', die sich nun auch im Assemblerprogramm anschließt. Der zweite Test ‚y=0' verläuft genau wie der erste. Wenn ‚y' größer als Null ist, erfolgt ein Sprung zum Label ‚yGT0'.

Nun wird es interessant: Wenn nämlich ‚y' doch Null ist, muß die Funktion „Ackermann" rekur-

Listing 1: Tausch

```
; Dieses Unterprogramm tauscht die Inhalte zweier gleichgroßer Speicher-
; bereiche gegeneinander aus. Die Bereiche sind so gewählt, daß sie im
; Bildschirmspeicher des Macintosh liegen, damit das Ergebnis überprüft
; werden kann.
; Zunächst einige Definitionen:

  ØØØ7A7ØØ    Bildschirm  EQU  $Ø7A7ØØ          ; Anfangsadr.v. Bildschirm
      ØØ4Ø    Zeile       EQU  $4Ø               ; Zahl der Bytes je Zeile
  ØØØ7AFØØ    Bereich1    EQU  $2Ø*Zeile+Bildschirm      ; Adr.v. Ber.1
  ØØØ7D7ØØ    Bereich2    EQU  $CØ*Zeile+Bildschirm      ; Adr.v. Ber.2
      24ØØ    Laenge      EQU  $9Ø*Zeile         ; Länge der Bereiche

; Dann folgt das eigentliche Programm:

                    Tausch                       ; Tauschen zweier Bereiche

Ø2FØ - 48E7 CØ6Ø          MOVEM.L DØ-D1/A1-A2,-(A7)   ; Register retten

Ø2F4 - 43F9 ØØØ7 AFØØ     LEA     Bereich1,A1     ; A1:= Adr.v. Bereich 1
Ø2FA - 45F9 ØØØ7 D7ØØ     LEA     Bereich2,A2     ; A2:= Adr.v. Bereich 2
Ø3ØØ - 3Ø3C Ø8FF          MOVE.W  #Laenge/4-1,DØ  ; DØ:= Zahl der Longs -1

                    TNext

Ø3Ø4 - 2211               MOVE.L  (A1),D1         ; D1:= Long aus Bereich 1
Ø3Ø6 - 22D2               MOVE.L  (A2),(A1)+      ; Long von Ber.2 in Ber.1
Ø3Ø8 - 24C1               MOVE.L  D1,(A2)+        ; D1 in Bereich 2 schreiben
Ø3ØA - 51C8 FFF8          DBF     DØ,TNext        ; Wdh., sooft DØ angibt

Ø3ØE - 4CDF Ø6Ø3          MOVEM.L (A7)+,DØ-D1/A1-A2   ; Register wiederholen
Ø312 - 4E75               RTS                     ; Rücksprung aus Tausch
```

Listing 2: Multiplikationsprogramme

```
; Diese beiden Unterprogramme multiplizieren zwei 16-Bit-Zahlen aus den
; Registern DØ und D1. Das 32-Bit-Ergebnis wird im Register DØ gespeichert.

                    MUL1                         ; Multiplikation mit "MULU"

Ø314 - CØC1               MULU    D1,DØ          ; DØ:= DØ * D1
Ø316 - 4E75               RTS                    ; Rücksprung aus MUL1

                    MUL2                         ; Multiplikation "zu Fuß"

Ø318 - 48E7 6ØØØ          MOVEM.L D1-D2,-(A7)    ; Register retten
Ø31C - 4282               CLR.L   D2             ; D2:= Ø
Ø31E - 34ØØ               MOVE.W  DØ,D2          ; D2:= DØ (-> 32-Bit-Zahl)
Ø32Ø - 428Ø               CLR.L   DØ             ; Ergebnis (DØ):= Ø

                    NextBit

Ø322 - 4A41               TST.W   D1             ; Vergleiche D1 mit Ø
Ø324 - 67ØØ xxxx          BEQ     Mul0k          ; Wenn D1=Ø, Sprung ans Ende

Ø328 - E249               LSR.W   #1,D1          ; Nächstes Bit von D1 in C
Ø32A - 64ØØ xxxx          BCC     BitWasØ        ; Wenn C-Flag=Ø, Sprung
Ø32E - DØ82               ADD.L   D2,DØ          ; Ergebnis (DØ) um D2 erhöhen

                    BitWasØ

Ø33Ø - E38A               LSL.L   #1,D2          ; D2 nach links schieben
Ø332 - 6ØEE               BRA     NextBit        ; Wiederhole die Schleife

                    Mul0k

Ø334 - 4CDF ØØØ6          MOVEM.L (A7)+,D1-D2    ; Register wiederholen
Ø338 - 4E75               RTS                    ; Rücksprung aus MUL2
```

Listing 3: Divisionsprogramme

```
; Diese beiden Unterprogramme dividieren die 32-Bit-Zahl in Register DØ
; durch die 16-Bit-Zahl in Register D1. Das 32-Bit-Ergebnis (Quotient und
; Rest) wird im Register DØ gespeichert.

                    DIV1                         ; Division mit "DIV"-Befehl

Ø33A - 8ØC1               DIVU    D1,DØ          ; DØ:= DØ / D1 (mit Rest)
Ø33C - 4E75               RTS                    ; Rücksprung aus DIV1

                    DIV2                         ; Division "zu Fuß"

Ø33E - 4A41               TST.W   D1             ; Vgl. Divisor (D1) mit Ø
Ø34Ø - 67ØØ xxxx          BEQ     DivØErr        ; Bei Division durch Ø, Sprung
```

siv aufgerufen werden. Ganz oben haben wir ja schon festgelegt, wie der Aufruf des Unterprogramms aussehen soll: Es wird Platz für das Ergebnis gemacht, die Parameter ‚xMin1' und 1 werden auf den Stack geschrieben, und das Unterprogramm wird erneut aufgerufen. Spätestens an dieser Stelle müssen wir uns überlegen, wie der Stack nach dem Unterprogramm aussehen soll.
Wir vereinbaren, daß der Stack bis einschließlich zum Parameter ‚x' aufgeräumt werden soll. Am Ende soll also nur noch das Funktionsergebnis auf dem Stack zu finden sein, wie es in der **Abb. 10** zu sehen ist.

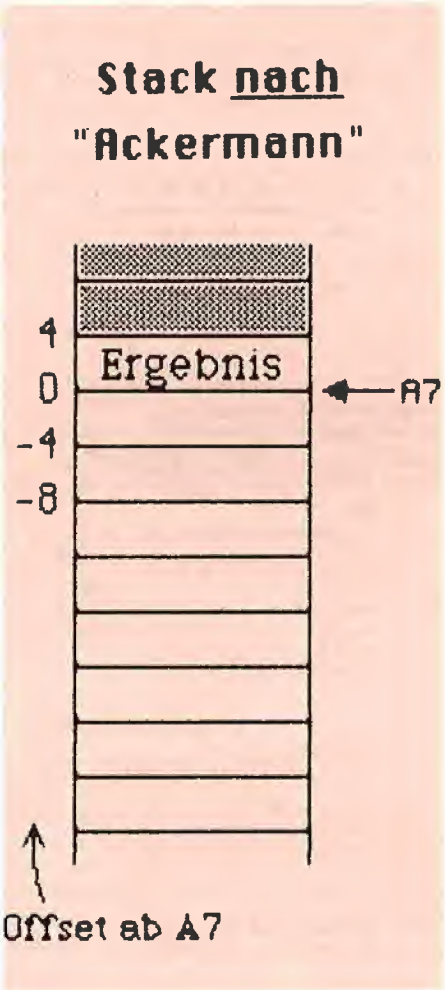


Abb. 10

Wir sind in unserem Assemblerprogramm an der Stelle nach dem rekursiven Aufruf stehengeblieben. Inzwischen wissen wir, daß hier das Ergebnis des rekursiven Funktionsaufrufs, also ‚A (x-1, 1)', auf dem Stack zu finden ist. Mit dem nächsten Befehl wird dieses Resultat als Ergebnis unserer Funktion gespeichert, und das Programm verzweigt an das Ende zu „AckEnd".

Im Prinzip dürfte nun klar sein, wie es nach dem Label ‚yGT0' weitergehen muß: Man muß zweimal ineinander verschachtelt die Funktion „Akkermann" aufrufen. Die Details sind es aber doch wert, genau angeschaut zu werden.

```
Ø344 - 48E7 7ØØØ          MOVEM.L D1-D3,-(A7)  ; Register retten
Ø348 - 4841               SWAP    D1           ; D1 {16..31}:= Divisor
Ø34A - 4241               CLR.W   D1           ; D1 {Ø..15}:= Ø
Ø34C - BØ81               CMP.L   D1,DØ        ; Vgl. Dividend (DØ) mit D1
Ø34E - 64ØØ xxxx          BCC     Overflow     ; Wenn DØ>=D1, zu Overflow

Ø352 - 4242               CLR.W   D2           ; Quotient (D2):= Ø
Ø354 - 76ØF               MOVEQ   #$F,D3       ; Bitzähler (D3):= 16 -1

                       DivLoop

Ø356 - E289               LSR.L   #1,D1        ; D1 um 1 Bit nach rechts
Ø358 - BØ81               CMP.L   D1,DØ        ; Vgl. Rest (DØ) mit D1
Ø35A - 65ØØ xxxx          BCS     LessThan     ; Wenn Rest<D1, Sprung
Ø35E - Ø7C2               BSET    D3,D2        ; D3-tes Bit in D2:= 1
Ø36Ø - 9Ø81               SUB.L   D1,DØ        ; Rest (DØ) um D1 erniedrigen

                       LessThan

Ø362 - 57CB FFF2          DBEQ    D3,DivLoop   ; Wdh. bis DØ=Ø oder D3=-1
Ø366 - 484Ø               SWAP    DØ           ; DØ {16..31}:= Rest
Ø368 - 3ØØ2               MOVE.W  D2,DØ        ; DØ {Ø..15}:= Quotient

                       DivEnd

Ø36A - 4CDF ØØØE          MOVEM.L (A7)+,D1-D3  ; Register wiederholen
Ø36E - 4E75               RTS                  ; Rücksprung aus DIV2

                       Overflow

Ø37Ø - ØØ3C ØØØ2          ORI.B   #2,CCR       ; V-Flag setzen
Ø374 - 6ØF4               BRA     DivEnd       ; Sprung ans Ende

                       DivØErr

                       ; Routine zur Fehlerbehandlung. Hier nicht näher
                       ; ausgeführt.
```

Listing 4: MUL/DIV-Test

```
; Dieses Programm testet die Multiplikations- und Divisionsbefehle des
; 68ØØØ. Hierzu werden diese Befehle sehr oft ausgeführt. Zum Vergleich
; werden genauso oft die eigenen Routinen MUL2 und DIV2 aufgerufen, die
; diese Befehle "zu Fuß" nachbilden.

                       ; Test1 ruft die Routinen MUL1 und DIV1 mit den
                       ; 68ØØØ-Befehlen auf.

                       Test1                     ; Einsprung für 68ØØØ-Befehle

Ø378 - 48E7 CØ3Ø          MOVEM.L DØ-D1/A2-A3,-(A7) ; Register retten
Ø37C - 45FA FF96          LEA     MUL1,A2       ; A2:= Adr.v. MUL1
Ø38Ø - 47FA FFB8          LEA     DIV1,A3       ; A3:= Adr.v. DIV1
Ø384 - 4ECØ xxxx          BRA     TestX         ; Zum weiteren Testprogramm

                       ; Test2 ruft die eigenen Routinen MUL2 und DIV2 auf.

                       Test2                     ; Einsprung für eigene Routinen

Ø388 - 48E7 CØ3Ø          MOVEM.L DØ-D1/A2-A3,-(A7) ; Register retten
Ø38C - 45FA FF8A          LEA     MUL2,A2       ; A2:= Adr.v. MUL2
Ø39Ø - 47FA FFAC          LEA     DIV2,A3       ; A3:= Adr.v. DIV2

                       ; Hier folgt das gemeinsame Testprogramm ...

                       TestX                     ; gemeinsames Testprogramm:

Ø394 - 223C ØØØØ 8ØØØ     MOVE.L  #$8ØØØ,D1     ; D1 (Zähler, x):= $8ØØØ
Ø39A - 3F3C ØØØ8          MOVE.W  #8,-(A7)      ; Dauer des Tons auf Stack
Ø39E - A9C8               _SysBeep              ; Anfangs-Ton ausgeben

                       TestLoop

Ø3AØ - 2ØØ1               MOVE.L  D1,DØ         ; DØ := x (D1)
Ø3A2 - 67ØØ xxxx          BEQ     TestEnd       ; Wenn x=Ø, Sprung ans Ende
Ø3A6 - 4E92               JSR     (A2)          ; Aufruf von MUL1 bzw. MUL2
Ø3A8 - 4E93               JSR     (A3)          ; Aufruf von DIV1 bzw. DIV2

Ø3AA - BØ81               CMP.L   D1,DØ         ; Vgl. Ergebnis (DØ) mit x
Ø3AC - 56C9 FFF2          DBNE    D1,TestLoop   ; Wdh. bis Fehler oder x=-1

                       ; An dieser Stelle ist ein Fehler aufgetreten!

Ø3BØ - 3F3C ØØ4Ø          MOVE.W  #$4Ø,-(A7)    ; Dauer für langen Ton
Ø3B4 - 4ECØ xxxx          BRA     TestBeep      ; Damit Ton ausgeben...

                       ; Wenn alles okay ist, landet das Programm aber an
                       ; dieser Stelle.

                       TestEnd
```

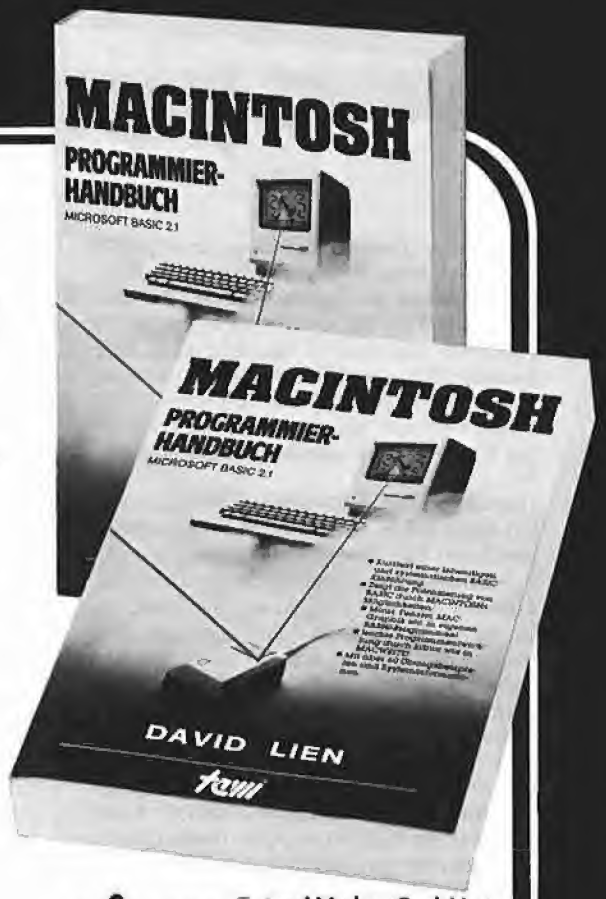
MS BASIC ist eine höchst moderne und leicht erlernbare Programmiersprache!

210 reservierte Begriffe...Programmsynthese aus Modulen (lokale Variablen, Wertübergaben mit COMMON)...Nachladen von Segmenten (CHAIN, mit Parameterübergabe)...Einbinden von MAC-Funktionen (Graphik, Maus, Fenster, Knöpfe usw.)...Fremddateizugriff (von MS-BASIC auf Dateien von z.B. MULTIPLAN, MACPAINT, WORD)...Programmablaufsteuerung (Ereigniserfassung wie ON TIMER, ON MOUSE)...Peripheriebefehle (wie COMI für DFÜ)...strukturierte BASIC-Programmierung ohne Zeilennummern...Gleitkommaarithmetik.....

MACINTOSH und MS BASIC bilden eine komfortable Programmierumgebung!

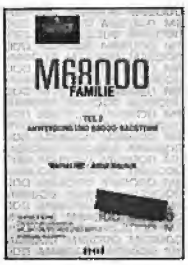
Mehrfachfenster für Simultanbeobachtung (z.B. Fenster 1: Hauptprogramm, Fenster 2: Unterprogramm; oder Fenster 1: Listing, Fenster 2: Ergebnisse)...Mausedition wie in Textprogrammen...Collagetechnik (Programmabschnitte ausschneiden und versetzen)...

HIER DER KURSTEXT einer lebendigen und systematischen BASIC-Einführung von dem US-Professor David Lien. Ein Text für das Selbststudium, das in anspruchsvolle BASIC-Programmierungen mit den Funktionen des MACINTOSH mündet. Ideal als Schultext.
450 Seiten, Softcover, DM 59,–



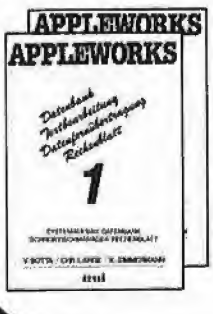
tewi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Weitere te-wi-Bücher



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,–

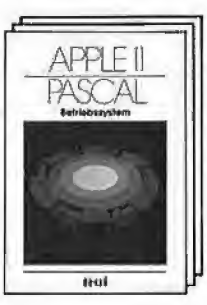
Das APPLE II/II+/IIe/IIc-Handbuch
L. Poole
Erst mit Hilfe dieses Leitfadens werden Sie Ihren Apple II erfolgreich einsetzen, denn Text und Bildmaterial gehen weit über das hinaus, was herstellerseitig an Literatur angeboten wird.
Neu überarbeitet und jetzt um die spezifischen Eigenheiten der Modelle **IIe** und **IIc** erweitert. 472 Seiten, Softcover, DM 66,–



APPLEWORKS integriert in APPLE II, IIe, IIc die Funktionen eines modernen Schreibtisches: Textverarbeitung, Datenbank, Rechenblatt, Datenfernübertragung. Sämtliche System/Anwendungsfragen in 2 Bänden.
Von Botta/Lange/Zimmermann, je 264 Seiten und je DM 49,–

DAS C-BUCH. **NEU**
Textbuch für C-Kurse und C-Anwendungen auf PCs. Beschreibt sämtliche Konstrukte der C-Sprache unter den Betriebssystemen MS DOS, CP/M, ISIS, UNIX und für die C-Compiler von MS, DR, LATTICE, INTEL. Didaktisch und typographisch außergewöhnlich. Mit über 100 lauffähigen Beispielprogrammen für PCs. Zeigt Realisierungen neuester Softwarestrategien in „C".
Von Herold/Unger. Herbst 86.
Etwa 500 Seiten. Softcover. DM 79,–

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,–



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

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

tm 4601

Zunächst wird erst einmal die Variable ‚yMin1' berechnet, wie im Pascal-Programm auch. Dann kommen die Funktionsaufrufe: Der „äußere" Aufruf benötigt ‚xMin1' als seinen ersten Parameter. Nachdem wir für das Ergebnis Platz gelassen haben, wird dieser Wert dann auch auf den Stack geschrieben. Wie sieht aber nun der zweite Parameter aus? Es handelt sich hier um das Ergebnis eines erneuten Aufrufs der Funktion.
Deshalb geht es an dieser Stelle mit dem zweiten Aufruf weiter. Es ist nun schon klar, wie das geschieht: Platz für Ergebnis, 1. Parameter ‚x', 2. Parameter ‚yMin1', Aufruf der Funktion. Das Ergebnis liegt jetzt direkt so auf dem Stack, wie wir es für den äußeren Aufruf benötigen. Wer hier ins Schwimmen kommt, sollte sich den Aufbau des Stacks nach jedem Schritt auf ein Blatt Papier aufmalen. So bekommt man am besten ein Gefühl dafür, was nun wirklich passiert.
Da an dieser Stelle die Parameter korrekt auf dem Stack zu finden sind, kann direkt der äußere Aufruf der Funktion erfolgen. Danach haben wir tatsächlich den Ausdruck ‚A (x-1, A (x,y-1))' berechnet! Dieses Ergebnis wird dann nur noch wie üblich abgespeichert, und schon befinden wir uns am Ende der Funktion, beim Label ‚AckEnd'.

Im Pascal-Programm kommen nun nur noch ein paar „end". Was bei uns kommt, ist fast genauso einfach: Zunächst könnte man an dieser Stelle mit einem „MOVEM"-Befehl die veränderten Register wieder vom Stack holen. Bei uns ist das nicht nötig. Danach wird der Stack zum Teil aufgeräumt, indem wir den Befehl „UNLK A6" ausführen lassen. Danach hat man den Zustand aus Bild 8. Das ist aber noch nicht das gewünschte Ergebnis, das man in Bild 10 sieht, denn es stören noch die Rücksprungadresse und die Parameter ‚x' und ‚y'. Diese werden wie folgt eliminiert: Man speichert sich die Rücksprungadresse im Register A0, erhöht dann den Stackpointer um 8 Bytes, so daß auch die Parameter „gelöscht" werden. Anschließend kehrt das Programm zu der Rücksprungadresse zurück, die im Register A0 gespeichert ist.

Das war nun doch ein ganz schönes Stück Arbeit! Das Assemblerprogramm sieht dafür aber auch nicht schlecht aus. Mit ein bißchen Übung kann man es fast genauso leicht lesen wie das Pascal-Programm. Genauso geht es einem mit der Programmierung solcher Prozeduren. Die Methode der Parameterübergabe ist immer dieselbe, so daß man sich nach ein paar Beispielen schnell daran gewöhnt hat.

Wir haben nun also eine durch und durch rekursive Prozedur in einer so „niedrigen" Sprache wie Assembler programmiert. Ob die Routine auch wirklich so läuft, wie wir uns das vorgestellt haben? Im **Listing 7** ist ein kleines Testprogramm angegeben. Die Details dieses Programms seien dem Leser überlassen. Das Unterprogramm „Ausgabe", welches die Rechenergebnisse in lesbarer Form ausgibt, wird hier nicht weiter aufgeführt, da es zu sehr auf die Eigenheiten des Macintosh eingeht.
Die Ergebnisse einiger Funktionsaufrufe in der **Abb. 11** zeigen, daß auch diese komplizierte

```
Ø3B8 - 3F3C ØØØ8          MOVE.W  #8,-(A7)      ; Dauer für kurzen Ton (ok)

                     TestBeep

Ø3BC - A9C8               _SysBeep              ; End-Ton ausgeben
Ø3BE - 4CDF ØC Ø3         MOVEM.L (A7)+,DØ-D1/A2-A3 ; Register wiederholen
Ø3C2 - 4E75               RTS                   ; Rücksprung aus TestX
```

Listing 5: Die Ackermann-Funktion in Pascal

```
function Ack (x, y : integer) : integer;

{Eingabe: zwei ganze Zahlen x und y}
{Aufgabe: Die Funktion berechnet die Ackermann-Funktion A (x,y).}
{Ausgabe: Das Funktionsergebnis A (x,y) wird zurückgegeben.}

var xMinl : integer;
    yMinl : integer;

begin {Ack}
  if x=Ø then Ack := y+1
  else begin
    xMinl := x-1;
    if y=Ø
    then Ack := Ack (xMinl, 1)
    else begin
      yMinl := y-1;
      Ack   := Ack (xMinl, Ack (x, yMinl));
    end; {y>Ø}
  end; {x>Ø}
end; {Ack}
```

Listing 6: Die Ackermann-Funktion in Assembler

```
; Dieses Unterprogramm berechnet die Ackermann-Funktion A(x,y). Die
; beiden Parameter x und y müssen auf dem Stack übergeben werden. Vor den
; Parametern muß außerdem ein Long Platz für das Ergebnis gelassen werden.

                     Ackermann                ; Ackermann-Funktion

Ø42A - 4E56 FFF8           LINK    A6,#-8      ; Platz für 2 Longs machen

                     x       EQU   $C(A6)     ; Adr.v. Parameter x
                     y       EQU   8(A6)      ; Adr.v. Parameter y
                     xMinl   EQU   -4(A6)     ; Adr.d. Variablen xMinl
                     yMinl   EQU   -8(A6)     ; Adr.d. Variablen yMinl
                     Result  EQU   $1Ø(A6)    ; Adr.v. Ergebnis

Ø42E - 4AAE ØØØC           TST.L   x           ; Vgl. Parameter x mit Ø
Ø432 - 6EØØ xxxx           BGT     xGTØ        ; Wenn x>Ø, Sprung
Ø436 - 2D6E ØØØ8 ØØ1Ø      MOVE.L  y,Result    ; Ergebnis:= y
Ø43C - 52AE ØØ1Ø           ADDQ.L  #1,Result   ; Ergebnis:= y+1
Ø44Ø - 4ECØ xxxx           BRA     AckEnd      ; Sprung ans Ende

                     xGTØ

Ø444 - 2D6E ØØØC FFFC      MOVE.L  x,xMinl     ; xMinl:= x
Ø44A - 53AE FFFC           SUBQ.L  #1,xMinl    ; xMinl:= x-1
Ø44E - 4AAE ØØØ8           TST.L   y           ; Vgl. Parameter y mit Ø
Ø452 - 6EØØ xxxx           BGT     yGTØ        ; Wenn y>Ø, Sprung

Ø456 - 42A7                CLR.L   -(A7)       ; Platz für Ergebnis machen
Ø458 - 2F2E FFFC           MOVE.L  xMinl,-(A7) ; 1.Parameter:= xMinl
Ø45C - 2F3C ØØØØ ØØØ1      MOVE.L  #1,-(A7)    ; 2.Parameter:= 1
Ø462 - 61C6                BSR     Ackermann   ; Berechne A(xMinl,1)
Ø464 - 2D5F ØØ1Ø           MOVE.L  (A7)+,Result ; Ergebnis:= A(xMinl,1)
Ø468 - 4ECØ xxxx           BRA     AckEnd      ; Sprung ans Ende

                     yGTØ

Ø46C - 2D6E ØØØ8 FFF8      MOVE.L  y,yMinl     ; yMinl:= y
Ø472 - 53AE FFF8           SUBQ.L  #1,yMinl    ; yMinl:= y-1

Ø476 - 42A7                CLR.L   -(A7)       ; Platz für Ergebnis (außen)
Ø478 - 2F2E FFFC           MOVE.L  xMinl,-(A7) ; 1.Par.(außen):= xMinl
Ø47C - 42A7                CLR.L   -(A7)       ; Platz für Ergebnis (innen)
Ø47E - 2F2E ØØØC           MOVE.L  x,-(A7)     ; 1.Par.(innen):= x
Ø482 - 2F2E FFF8           MOVE.L  yMinl,-(A7) ; 2.Par.(innen):= yMinl
Ø486 - 61A2                BSR     Ackermann   ; 2.Par.(außen):= A(x,yMinl)
Ø488 - 61AØ                BSR     Ackermann   ; A(xMinl), A(x,yMinl)
Ø48A - 2D5F ØØ1Ø           MOVE.L  (A7)+,Result ; Ergebnis:= " " " "

                     AckEnd

Ø48E - 4E5E                UNLK    A6          ; lokale Variablen löschen
Ø49Ø - 2Ø5F                MOVEA.L (A7)+,AØ    ; AØ:= Rücksprungadresse
Ø492 - 5Ø8F                ADDQ.L  #8,A7       ; 8 Bytes Parameter löschen
Ø494 - 4EDØ                JMP     (AØ)        ; Rücksprung aus Ackermann
```

Funktion korrekt arbeitet. Das vermittelt doch einen Einblick in die Leistungsfähigkeit des 16-Bit-Mikroprozessors MC 68000. In diesem Artikel konnten weitere interessante Eigenschaften dieses Prozessors leider nicht weiter erwähnt werden, da der Platz nicht zur Verfügung stand. Ich hoffe, daß ich Sie dennoch ein wenig für diesen Prozessor interessieren konnte.

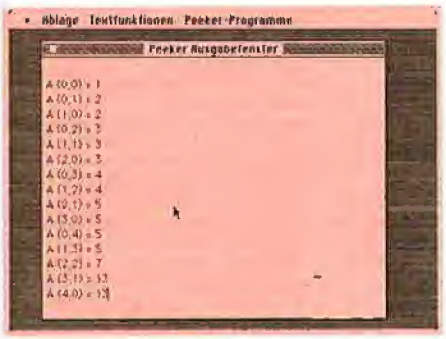


Abb. 11

Zum Abschluß noch eine Bemerkung für die ganz kritischen Leser: Am Ende unseres Unterprogramms haben wir die Rücksprungadresse vom Stack in das Register A0 geladen, ohne A0 zu retten. Das heißt, das Unterprogramm ändert den Inhalt von A0! Dies widerspricht zwar der Philosophie, alle Register unverändert zu belassen, stellt aber einen recht guten Kompromiß dar, da auf diese Art das Ende der Unterprogramme genauso leicht zu behandeln ist wie der Anfang der Unterprogramme.

Listing 7: Test der Ackermann-Funktion

```
; Mit diesem Unterprogramm kann die Ackermann-Funktion getestet werden.
; Diese Funktion wird mit allen 15 Parameter-Paaren aufgerufen, deren
; Summe nicht größer als 4 ist, also z.B. mit (1,Ø), (4,Ø), (2,2), aber
; nicht mit (3,2). Die Ergebnisse werden mit dem Unterprogramm "Ausgabe"
; in lesbarer Form auf dem Bildschirm ausgegeben.

                        AckTest                     ; Test der Ackermann-Fkt.

Ø496 - 48E7 1EØØ            MOVEM.L D3-D6,-(A7)     ; Register retten
Ø49A - 76ØØ                 MOVEQ   #Ø,D3           ; Parametersumme (D3):= Ø

                        NextD3

Ø49C - 78ØØ                 MOVEQ   #Ø,D4           ; Parameter x (D4):= Ø
Ø49E - 2AØ3                 MOVE.L  D3,D5           ; Parameter y (D5):= D3

                        NextCall

Ø4AØ - 42A7                 CLR.L   -(A7)           ; Platz f. Ergebnis machen
Ø4A2 - 2FØ4                 MOVE.L  D4,-(A7)        ; Parameter x auf Stack
Ø4A4 - 2FØ5                 MOVE.L  D5,-(A7)        ; Parameter y auf Stack
Ø4A6 - 6182                 BSR     Ackermann       ; Berechne A (x,y)
Ø4A8 - 2C1F                 MOVE.L  (A7)+,D6        ; D6:= A (x,y)

Ø4AA - 61ØØ FF18            BSR     Ausgabe         ; Ergebnis ausgeben

Ø4AE - 5284                 ADDQ.L  #1,D4           ; Parameter x erhöhen
Ø4BØ - 5385                 SUBQ.L  #1,D5           ; Parameter y erniedrigen
Ø4B2 - 6CEC                 BGE     NextCall        ; Wenn y>=Ø, wiederholen

Ø4B4 - 5283                 ADDQ.L  #1,D3           ; Parametersumme erhöhen
Ø4B6 - ØC83 ØØØØ ØØØ4       CMPI.L  #4,D3           ; Vgl. Parametersumme mit 4
Ø4BC - 6FDE                 BLE     NextD3          ; Wenn Param.sum.<=4, wdh.

Ø4BE - 4CDF Ø078            MOVEM.L (A7)+,D3-D6     ; Register wiederholen
Ø4C2 - 4E75                 RTS                     ; Rücksprung aus AckTest
```

Tabelle 1: Übersicht über die Adressierungsarten des 68000

Bezeichnung	Code M	Code R	Schreibweise	Beispiel	Ergebnis	Effektive Adresse
Datenregister direkt	ØØØ	n	Dn	MOVE.W DØ,D3	D3 = FEDCFFFA	Register DØ
Adressregister direkt	ØØ1	n	An	MOVE.W A6,D3	D3 = FEDCA7ØØ	Register A6
Adressregister indirekt	ØØ2	n	(An)	MOVE.W (A6),D3	D3 = FEDC1122	Ø7A7ØØ
Adressregister indirekt mit Postinkrement	ØØ3	n	(An)+	MOVE.W (A6)+,D3	D3 = FEDC1122 A6 = ØØØ7A7Ø2	Ø7A7ØØ
Adressregister indirekt mit Predekrement	ØØ4	n	-(An)	MOVE.W -(A6),D3	A6 = ØØØ7A6FE D3 = FEDCFEFF	Ø7A7ØØ-2 = Ø7A6FE
Adressregister indirekt mit Verschiebung	ØØ5	n	d(An)	MOVE.W 4(A6),D3	D3 = FEDC5566	Ø7A7ØØ+4 = Ø7A7Ø4
Adressregister indirekt mit Index	ØØ6	n	d(An,Rx)	MOVE.W 2(A6,DØ.W),D3	D3 = FEDCFCFD	Ø7A7ØØ+(-6)+2 = Ø7A6FC
Absolut kurze Adresse	ØØ7	ØØØ	xxx	MOVE.W $4ØØØ,D3	D3 = FEDCØ1Ø2	ØØ4ØØØ
Absolute Doppelwort-Adresse	ØØ7	ØØ1	xxxxx	MOVE.W $Ø7A6F8,D3	D3 = FEDCF8F9	Ø7A6F8
Programmzaehler mit Verschiebung (relativ)	ØØ7	ØØ2	d(PC)	MOVE.W $67Ø2(PC),D3	D3 = FEDC3344	Ø74ØØØ+67Ø2 = Ø7A7Ø2
Programmzaehler mit Index	ØØ7	ØØ3	d(PC,Rx)	MOVE.W $A(PC,DØ.L),D3	D3 = FEDCØ5Ø6	Ø74ØØØ+(-Ø7ØØØ6) +A = ØØ4ØØ4
Direkte Daten Index	ØØ7	ØØ4	#xxx	MOVE.W #$AFFE,D3	D3 = FEDCAFFE	Erweiterungswort AFFE
Statusregister	ØØ7	ØØ4	SR/CCR	ORI.B #%ØØ11ØØØØ,CCR	CCR=%ØØ11ØØØØ	untere Haelfte von SR (= CCR)

Als Ausgangspunkt sei folgende Situation angenommen:

Register :

```
DØ = FFF9FFFA (= -Ø7ØØØ6)      PC  = ØØØ74ØØØ
D3 = FEDCBA98                  CCR = %ØØØØØØØØ
A6 = ØØØ7A7ØØ                  (alle Flags auf Ø)
```

Speicher :

```
ab Adresse ØØ4ØØØ : Ø1 Ø2 Ø3 Ø4 Ø5 Ø6 Ø7 Ø8
ab Adresse Ø7A6F8 : F8 F9 FA FB FC FD FE FF
ab Adresse Ø7A7ØØ : 11 22 33 44 55 66 77 88
```

Tabelle 2: Befehle des 68000

Mnem.	Beschreibung Englisch	Deutsch
ABCD	Add Decimal with Extend	Addiere dezimal mit X-Flag
ADD	Add	Addiere binaer
AND	Logical And	Logisches UND
ASL	Arithmetic Shift Left	Arithmetisches Schieben Links
ASR	Arithmetic Shift Right	Arithmetisches Schieben Rechts
Bcc	Branch Conditionally	Bedingte Verzweigung
BCHG	Bit Test and Change	Bit pruefen und aendern
BCLR	Bit Test and Clear	Bit pruefen und auf Ø setzen
BRA	Branch Always	Unbedingte Verzweigung
BSET	Bit Test and Set	Bit pruefen und auf 1 setzen
BTST	Bit Test	Bit pruefen
CHK	Check Register Against Bounds	Pruefe Register auf Grenzen
CLR	Clear Operand	Operanden auf Ø setzen
CMP	Compare	Vergleiche
DBcc	Test Condition, Decrement and Branch	Bedingung pruefen, Register erniedrigen und Verzweigung
DIVS	Signed Divide	Division mit Vorzeichen
DIVU	Unsigned Devide	Division ohne Vorzeichen
EOR	Logical Exclusive Or	Logisches Exklusiv-ODER
EXG	Exchange Registers	Registerinhalte vertauschen
EXT	Sign Extend	Vorzeichenerweiterung
JMP	Jump	Sprung
JSR	Jump to Subroutine	Unterprogrammaufruf
LEA	Load Effective Address	Lade die Effektive Adresse
LINK	Link and Allocate	Platz auf Stack reservieren
LSL	Logical Shift Left	Logisches Schieben Links
LSR	Logical Shift Right	Logisches Schieben Rechts
MOVE	Move Data	Daten transportieren
MOVEM	Move Multiple Registers	Transportiere mehrere Register
MOVEP	Move Peripheral Data	Transport. Daten d. Peripherie
MULS	Signed Multiply	Multiplikation mit Vorzeichen
MULU	Unsigned Multiply	Multiplikation ohne Vorzeichen
NBCD	Negate Decimal with Extend	Negiere dezimal mit X-Flag
NEG	Negate	Negiere binaer
NOP	No Operation	Keine Operation
NOT	One's Complement	1-er Komplement
OR	Logical Or	Logisches ODER
PEA	Push Effective Address	Effektive Adresse auf Stack
RESET	Reset External Devices	Normieren externer Einheiten
ROL	Rotate Left without Extend	Rotieren Links ohne X-Flag
ROR	Rotate Right without Extend	Rotieren Rechts ohne X-Flag
ROXL	Rotate Left with Extend	Rotieren Links mit X-Flag
ROXR	Rotate Right with Extend	Rotieren Rechts mit X-Flag
RTE	Return from Exception	Ruecksprung v. Ausnahmeroutine
RTR	Return and Restore	Ruecksprg. u. Flags rueckladen
RTS	Return from Subroutine	Ruecksprung vom Unterprogramm
SBCD	Subtract Decimal with Extend	Subtrahiere dezimal mit X-Flag
Scc	Set Conditional	Bedingtes Setzen des Operanden
STOP	Stop	Statusreg. laden und anhalten
SUB	Subtract	Subtraktion
SWAP	Swap Data Register Halves	Vertausche Registerhaelften
TAS	Test and Set	Pruefe und setze Operanden
TRAP	Trap	Unterbrechung ("Falle")
TRAPV	Trap on Overflow	Unterbrechung bei Ueberlauf
TST	Test	Pruefe Operanden
UNLK	Unlink	Platz auf Stack freigeben

Tabelle 3: Befehlsvariationen beim 68000

Befehlstyp und	Variation	Beschreibung Englisch	Deutsch
ADD	ADD	Add	Addition (binaer)
	ADDA	Add Address	Addiere Adresse
	ADDQ	Add Quick	Addiere schnell (und direkt)
	ADDI	Add Immediate	Addiere direkte Daten
	ADDX	Add with Extend	Addiere mit X-Flag
AND	AND	Logical AND	Logisches UND
	ANDI	Logical AND Immediate	Logisches UND, direkte Daten
CMP	CMP	Compare	Vergleich
	CMPA	Compare Address	Vergleiche Adresse
	CMPM	Compare Memory	Vergleiche im Speicher
	CMPI	Compare Immediate	Vergleich mit direkten Daten
EOR	EOR	Logical Exclusive OR	Logisches Exklusiv-ODER
	EORI	Exclusive-OR Immediate	Logisches Ex.-ODER, dir. Daten
MOVE	MOVE	Move Data	Transportiere Daten
	MOVEA	Move Address	Transportiere Adresse
	MOVEQ	Move Quick	Transportiere schnell & direkt
	MOVE from SR	Move from Statusreg.	Transport. vom Statusregister
	MOVE to SR	Move to Statusregister	Transport. zum Statusregister
	MOVE to CCR	Move to Conditioncodes	Transportiere zu den Flags
	MOVE USP	Move User Stackpointer	Transp. Anwender-Stapelzeiger
NEG	NEG	Negate	Negation (binaer)
	NEGX	Negate with Extend	Negiere mit X-Flag
OR	OR	Logical OR	Logisches ODER
	ORI	Logical OR Immediate	Logisches ODER, direkte Daten
SUB	SUB	Subtract	Subtraktion (binaer)
	SUBA	Subtract Address	Subtrahiere Adresse
	SUBQ	Subtract Quick	Subtrahiere schnell & direkt
	SUBI	Subtract Immediate	Subtrahiere direkte Daten
	SUBX	Subtract with Extend	Subtrahiere mit X-Flag

Kurzrezension Assembler-Bücher

In der Peeker-Redaktion sind drei Assembler-Bücher eingegangen, die hier kurz besprochen werden sollen:

(1) T. King und B. Knight: **Programmierung des M68000,** 1986 (Verlagsort fehlt im Copyright-Vermerk und auf dem Umschlag), 201 S., kart.
Bemerkungen: Leichtverständliche, kurzgefaßte, aber unvollständige Einführung. Als „Schnupperkurs" für Anhänger höherer Programmiersprachen geeignet. Skurrile Besonderheiten: Hex-Bytes existieren für die Autoren nicht. Was ein Opcode ist, wird nicht einmal exemplarisch gezeigt. Eine bildliche Darstellung der Daten- und Adreßregister wird man ebenfalls vergeblich suchen.

(2) C. Vieillefond: **Programmierung des 68000,** Düsseldorf 1985, 453 S., kart.
Bemerkungen: Sehr umfangreiche und vorzüglich übersetzte Gesamtdarstellung unter besonderer Berücksichtigung hardwaretechnischer Belange (Peripheriebausteine, Signale, Datenbus usw.). Die Programmierung selbst kommt dabei allerdings etwas zu kurz, so daß der Buchtitel insoweit irreführend ist. Daher mehr für Hardware-Leute empfehlenswert.

(3) G. Kane, D. Hawkins und L. Leventhal: **68000 Assembly Language Programming,** Berkeley 1981, ca. 500 S., kart.
Bemerkungen: Obwohl bereits 1981 erstmals erschienen, ist dieses Buch wahrscheinlich die beste Einführung für Programmierer, denn es wird nicht nur jeder Befehl anhand vorzüglicher Diagramme erklärt, sondern auch durch kleine Beispielprogramme eingeübt (Abschnitte: Anfängerprogramme; Einfache Schleifen; Codierung und Konvertierung, Arithmetische Programme, Listenverarbeitung usw.). *us*

Tabelle 4: Opcodes und Taktzyklen aller 68000-Befehle

Teil 1: Befehle mit variabler „Effektiver Adresse“

Assembler-Syntax & Operations-beschreibung	Flags XNZVC	s	Effektive Adresse (ea): Dn Code :Cy	An Code :Cy	(An) Code :Cy	(An)+ Code :Cy	-(An) Code :Cy	d(An) Code :Cy	d(An,Rx) Code :Cy	xx.W Code :Cy	xxxx.L Code :Cy	d(PC) Code :Cy	d(PC,Rx) Code :Cy	#xxx Code :Cy	CCR/SR Code :Cy
ADD.s ea,Dm	*****	B	15m00n: 4	---	15m02n: 8	15m03n: 8	15m04n:10	15m05n:12	15m06n:14	15m070:12	15m071:16	15m072:12	15m073:14	15m074: 8	---
(Dm)+(ea)->Dm		W	15m10n: 4	15m11n: 4	15m12n: 8	15m13n: 8	15m14n:10	15m15n:12	15m16n:14	15m170:12	15m171:16	15m172:12	15m173:14	15m174: 8	---
		L	15m20n: 8	15m21n: 8	15m22n:14	15m23n:14	15m24n:16	15m25n:18	15m26n:20	15m270:18	15m271:22	15m272:18	15m273:20	15m274:14	---
ADD.s Dm,ea	*****	B	---	---	15m42n:12	15m43n:12	15m44n:14	15m45n:16	15m46n:18	15m470:16	15m471:20	---	---	---	---
(ea)+(Dm)->ea		W	---	---	15m52n:12	15m53n:12	15m54n:14	15m55n:16	15m56n:18	15m570:16	15m571:20	---	---	---	---
		L	---	---	15m62n:20	15m63n:20	15m64n:22	15m65n:24	15m66n:26	15m670:24	15m671:28	---	---	---	---
ADDA.s ea,Am	------	B	---	---	---	---	---	---	---	---	---	---	---	---	---
(Am)+(ea)->Am		W	15m30n: 8	15m31n: 8	15m32n:12	15m33n:12	15m34n:14	15m35n:16	15m36n:18	15m370:16	15m371:20	15m372:16	15m373:18	15m374:12	---
		L	15m70n: 8	15m71n: 8	15m72n:14	15m73n:14	15m74n:16	15m75n:18	15m76n:20	15m770:18	15m771:22	15m772:18	15m773:20	15m774:14	---
ADDI.s #xxx,ea	*****	B	00300n: 8	---	00302n:16	00303n:16	00304n:18	00305n:20	00306n:22	00307n:20	00307:24	---	---	---	---
(ea)+xxx->ea		W	00310n: 8	---	00312n:16	00313n:16	00314n:18	00315n:20	00316n:22	00317n:20	00317:24	---	---	---	---
		L	00320n:16	---	00322n:28	00323n:28	00324n:30	00325n:32	00326n:34	00327n:32	00327:36	---	---	---	---
ADDQ.s #x,ea	*****	B	05x00n: 4	---	05x02n:12	05x03n:12	05x04n:14	05x05n:16	05x06n:18	05x070:16	05x071:20	---	---	---	---
(ea)+x->ea		W	05x10n: 4	05x11n: 8	05x12n:12	05x13n:12	05x14n:14	05x15n:16	05x16n:18	05x170:16	05x171:20	---	---	---	---
		L	05x20n: 8	05x21n: 8	05x22n:20	05x23n:20	05x24n:22	05x25n:24	05x26n:26	05x270:24	05x271:28	---	---	---	---
AND.s ea,Dm	-**00	B	14m00n: 4	---	14m02n: 8	14m03n: 8	14m04n:10	14m05n:12	14m06n:14	14m070:12	14m071:16	14m072:12	14m073:14	14m074: 8	---
(Dm)&(ea)->Dm		W	14m10n: 4	---	14m12n: 8	14m13n: 8	14m14n:10	14m15n:12	14m16n:14	14m170:12	14m171:16	14m172:12	14m173:14	14m174: 8	---
		L	14m20n: 8	---	14m22n:14	14m23n:14	14m24n:16	14m25n:18	14m26n:20	14m270:18	14m271:22	14m272:18	14m273:20	14m274:14	---
AND.s Dm,ea	-**00	B	---	---	14m42n:12	14m43n:12	14m44n:14	14m45n:16	14m46n:18	14m470:16	14m471:20	---	---	---	---
(ea)&(Dm)->ea		W	---	---	14m52n:12	14m53n:12	14m54n:14	14m55n:16	14m56n:18	14m570:16	14m571:20	---	---	---	---
		L	---	---	14m62n:20	14m63n:20	14m64n:22	14m65n:24	14m66n:26	14m670:24	14m671:28	---	---	---	---
ANDI.s #xxx,ea	-**00	B	00100n: 8	---	00102n:16	00103n:16	00104n:18	00105n:20	00106n:22	00107n:20	00107:24	---	---	---	00107:20
(ea)&xxx->ea		W	00110n: 8	---	00112n:16	00113n:16	00114n:18	00115n:20	00116n:22	00117n:20	00117:24	---	---	---	00117:20
		L	00120n:16	---	00122n:28	00123n:28	00124n:30	00125n:32	00126n:34	00127n:32	00127:36	---	---	---	---
ASL ea	*****	B	---	---	---	---	---	---	---	---	---	---	---	---	---
X/C<=(ea)<=0		W	---	---	16072n:16	16073n:16	16074n:18	16075n:20	16076n:22	16077n:20	16077:24	---	---	---	---
um 1 Bit		L	---	---	---	---	---	---	---	---	---	---	---	---	---
ASR ea	*****	B	---	---	---	---	---	---	---	---	---	---	---	---	---
(ea)[F]=>(ea)=>X/C		W	---	---	16032n:16	16033n:16	16034n:18	16035n:20	16036n:22	16037n:20	16037:24	---	---	---	---
um 1 Bit		L	---	---	---	---	---	---	---	---	---	---	---	---	---
BCHG Dm,ea	--*--	B	---	---	00m52n:12	00m53n:12	00m54n:14	00m55n:16	00m56n:18	00m570:16	00m571:20	---	---	---	---
-(ea)[(Dm)]->Z		W	---	---	---	---	---	---	---	---	---	---	---	---	---
-(ea)[(Dm)]->ea[(Dm)]		L	00m50n: 8	---	---	---	---	---	---	---	---	---	---	---	---
BCHG #x,ea	--*--	B	---	---	00412n:16	00413n:16	00414n:18	00415n:20	00416n:22	00417n:20	00417:24	---	---	---	---
-(ea)[x]->Z/ea[x]		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	00410n:12	---	---	---	---	---	---	---	---	---	---	---	---
BCLR Dm,ea	--*--	B	---	---	00m52n:12	00m53n:12	00m54n:14	00m55n:16	00m56n:18	00m570:16	00m571:20	---	---	---	---
-(ea)[(Dm)]->Z		W	---	---	---	---	---	---	---	---	---	---	---	---	---
0->ea[(Dm)]		L	00m60n:10	---	---	---	---	---	---	---	---	---	---	---	---
BCLR #x,ea	--*--	B	---	---	00422n:16	00423n:16	00424n:18	00425n:20	00426n:22	00427n:20	00427:24	---	---	---	---
-(ea)[x]->Z,0->ea[x]		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	00420n:14	---	---	---	---	---	---	---	---	---	---	---	---
BSET Dm,ea	--*--	B	---	---	00m72n:12	00m73n:12	00m74n:14	00m75n:16	00m76n:18	00m770:16	00m771:20	---	---	---	---
-(ea)[(Dm)]->Z		W	---	---	---	---	---	---	---	---	---	---	---	---	---
1->ea[(Dm)]		L	00m70n: 8	---	---	---	---	---	---	---	---	---	---	---	---
BSET #x,ea	--*--	B	---	---	00432n:16	00433n:16	00434n:18	00435n:20	00436n:22	00437n:20	00437:24	---	---	---	---
-(ea)[x]->Z,1->ea[x]		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	00430n:12	---	---	---	---	---	---	---	---	---	---	---	---
BTST Dm,ea	--*--	B	---	---	00m42n: 8	00m43n: 8	00m44n:10	00m45n:12	00m46n:14	00m470:12	00m471:16	00m472:12	00m473:14	00m474: 8	---
-(ea)[(Dm)]->Z		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	00m40n: 6	---	---	---	---	---	---	---	---	---	---	---	---
BTST #x,ea	--*--	B	---	---	00402n:12	00403n:12	00404n:14	00405n:16	00406n:18	00407n:16	00407:20	00407:16	00407:18	---	---
-(ea)[x]->Z		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	00400n:10	---	---	---	---	---	---	---	---	---	---	---	---
CHK ea,Dm	-*UUU	B	---	---	---	---	---	---	---	---	---	---	---	---	---
Wenn (Dm)<0 oder		W	04m06n:40	---	04m62n:44	04m63n:44	04m64n:46	04m65n:48	04m66n:50	04m670:48	04m671:52	04m672:48	04m673:50	04m673:44	---
(Dm)>(ea), dann Trap		L	--- 8	---	--- 12	--- 12	--- 14	--- 16	--- 18	--- 16	--- 20	--- 16	--- 18	--- 12	---
CLR.s ea	-0100	B	04100n: 4	---	04102n:12	04103n:12	04104n:14	04105n:16	04106n:18	04107n:16	04107:20	---	---	---	---
0->ea		W	04110n: 4	---	04112n:12	04113n:12	04114n:14	04115n:16	04116n:18	04117n:16	04117:20	---	---	---	---
		L	04120n: 6	---	04122n:20	04123n:20	04124n:22	04125n:24	04126n:26	04127n:24	04127:28	---	---	---	---
CMP.s ea,Dm	-****	B	13m00n: 4	---	13m02n: 8	13m03n: 8	13m04n:10	13m05n:12	13m06n:14	13m070:12	13m071:16	13m072:12	13m073:14	13m074: 8	---
Test von (Dm)-(ea)		W	13m10n: 4	13m11n: 4	13m12n: 8	13m13n: 8	13m14n:10	13m15n:12	13m16n:14	13m170:12	13m171:16	13m172:12	13m173:14	13m174: 8	---
		L	13m20n: 6	13m21n: 6	13m22n:14	13m23n:14	13m24n:16	13m25n:18	13m26n:20	13m270:18	13m271:22	13m272:18	13m273:20	13m274:14	---
CMPA.s ea,Am	-****	B	---	---	---	---	---	---	---	---	---	---	---	---	---
Test von (Am)-(ea)		W	13m30n: 6	13m31n: 6	13m32n:10	13m33n:10	13m34n:12	13m35n:14	13m36n:16	13m370:14	13m371:18	13m372:14	13m373:16	13m374:10	---
		L	13m70n: 6	13m71n: 6	13m72n:14	13m73n:14	13m74n:16	13m75n:18	13m76n:20	13m770:18	13m771:22	13m772:18	13m773:20	13m774:14	---
CMPI.s #xxx,ea	-****	B	00600n: 8	---	00602n:12	00603n:12	00604n:14	00605n:16	00606n:18	00607n:16	00607:20	---	---	---	---
Test von (ea)-xxx		W	00610n: 8	---	00612n:12	00613n:12	00614n:14	00615n:16	00616n:18	00617n:16	00617:20	---	---	---	---
		L	00620n:14	---	00622n:20	00623n:20	00624n:22	00625n:24	00626n:26	00627n:24	00627:28	---	---	---	---
DIVS ea,Dm	-***0	B	---	---	---	---	---	---	---	---	---	---	---	---	---
(Dm)/(ea)->Dm		W	10m70n:58	---	10m72n:52	10m73n:52	10m74n:54	10m75n:56	10m76n:58	10m770:56	10m771:70	10m772:56	10m773:58	10m773:52	---
		L	---	---	---	---	---	---	---	---	---	---	---	---	---
DIVU ea,Dm	-***0	B	---	---	---	---	---	---	---	---	---	---	---	---	---
(Dm)/(ea)->Dm		W	10m30n:40	---	10m32n:44	10m33n:44	10m34n:46	10m35n:48	10m36n:50	10m370:48	10m371:52	10m372:48	10m373:50	10m373:44	---
		L	---	---	---	---	---	---	---	---	---	---	---	---	---
EOR.s Dm,ea	-**00	B	13m40n: 4	---	13m42n:12	13m43n:12	13m44n:14	13m45n:16	13m46n:18	13m470:16	13m471:20	---	---	---	---
(ea) EOR (Dm)->ea		W	13m50n: 4	---	13m52n:12	13m53n:12	13m54n:14	13m55n:16	13m56n:18	13m570:16	13m571:20	---	---	---	---
		L	13m60n: 8	---	13m62n:20	13m63n:20	13m64n:22	13m65n:24	13m66n:26	13m670:24	13m671:28	---	---	---	---
EORI.s #xxx,ea	-**00	B	00500n: 8	---	00502n:16	00503n:16	00504n:18	00505n:20	00506n:22	00507n:20	00507:24	---	---	---	00507:20
(ea) EOR xxx->ea		W	00510n: 8	---	00512n:16	00513n:16	00514n:18	00515n:20	00516n:22	00517n:20	00517:24	---	---	---	00517:20
		L	00520n:16	---	00522n:28	00523n:28	00524n:30	00525n:32	00526n:34	00527n:32	00527:36	---	---	---	---
JMP ea	------		---	---	04732n: 8	---	---	04735n:10	04736n:14	04737n:10	04737:12	04737:10	04737:14	---	---
ea->PC															
JSR ea	------		---	---	04722n:16	---	---	04725n:18	04726n:22	04727n:18	04727:20	04727:18	04727:22	---	---
(PC)->-(SP),ea->PC															
LEA ea,Am	------	B	---	---	---	---	---	---	---	---	---	---	---	---	---
ea->Am		W	---	---	---	---	---	---	---	---	---	---	---	---	---
		L	---	---	04m72n: 4	---	---	04m75n: 8	04m76n:12	04m770: 8	04m771:12	04m772: 8	04m773:12	---	---

PEEKER Börse

Verkauf Hardware

Macintosh 128 auf 512KB DM 299,–. Fa. Schlösser, Tel. ab 17.00 Uhr, 089 - 98 58 89

Komplett: II+ komp., 2 Laufwerke abges. dt. Preh Tastatur, 64K, Grafikprinter u. Seriellinterf., Z80, 80 Z, im Metallgehäuse, Monitor, viel Dokumentation, DM 2000,– ab 18 h, T. (0 89) 3 20 11 00

IIe, 80Z + 64K, Matrixdr. 80Z/Sek. grafikfäh., Juki 6100 Typenrad 20 Z/Sek. + 2. Disclaufw./Monitor Textverarb. progr. VB DM 2500,–. Tel. 0 70 21 / 5 94 62.

II+ Comp. 64K+Z80+16K AKKU-RAM+ Super serielle + Centronix + Eprom-Burner + Laufwerk / Controller Grün Monitor; Olivetti-Drucker; 100 Disketten + Literatur – komplett DM 1700,– oder einz. T. 0 45 31 / 8 47 18

AP-35 Grafikkarte mit Handb. + Software DM 450,–. Tel. 02 08 / 40 79 06

IIe-komp., 80 Z + 64K, Z80, 640K-LW, Erphi-Contr. Monitor, zus. DM 1500,– II + 80Z DM 100,– 16K. Tel. 089 / 850 78 34.

Apple IIe (128K), Z80, 80Z, Lüfter, 2 × Laufwerke, Par. Interface, Lit. + Software, VB DM 2300,–. Tel. 0 23 25 / 4 10 69.

Apple II + kompat, 2×TEAC Slim!. Disk Shugart/Ehringcontroller, 80Z/Softswitch, Z80, abgesetzte Preh-Kommander/Zahlenblock, bernstein Monitor, Grappler Card, Cosmos 80 Drucker, div. SW, DM 2500,– VHS.
Tel. 0 43 46 / 66 47.

Z-80 Microsoft-Premiumcard incl. 80Z für IIe DM 700,–. 6522 PIO-Card DM 110,–. Super-Serial-Card DM 160,–. Orig. Apple Controller DM 100,–.
Tel. 0 71 61 / 7 91 17 ab 18.00 Uhr.

Apple IIc, Monitor, Stand, Joyst., RefMan. Lit., Software, Garantie, VB 2000,–. Imagewriter II + Garantie DM 1800,–.
Mahr (07 11) 319 42 55.

Apple II+ comp 64K 80Z + Z80A-Karte TEAC-L 35-40-80 TR 640KB Erphicon. IBM Geh. + Tastatur Zenith-Monitor – bernstein VHB 1600,–.
Telefon 0 52 01 / 31 29

Netzteil 12V/2, 5A für Apple II, VB DM 150,–.
Tel. 0 22 37 / 6 17 64.

UNIDISK 3,5 mit UniDOS 3.3 (neu), Z-RAM (512 K) für IIc (neu) jeweils DM 1200,–.
Tel. 0 71 56 / 2 74 51.

Apple-Profile 5MB/1985 neuwertig mit Controller VB: DM 2900,– Loth. Bauer, T. 0 65 25 / 491.

Apple IIe orig. geg. Gebot zu verkaufen. Tel. 02 12 / 5 72 35.

Orig. US-Apple IIe, inkl. Trafo. Monitor, 2 Disk II, 80Z, Par.-Karte, viel Lit., sehr viel Softw. VB DM 2700,–. Tel. 06 51 / 1 79 25.

Softcard IIe (64K, CP/M, 80Z) DM 600,– Apple IIe Extended 80 Zeichen Karte DM 250,–.
A. Grün, Tel. 0 21 51 / 30 21 77.

Apple II m. Monitor v. Floppy, 10MB Profile, Apple-DMP-Matrix-Drucker 8 Mhz Z80 (AP22). Kompl. DM 4500,– auch einzel zu verk. Tel. ab 19 h (0 25 07) 14 90.

Orig. DTACK-512K wie Peeker 6/86.
M. Ries, Salvianstr. 2, Trier.

AP 2 (200,–), AP 13B (150,–) von IBS für Apple II.
Tel. 0 47 31 / 52 68.

Gibson Light PEN für Apple IIe, Preis VB. T. 07 11 / 456 03 20.

IIe komp. IBM-Geh. ext. Tastatur, 128K. Monit. Z80, Palk., Grappler+, orig. Apple-II-Disk u. Controller, DM 1500,–.
Tel. 05 31 / 6 33 66.

Original Apple-Produkte 12 % unter Listenpreis; Apple IIc + Monitor + Mo.-Stand DM 1810,– (incl. MwSt.) Liste bei Datent. Schoeben, Bahnhofstr. 12, 8900 Augsburg, Tel.: 08 21 / 15 23 77.

Apple II+ komp., 64KB, Monitor, 80Z., Z80, Operator II-Tast., div. Handb. u. SW, Drucker NEC P2, VB DM 4500,–.
Tel. 06 61 / 6 26 44 ab 18.00 Uhr.

Brother CE60 + Interface IF50 umschaltbar RS232C-Centronics anschlußfertig für IIc mit Kabel.
Klamser, Kreitzstr. 4, 6200 Wiesbaden (auch im Raum Goslar).

Apple komp. Computer, Peripherie. Gelsen-Electr. K-Schumacher-Str. 124, 4650 Gelsenkirchen, Tel. 02 09 / 8 30 33.

Lehrer verk. wegen Syst.w. in der Schule sein. priv. Apple IIe 128K + 80Zeich + Sup + Ser. Card + Duo Disk + IIe Mon. Imagewr. 12" + Mouse + alle Manuals Lit. für DM 400,– + AppleWorks dt/engl + ProDOS + Mous Paint + priv. Buchhalt. Neupr. DM 10800. 18 Mon. alt DM 6500,–.
Tel. 0 60 74 / 2 76 96.

IIe, 80Z + 64K, Z80, Super Serial, Disk-Contr., 2 Laufwerke, Monitor, VB DM 2500,–.
Tel. 0 61 06 / 2 35 83.

Apple II+ (orig.) in Metallgeh., 2 Laufw. (orig.), prof. Tastatur, Monitor, Einsteckkarten, viel Softw. + Literatur, DM 1 700,–.
Bauer, Tel. 0 71 21 / 4 05 18.

ITOH 8500, ITOGRAPH-IF, Manuals, NP: 1600,–, VB: 650,–.
Tel. (02 09) 2 30 23.

Verkaufe Apple IIc für DM 1350,–.
Tel. 069 / 35 96 00, ab 18.00 Uhr.

IBS Ramdisk 256 KB, mit Software und Dokumentation, Preis VS, Tel. 0 21 02 / 47 37 50.

Verkauf Software

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

BIORHYTHMUS mit Partnervergleich druckt Ihren natürlichen Lebensrhythmus als Sinuskurve/Biozahl. f. 1-365 Tage m. ausf. Beschreibung nur DM 67,85 für alle Apple II. Junker, A-Möllerstr. 1, 6390 Usingen.

Alle Lotto-Zahlen 6 aus 49 von 1955 – Ende 1985 für Apple II auf Diskette incl. interessantem Demo-Progr. für DM 30,– Nachnahme. Tel. 0 30 / 7 52 01 94

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.

StatWorks Statistikprogramm DM 425,–. Zu beziehen durch: Heyden & Son GmbH, D-4440 Rheine, Postfach 529, Tel. 0 59 71 / 5 51 11

Video Cassettenverwaltung Apple II+ und comp. DM 68,– voll menuegesteuert. Suchen nach: Titel, Cass-Nr., Filmart, Bemerk, uvm. Drucken v. Filmlisten, Etiketten. Bequeme Editiermöglichkeiten. Biesler, Tel. 0 96 61 / 13 00 nach 19.00 h, 8458 Sulzbach, Max-Planck-Str. 12.

****** KOPIERSCHUTZ ******
Pirate Defence: Das professionelle Schutzsystem für Software unter ProDOS, DOS, DSR, ... Macht Disketten gegen neueste Kopierer sicher. (LOCKSMITH 6, COPY II+ 6). Info bei: C. Bregler, Tulpenstr. 2, 7519 Eppingen, 0 72 62 / 44 14.

***** SOFTWARETAUSCH *****
Kontakt mit K. Brouwer, Gustav-Schwab-Str. 2 in 7317 Wendlingen

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

Der Renner: Das neue Mittwochs-Lotto zu DM 35,–. Restposten Der Turm von Hanoi für DM 10,–. Förter, Am Wetterbach 53, 7500 Karlsruhe.

Informatik-Projekt: Grafische Darstellungen.
Info Tel. 02 11 / 33 02 70.

Ankauf Software

Suche Software für Apple IIe,c! Tel. 0 92 51 / 54 76.

Suche Software CP/M 80
Suche eine Kopie von der Original Disk. CP/M 80 56K. Vers2. 20B von Microsoft. – Gegen Bezahlung. CH-0041 / 41 5550, 77 Luzern.

Handwerkerprogramm/ Schreiner auf IIe. W. Müller, Sandhäuser Str. 1, 6900 Heidelberg, Tel. 0 62 21 / 78 11 32.

Suche Init / Treiber usw. Für BASIS 256K Ramkarte / IIe / UCSD PASCAL 1.2 (evtl. auch DOS 3.3). Ingram, Lütticherstr. 30, Köln 1.

CP/M 2.2 Systemdiskette gesucht für IIe A. Grün Telefon 0 21 51 / 30 21 77

Hausverwaltung-Programm für Apple IIe. Kallmeyer, Seegefelder 75, 1000 Berlin 20.

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

PEEKER

Abo-Karte

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

Für Ihre Unterlagen

Folgende Bücher bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Versandbuchhandlung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Buch-Karte

Bitte senden Sie mir gegen Rechnung folgende Bücher:

- ☐ 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, ca. DM 40,–
- ☐ 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

Für Ihre Unterlagen

Folgende Disketten
und Programme bestellt:

_____

_____

_____

_____

_____

am: _____

bei:

Peeker
Softwareabteilung
Postfach 10 28 69
6900 Heidelberg 1

PEEKER

Software-Karte

Bitte senden Sie mir gegen Rechnung folgende Disketten:

- ☐ Peeker-Sammeldiskette, einzeln
  Disk#_____, Disk# _____
  Disk#_____, Disk# _____
  Preis je Disk DM 28,– (einzeln)
- ☐ Peeker-Sammeldiskette, im Fortsetzungsbezug ab Disk# _____
  (Mindestbezug 6 Disketten)
  Preis je Disk DM 20,–
- ☐ 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,–
- ☐ Softbreaker 1.0, Programm, DM 20,–
- ☐ 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.

POSTKARTE

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

Buch-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

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

Software-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

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

PEEKER Börse

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

Hochauflös. Videoscanner + CAD (Elektro / Elektronik) für Apple IIe gesucht. Eilt! St. Richter 8650 Kulmbach, Kalte Marter 3.

Verkaufe f. Apple IIe
Driver Card DM 60,–. Auto-Eprom Prog. DM 100,–. Z80 Card DM 80,–. 80Z/64K DM 260,–. Visikalc-Package DM 350,–. Apple-Works DM 650,–. Tel. 06646/527.

Mailboxen in der Schweiz: die interessantesten Auszüge auf randvoller Apple-Diskette gegen Einsendung von DM/Fr. 10,– an Matthias Meyer, Postfach 191, CH-9001 St. Gallen

!!!! Da gibt's was umsonst !!!!
4 × im Jahr den neuen Katalog. Bühler Elektronik, Postfach 32, 7570 Baden-Baden

Systemwechsel Superpreise alles neuwertig! Accelerator IIe 450,–, Word-Juggler (Dt. Textv. incl. Wörterb.) 450,–, IIc 1700,–, Imagewriter 800,–, 10 MB-HD 2500,–; Gernot Eiler, Tel. 0043-05513/67083

* **DISKETTEN** *
* 5 1/4", 48tpi, DM 1,99 *
* 3 1/2", 135tpi, DM 4,65 *
* auch andere, 6 Mon. Garant. *
* Allg. Austro-AG, Ringstr. 10 *
* D-8057 Eching, T. 08133/6116 *

Apple IIe-64KB, Monitor II, DuoDisk-Laufwerk, Superserielle Karte, 80Z Karte. Incl. Software z.B. Apple Writer II, Quick File und vieles mehr sowie Handbücher. Neupreis ohne Software DM 4300,– Verkaufspreis komplett DM 2700,– Tel. 08389/256

Suche schriftl. Erfahrungsaustausch Apple IIe & (Kyan) Pascal, C. Heeren, Leliestr. 56, NL-Eibergen.

PASCMAK: Suche Vertriebspartner und Interessenten für mein neues Immobilienmakler-Programm. Heinz, Waldgürtel 7, 5060 Berg. Gld. 1

Ihre Erphi-Vertretung für die Schweiz: Beltronic, Im Chapf, 8455 Rüdlingen, Tel. 018673141, Telex 825981.

Tausch

Tausche Apple comp. 64K, Z80, CPU, IBM-Gehäuse, separate Tastatur 80-Zeichen, 2 Laufwerke 256K und Monitor gegen Apple IIc + Monitor. Tel. 02406/79883.

Inserentenverzeichnis Peeker 9/86

Der nächste Peeker, Heft 10/1986, erscheint in der vierten September-Woche

TurtleGraphics-Library-Paket
von Dieter Geiß

Turtle-Utilities für Fenstertechnik und Apple-Maus in einfacher und doppelter Hires-Grafik für Pascal 1.2 auf Apple IIe/c mit Maus oder Joystick. 2 Disketten mit umfangreichem Manual, DM 98,–. Unter Pascal 1.1 mit 64K nur eingeschränkt lauffähig

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

Hüthig Software Service · Postfach 10 28 69 · 6900 Heidelberg

Assembler-Syntax & Operations-beschreibung	Flags XNZVC	s	Effektive Adresse (ea) Dn Code :Cy	An Code :Cy	(An) Code :Cy	(An)+ Code :Cy	-(An) Code :Cy	d(An) Code :Cy	d(An,Rx) Code :Cy	xx.W Code :Cy	xxxx.L Code :Cy	d(PC) Code :Cy	d(PC,Rx) Code :Cy	#xxx Code :Cy	CCR/SR Code :Cy
LSL ea	***Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
X/C<=(ea)<-Ø		W	——	——	16172n:16	16173n:16	16174n:18	16175n:2Ø	16176n:22	16177Ø:2Ø	16177l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
LSR ea	***Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
Ø->(ea)=>X/C		W	——	——	16132n:16	16133n:16	16134n:18	16135n:2Ø	16136n:22	16137Ø:2Ø	16137l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
MOVE.s ea,Dm	-**ØØ	B	Ø1mØ2n: 4	——	Ø1mØ2n: 8	Ø1mØ3n: 8	Ø1mØ4n:1Ø	Ø1mØ5n:12	Ø1mØ6n:14	Ø1mØ7Ø:12	Ø1mØ7l:16	Ø1mØ72:12	Ø1mØ73:14	Ø1mØ74: 8	——
(ea)->Dm		W	Ø3mØØn: 4	Ø3mØ1n: 4	Ø3mØ2n: 8	Ø3mØ3n: 8	Ø3mØ4n:1Ø	Ø3mØ5n:12	Ø3mØ6n:14	Ø3mØ7Ø:12	Ø3mØ7l:16	Ø3mØ72:12	Ø3mØ73:14	Ø3mØ74: 8	——
		L	Ø2mØØn: 4	Ø2mØ1n: 4	Ø2mØ2n:12	Ø2mØ3n:12	Ø2mØ4n:14	Ø2mØ5n:16	Ø2mØ6n:18	Ø2mØ7Ø:16	Ø2mØ7l:2Ø	Ø2mØ72:16	Ø2mØ73:18	Ø2mØ74:12	——
MOVE.s ea,(Am)	-**ØØ	B	Ø1m2Ø: 8	——	Ø1m22:12	Ø1m23:12	Ø1m24:14	Ø1m25:16	Ø1m26:18	Ø1m27Ø:16	Ø1m27l:2Ø	Ø1m272:16	Ø1m273:18	Ø1m274:12	——
(ea)->(Am)		W	Ø3m2Ø: 8	Ø3m21n: 8	Ø3m22:12	Ø3m23:12	Ø3m24:14	Ø3m25:16	Ø3m26:18	Ø3m27Ø:16	Ø3m27l:2Ø	Ø3m272:16	Ø3m273:18	Ø3m274:12	——
		L	Ø2m2Ø:12	Ø2m21n:12	Ø2m22:2Ø	Ø2m23:2Ø	Ø2m24:22	Ø2m25:24	Ø2m26:26	Ø2m27Ø:24	Ø2m27l:28	Ø2m272:24	Ø2m273:26	Ø2m274:2Ø	——
MOVE.s ea,(Am)+	-**ØØ	B	Ø1m3Ø: 8	——	Ø1m32:12	Ø1m33:12	Ø1m34:14	Ø1m35:16	Ø1m36:18	Ø1m37Ø:16	Ø1m37l:2Ø	Ø1m372:16	Ø1m373:18	Ø1m374:12	——
(ea)->(Am),(Am)+ss->Am		W	Ø3m3Ø: 8	Ø3m31n: 8	Ø3m32:12	Ø3m33:12	Ø3m34:14	Ø3m35:16	Ø3m36:18	Ø3m37Ø:16	Ø3m37l:2Ø	Ø3m372:16	Ø3m373:18	Ø3m374:12	——
		L	Ø2m3Ø:12	Ø2m31n:12	Ø2m32:2Ø	Ø2m33:2Ø	Ø2m34:22	Ø2m35:24	Ø2m36:26	Ø2m37Ø:24	Ø2m371:28	Ø2m372:24	Ø2m373:26	Ø2m374:2Ø	——
MOVE.s ea,-(Am)	-**ØØ	B	Ø1m4Ø: 8	——	Ø1m42:12	Ø1m43:12	Ø1m44:14	Ø1m45:16	Ø1m46:18	Ø1m47Ø:16	Ø1m47l:2Ø	Ø1m472:16	Ø1m473:18	Ø1m474:12	——
(Am)-ss->Am,(ea)->(Am)		W	Ø3m4Ø: 8	Ø3m41n: 8	Ø3m42:12	Ø3m43:12	Ø3m44:14	Ø3m45:16	Ø3m46:18	Ø3m47Ø:16	Ø3m47l:2Ø	Ø3m472:16	Ø3m473:18	Ø3m474:12	——
		L	Ø2m4Ø:14	Ø2m41n:14	Ø2m42:2Ø	Ø2m43:2Ø	Ø2m44:22	Ø2m45:24	Ø2m46:26	Ø2m47Ø:24	Ø2m471:28	Ø2m472:24	Ø2m473:26	Ø2m474:2Ø	——
MOVE.s ea,d(Am)	-**ØØ	B	Ø1m5Ø:12	——	Ø1m52:16	Ø1m53:16	Ø1m54:18	Ø1m55:2Ø	Ø1m56:22	Ø1m57Ø:2Ø	Ø1m57l:24	Ø1m572:2Ø	Ø1m573:22	Ø1m574:16	——
(ea)->d+(Am)		W	Ø3m5Ø:12	Ø3m51n:12	Ø3m52:16	Ø3m53:16	Ø3m54:18	Ø3m55:2Ø	Ø3m56:22	Ø3m57Ø:2Ø	Ø3m57l:24	Ø3m572:2Ø	Ø3m573:22	Ø3m574:16	——
		L	Ø2m5Ø:16	Ø2m51n:16	Ø2m52:24	Ø2m53:24	Ø2m54:26	Ø2m55:28	Ø2m56:3Ø	Ø2m57Ø:28	Ø2m57l:32	Ø2m572:28	Ø2m573:3Ø	Ø2m574:24	——
MOVE.s ea,d(Am,Rx)	-**ØØ	B	Ø1m6Ø:14	——	Ø1m62n:18	Ø1m63n:18	Ø1m64n:2Ø	Ø1m65n:22	Ø1m66n:24	Ø1m67Ø:22	Ø1m67l:26	Ø1m672:22	Ø1m673:24	Ø1m674:18	——
(ea)->d+(Am)+(Rx)		W	Ø3m6Ø:14	Ø3m61n:14	Ø3m62n:18	Ø3m63n:18	Ø3m64n:2Ø	Ø3m65n:22	Ø3m66n:24	Ø3m67Ø:22	Ø3m67l:26	Ø3m672:22	Ø3m673:24	Ø3m674:18	——
		L	Ø2m6Ø:18	Ø2m61n:18	Ø2m62n:26	Ø2m63n:26	Ø2m64n:28	Ø2m65n:3Ø	Ø2m66n:32	Ø2m67Ø:3Ø	Ø2m67l:34	Ø2m672:3Ø	Ø2m673:32	Ø2m674:26	——
MOVE.s ea,xx.W	-**ØØ	B	Ø1Ø7Øn:12	——	Ø1Ø72n:16	Ø1Ø73n:16	Ø1Ø74n:18	Ø1Ø75n:2Ø	Ø1Ø76n:22	Ø1Ø77Ø:2Ø	Ø1Ø77l:24	Ø1Ø772:2Ø	Ø1Ø773:22	Ø1Ø774:16	——
(ea)->ØØxx		W	Ø3Ø7Øn:12	Ø3Ø71n:12	Ø3Ø72n:16	Ø3Ø73n:16	Ø3Ø74n:18	Ø3Ø75n:2Ø	Ø3Ø76n:22	Ø3Ø77Ø:2Ø	Ø3Ø77l:24	Ø3Ø772:2Ø	Ø3Ø773:22	Ø3Ø774:16	——
		L	Ø2Ø7Øn:16	Ø2Ø71n:16	Ø2Ø72n:24	Ø2Ø73n:24	Ø2Ø74n:26	Ø2Ø75n:28	Ø2Ø76n:3Ø	Ø2Ø77Ø:28	Ø2Ø77l:32	Ø2Ø772:28	Ø2Ø773:3Ø	Ø2Ø774:24	——
MOVE.s ea,xxxx.L	-**ØØ	B	Ø117Øn:16	——	Ø1172n:2Ø	Ø1173n:2Ø	Ø1174n:22	Ø1175n:24	Ø1176n:26	Ø1177Ø:24	Ø1177l:28	Ø1772:24	Ø1773:26	Ø1774:2Ø	——
(ea)->xxxx		W	Ø317Øn:16	Ø317l:16	Ø3172n:2Ø	Ø3173n:2Ø	Ø3174n:22	Ø3175n:24	Ø3176n:26	Ø3177Ø:24	Ø3177l:28	Ø3772:24	Ø3773:26	Ø3774:2Ø	——
		L	Ø217Øn:2Ø	Ø217l:2Ø	Ø2172n:28	Ø2173n:28	Ø2174n:3Ø	Ø2175n:32	Ø2176n:34	Ø2177Ø:32	Ø2177l:36	Ø2772:32	Ø2773:34	Ø2774:28	——
MOVE ea,CCR	*****	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(ea)->CCR		W	Ø423Øn:12	——	Ø4232n:16	Ø4233n:16	Ø4234n:18	Ø4235n:2Ø	Ø4236n:22	Ø4237Ø:2Ø	Ø4237l:24	Ø4372:2Ø	Ø4373:22	Ø4374:16	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
MOVE ea,SR	*****	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(ea)->SR		W	Ø433Øn:12	——	Ø4332n:16	Ø4333n:16	Ø4334n:18	Ø4335n:2Ø	Ø4336n:22	Ø4337Ø:2Ø	Ø4337l:24	Ø4372:2Ø	Ø4373:22	Ø4374:16	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
MOVE SR,ea	-----	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(SR)->ea		W	Ø403Øn: 6	——	Ø4032n:12	Ø4033n:12	Ø4034n:14	Ø4035n:16	Ø4036n:18	Ø4037Ø:16	Ø4037l:2Ø	——	——	——	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
MOVEA.s ea,Am	-----	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(ea)->Am		W	Ø3m1Øn: 4	Ø3m11n: 4	Ø3m12n: 8	Ø3m13n: 8	Ø3m14n:1Ø	Ø3m15n:12	Ø3m16n:14	Ø3m17Ø:12	Ø3m17l:16	Ø3m172:12	Ø3m173:14	Ø3m174: 8	——
		L	Ø2m1Øn: 4	Ø2m11n: 4	Ø2m12n:12	Ø2m13n:12	Ø2m14n:14	Ø2m15n:16	Ø2m16n:18	Ø2m17Ø:16	Ø2m17l:2Ø	Ø2m172:16	Ø2m173:18	Ø2m174:12	——
MOVEM.s ea,RL	-----	B	——	——	—— 12	—— 12	——	—— 16	—— 18	16	—— 2Ø	—— 16	—— 18	——	——
(ea)->RL		W	——	——	Ø4622n: 4	Ø4623n: 4	——	Ø4625n: 4	Ø4626n: 4	Ø4627Ø: 4	Ø4627l: 4	Ø4627Ø: 4	Ø4627Ø: 4	——	——
		L	——	——	Ø4632n: 8	Ø4633n: 8	——	Ø4635n: 8	Ø4636n: 8	Ø4637Ø: 8	Ø4637l: 8	Ø4637Ø: 8	Ø4637Ø: 8	——	——
MOVEM.s RL,ea	-----	B	——	——	—— 8	—— 8	—— 12	—— 14	—— 12	—— 16	——	——	——	——	——
(RL)->ea		W	——	——	Ø4422n: 5	——	Ø4423n: 5	Ø4425n: 5	Ø4426n: 5	Ø4427Ø: 5	Ø4427l: 5	——	——	——	——
		L	——	——	Ø4432n:1Ø	——	Ø4433n:1Ø	Ø4435n:1Ø	Ø4436n:1Ø	Ø4437Ø:1Ø	Ø4437l:1Ø	——	——	——	——
MULS ea,Dm	-**ØØ	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(Dm)*(ea)->Dm		W	14m7Øn:7Ø	——	14m72n:74	14m73n:74	14m74n:76	14m75n:78	14m76n:8Ø	14m77Ø:78	14m771:82	14m772:78	14m773:8Ø	14m774:74	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
MULU ea,Dm	-**ØØ	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(Dm)*(ea)->Dm		W	14m3Øn:7Ø	——	14m32n:74	14m33n:74	14m34n:76	14m35n:78	14m36n:8Ø	14m37Ø:78	14m371:82	14m372:78	14m373:8Ø	14m373:74	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
NBCD ea	*U*U*	B	Ø44ØØn: 6	——	Ø44Ø2n:12	Ø44Ø3n:12	Ø44Ø4n:14	Ø44Ø5n:16	Ø44Ø6n:18	Ø44Ø7Ø:16	Ø44Ø71:2Ø	——	——	——	——
Ø-(ea)-X->ea		W	——	——	——	——	——	——	——	——	——	——	——	——	——
		L	——	——	——	——	——	——	——	——	——	——	——	——	——
NEG.s ea	*****	B	Ø42Øn: 4	——	Ø42Ø2n:12	Ø42Ø3n:12	Ø42Ø4n:14	Ø42Ø5n:16	Ø42Ø6n:18	Ø42Ø7Ø:16	Ø42Ø71:2Ø	——	——	——	——
Ø-(ea)->ea		W	Ø421Øn: 4	——	Ø4212n:12	Ø4213n:12	Ø4214n:14	Ø4215n:16	Ø4216n:18	Ø4217Ø:16	Ø4217l:2Ø	——	——	——	——
		L	Ø422Øn: 6	——	Ø4222n:2Ø	Ø4223n:2Ø	Ø4224n:22	Ø4225n:24	Ø4226n:26	Ø4227Ø:24	Ø4227l:28	——	——	——	——
NEGX.s ea	*****	B	Ø4ØØn: 4	——	Ø4ØØ2n:12	Ø4ØØ3n:12	Ø4ØØ4n:14	Ø4ØØ5n:16	Ø4ØØ6n:18	Ø4ØØ7Ø:16	Ø4ØØ71:2Ø	——	——	——	——
Ø-(ea)-X->ea		W	Ø401Øn: 4	——	Ø4012n:12	Ø4013n:12	Ø4014n:14	Ø4015n:16	Ø4016n:18	Ø4017Ø:16	Ø4017l:2Ø	——	——	——	——
		L	Ø4Ø2Øn: 6	——	Ø4022n:2Ø	Ø4023n:2Ø	Ø4024n:22	Ø4025n:24	Ø4026n:26	Ø4027Ø:24	Ø4027l:28	——	——	——	——
NOT.s ea	-**ØØ	B	Ø43ØØn: 4	——	Ø4302n:12	Ø4303n:12	Ø4304n:14	Ø4305n:16	Ø4306n:18	Ø4307Ø:16	Ø4307l:2Ø	——	——	——	——
-(ea)->ea		W	Ø431Øn: 4	——	Ø4312n:12	Ø4313n:12	Ø4314n:14	Ø4315n:16	Ø4316n:18	Ø4317Ø:16	Ø4317l:2Ø	——	——	——	——
		L	Ø432Øn: 6	——	Ø4322n:2Ø	Ø4323n:2Ø	Ø4324n:22	Ø4325n:24	Ø4326n:26	Ø4327Ø:24	Ø4327l:28	——	——	——	——
OR.s ea,Dm	-**ØØ	B	1ØmØ2n: 4	——	1ØmØ2n: 8	1ØmØ3n: 8	1ØmØ4n:1Ø	1ØmØ5n:12	1ØmØ6n:14	1ØmØ7Ø:12	1ØmØ7l:16	1ØmØ72:12	1ØmØ73:14	1ØmØ74: 8	——
(Dm) v (ea)->Dm		W	1Øm1Øn: 4	——	1Øm12n: 8	1Øm13n: 8	1Øm14n:1Ø	1Øm15n:12	1Øm16n:14	1Øm17Ø:12	1Øm17l:16	1Øm172:12	1Øm173:14	1Øm174: 8	——
		L	1Øm2Øn: 8	——	1Øm22n:14	1Øm23n:14	1Øm24n:16	1Øm25n:18	1Øm26n:2Ø	1Øm27Ø:18	1Øm271:22	1Øm272:18	1Øm273:2Ø	1Øm274:14	——
OR.s Dm,ea	-**ØØ	B	——	——	1Øm42n:12	1Øm43n:12	1Øm44n:14	1Øm45n:16	1Øm46n:18	1Øm47Ø:16	1Øm47l:2Ø	——	——	——	——
(ea) v (Dm)->ea		W	——	——	1Øm52n:12	1Øm53n:12	1Øm54n:14	1Øm55n:16	1Øm56n:18	1Øm57Ø:16	1Øm57l:2Ø	——	——	——	——
		L	——	——	1Øm62n:2Ø	1Øm63n:2Ø	1Øm64n:22	1Øm65n:24	1Øm66n:26	1Øm67Ø:24	1Øm67l:28	——	——	——	——
ORI.s #xxx,ea	-**ØØ	B	ØØØØØn: 8	——	ØØØØ2n:16	ØØØØ3n:16	ØØØØ4n:18	ØØØØ5n:2Ø	ØØØØ6n:22	ØØØ07Ø:2Ø	ØØØ07l:24	——	——	——	ØØØ074:2Ø
(ea) v xxx->ea		W	ØØØ1Øn: 8	——	ØØØ12n:16	ØØØ13n:16	ØØØ14n:18	ØØØ15n:2Ø	ØØØ16n:22	ØØØ17Ø:2Ø	ØØØ171:24	——	——	——	ØØØ174:2Ø
		L	ØØØ2Øn:16	——	ØØØ22n:28	ØØØ23n:28	ØØØ24n:3Ø	ØØØ25n:32	ØØØ26n:34	ØØØ27Ø:32	ØØØ271:36	——	——	——	——
PEA ea	-----	B	——	——	——	——	——	——	——	——	——	——	——	——	——
ea->-(SP)		W	——	——	——	——	——	——	——	——	——	——	——	——	——
		L	——	——	4412n:12	——	——	4415n:16	4416n:2Ø	4417Ø:16	4417l:2Ø	4417Ø:16	4417Ø:2Ø	——	——
ROL ea	-**Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
C<=(ea)<=(ea)[F]		W	——	——	16372n:16	16373n:16	16374n:18	16375n:2Ø	16376n:22	16377Ø:2Ø	16377l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
ROR ea	-**Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
(ea)[Ø]=>(ea)=>C		W	——	——	16332n:16	16333n:16	16334n:18	16335n:2Ø	16336n:22	16337Ø:2Ø	16337l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
ROXL ea	***Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
X/C<=(ea)<=X		W	——	——	16272n:16	16273n:16	16274n:18	16275n:2Ø	16276n:22	16277Ø:2Ø	16277l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
ROXR ea	***Ø*	B	——	——	——	——	——	——	——	——	——	——	——	——	——
X=>(ea)=>X/C		W	——	——	16232n:16	16233n:16	16234n:18	16235n:2Ø	16236n:22	16237Ø:2Ø	16237l:24	——	——	——	——
um 1 Bit		L	——	——	——	——	——	——	——	——	——	——	——	——	——
Scc ea	***Ø*	B	Ø5Ø3Øn: 4	——	Ø5Ø32n:12	Ø5Ø33n:12	Ø5Ø34n:14	Ø5Ø35n:16	Ø5Ø36n:18	Ø5Ø37Ø:16	Ø5Ø37l:2Ø	——	——	——	——
Wenn cc dann FF->ea		W	——	——	——	——	——	——	——	——	——	——	——	——	——
sonst Ø->ea		L	——	——	——	——	——	——	——	——	——	——	——	——	——

Assembler-Syntax & Operationsbeschreibung	Flags XNZVC	s	Effektive Adresse (ea): Dn Code :Cy	An Code :Cy	(An) Code :Cy	(An)+ Code :Cy	-(An) Code :Cy	d(An) Code :Cy	d(An,Rx) Code :Cy	xx.W Code :Cy	xxxx.L Code :Cy	d(PC) Code :Cy	d(PC,Rx) Code :Cy	#xxx Code :Cy	CCR/SR Code :Cy
SUB.s ea,Dm	*****	B	11m000n: 4	---	11m002n: 8	11m003n: 8	11m004n:10	11m005n:12	11m006n:14	11m070:12	11m071:16	11m072:12	11m073:14	11m074: 8	---
(Dm)-(ea)->Dm		W	11m10n: 4	11m11n: 4	11m12n: 8	11m13n: 8	11m14n:10	11m15n:12	11m16n:14	11m170:12	11m171:16	11m172:12	11m173:14	11m174: 8	---
		L	11m20n: 8	11m21n: 8	11m22n:14	11m23n:14	11m24n:16	11m25n:18	11m26n:20	11m270:18	11m271:22	11m272:18	11m273:20	11m274:14	---
SUB.s Dm,ea	*****	B	---	---	11m42n:12	11m43n:12	11m44n:14	11m45n:16	11m46n:18	11m470:16	11m471:20	---	---	---	---
(ea)-(Dm)->ea		W	---	---	11m52n:12	11m53n:12	11m54n:14	11m55n:16	11m56n:18	11m570:16	11m571:20	---	---	---	---
		L	---	---	11m62n:20	11m63n:20	11m64n:22	11m65n:24	11m66n:26	11m670:24	11m671:28	---	---	---	---
SUBA.s ea,Am	-----	B	---	---	---	---	---	---	---	---	---	---	---	---	---
(Am)-(ea)->Am		W	11m30n: 8	11m31n: 8	11m32n:12	11m33n:12	11m34n:14	11m35n:16	11m36n:18	11m370:16	11m371:20	11m372:16	11m373:18	11m374:12	---
		L	11m70n: 8	11m71n: 8	11m72n:14	11m73n:14	11m74n:16	11m75n:18	11m76n:20	11m770:18	11m771:22	11m772:18	11m773:20	11m774:14	---
SUBI.s #xxx,ea	*****	B	00200n: 8	---	00202n:16	00203n:16	00204n:18	00205n:20	00206n:22	00207n:20	00207l:24	---	---	---	---
(ea)-xxx->ea		W	00210n: 8	---	00212n:16	00213n:16	00214n:18	00215n:20	00216n:22	00217n:20	00217l:24	---	---	---	---
		L	00220n:16	---	00222n:28	00223n:28	00224n:30	00225n:32	00226n:34	00227n:32	00227l:36	---	---	---	---
SUBQ.s #x,ea	*****	B	05x40n: 4	---	05x42n:12	05x43n:12	05x44n:14	05x45n:16	05x46n:18	05x470:16	05x471:20	---	---	---	---
(ea)-x->ea		W	05x50n: 4	05x51n: 8	05x52n:12	05x53n:12	05x54n:14	05x55n:16	05x56n:18	05x570:16	05x571:20	---	---	---	---
		L	05x60n: 8	05x61n: 8	05x62n:20	05x63n:20	05x64n:22	05x65n:24	05x66n:26	05x670:24	05x671:28	---	---	---	---
TAS ea	-**00	B	04530n: 4	---	04532n:14	04533n:14	04534n:16	04535n:18	04536n:20	04537n:18	04537l:22	---	---	---	---
Test von (ea) ->CCR		W	---	---	---	---	---	---	---	---	---	---	---	---	---
1->ea[7]		L	---	---	---	---	---	---	---	---	---	---	---	---	---
TST.s ea	-**00	B	04500n: 4	---	04502n: 8	04503n: 8	04504n:10	04505n:12	04506n:14	04507n:12	04507l:16	---	---	---	---
Test von (ea) ->CCR		W	04510n: 4	---	04512n: 8	04513n: 8	04514n:10	04515n:12	04516n:14	04517n:12	04517l:16	---	---	---	---
		L	04520n: 4	---	04522n:12	04523n:12	04524n:14	04525n:16	04526n:18	04527n:16	04527l:20	---	---	---	---

Anmerkungen

zu CHK : Die oberen Taktzyklen gelten für den Fall, daß ein Trap auftritt, die unteren, falls kein Trap auftritt.
zu DIVS/DIVU : Die maximal benötigten Taktzyklen errechnen sich wie folgt: <angegebener Wert> + 100 Zyklen
zu MOVEM : Die Taktzyklen errechnen sich wie folgt : <Zyklen der Zeile "B"> + <Zahl der Register> * <Zyklen der Zeile "W" bzw. "L">
zu MULS/MULU : Es handelt sich um die maximal benötigten Taktzyklen
zu NBCD : Die Negation wird im BCD-Code gerechnet
zu SCC Dn : Wenn "cc" richtig ist, werden 6 Taktzyklen benötigt

Die Zeichen in der Spalte "Flags" bedeuten:
- : Flag wird nicht verändert
0 : Flag wird auf 0 gesetzt
* : Flag wird wie üblich verändert
U : Flag hat undefinierten Wert

Teil 2: Befehle mit fester „Effektiver Adresse"

Assembler-Syntax & Operationsbeschreibung	XNZVC	s	Code : Cyc
ABCD Dn,Dm	*U*U*	B	14m400n: 6
(Dm)+(Dn)+X->Dm		W	-----
		L	-----
ABCD -(An),-(Am)	*U*U*	B	14m41n: 18
(-(Am))+(-(An))+X->(Am)		W	-----
		L	-----
ADDX.s DW,Dm	*****	B	15m40n: 4
(Dm)+(Dn)+X->Dm		W	15m50n: 4
		L	15m60n: 8
ADDX.s -(An),-(Am)	*****	B	15m41n: 18
(-(Am))+(-(An))+X->(Am)		W	15m51n: 18
		L	15m61n: 30
ASL.s Dm,Dn	*****	B	16m44n:6+2*d
X/C<=(Dn)<=0		W	16m54n:6+2*d
um (Dm) Bits		L	16m64n:8+2*d
ASL.s #x,Dn	*****	B	16x40n:6+2*x
X/C<=(Dn)<=0		W	16x50n:6+2*x
um x Bits		L	16x60n:8+2*x
ASR.s Dm,Dn	*****	B	16m04n:6+2*d
(Dn)[MSB]=>(Dn)=>X/C		W	16m14n:6+2*d
um (Dm) Bits		L	16m24n:8+2*d
ASR.s #x,Dn	*****	B	16x00n:6+2*x
(Dn)[MSB]=>(Dn)=>X/C		W	16x10n:6+2*x
um x Bits		L	16x20n:8+2*x
Bcc <Label>	-----	B	0600dd:10/ 8
Wenn cc dann (PC)+ddd->PC		W	060000:10/12
		L	-----
BRA <Label>	-----	B	0600dd: 10
(PC)+ddd->PC		W	060000: 10
		L	-----
BSR <Label>	-----	B	0604dd: 18
(PC)->-(SP),(PC)+ddd->PC		W	060400: 18
		L	-----
CMPM.s (An)+,(Am)+	-****	B	13m41n: 12
Test von ((Am)+)-((An)+) ->CCR		W	13m51n: 12
		L	13m61n: 20
DBcc Dn,<Label>	-----	B	-----
Wenn -cc dann { (Dn)-1 ->Dn,		W	05031n:10/14
Wenn (Dn)#-1 dann (PC)+ddd->PC }		L	-----
EXG Dm,Dn	-----	B	-----
(Dm)<->(Dn)		W	-----
		L	14m50n: 6
EXG Am,An	-----	B	-----
(Am)<->(An)		W	-----
		L	14m51n: 6
EXG Am,Dn	-----	B	-----
(Am)<->(Dn)		W	-----
		L	14m61n: 6

Assembler-Syntax & Operationsbeschreibung	XNZVC	s	Code : Cyc
EXT.s Dn	-**00	B	-----
(Dn) vorzeichenerweitert ->Dn		W	04420n: 4
		L	04430n: 4
ILLEGAL {nicht bei jedem Assembler vorhanden !}	-----		045378: 34
Trap wegen illegalem Befehl			
LINK An,#xxx	-----		04712n: 18
(An)->-(SP),(SP)->An,(SP)+xxx->SP			
LSL.s Dm,Dn	***0*	B	16m45n:6+2*d
X/C<=(Dn)<=0		W	16m55n:6+2*d
um (Dm) Bits		L	16m65n:8+2*d
LSL.s #x,Dn	***0*	B	16x41n:6+2*x
X/C<=(Dn)<=0		W	16x51n:6+2*x
um x Bits		L	16x61n:8+2*x
LSR.s Dm,Dn	***0*	B	16m05n:6+2*d
0=>(Dn)=>X/C		W	16m15n:6+2*d
um (Dm) Bits		L	16m25n:8+2*d
LSR.s #x,Dn	***0*	B	16x01n:6+2*x
0=>(Dn)=>X/C		W	16x11n:6+2*x
um x Bits		L	16x21n:8+2*x
MOVE USP,An	-----	B	-----
(USP)->An		W	-----
		L	04715n: 4
MOVE An,USP	-----	B	-----
(An)->USP		W	-----
		L	04714n: 4
MOVEP Dm,d(An)	-----	B	-----
(Dm)->(An)+d		W	00m61n: 16
in alternierende Bytes		L	00m71n: 24
MOVEP d(An),Dm	-----	B	-----
((An)+d)->Dm		W	00m41n: 16
von alternierenden Bytes		L	00m51n: 24
MOVEQ #xxx,Dm	-**00	B	-----
xxx->Dm		W	-----
		L	07mxxx: 4
NOP	-----		04716l: 4
keine			
RESET	-----		04716@: 132
externe Geräte normieren			
ROL.s Dm,Dn	-**0*	B	16m47n:6+2*d
C<=(ea)<=(ea)[F]		W	16m57n:6+2*d
um (Dm) Bits		L	16m67n:8+2*d
ROL.s #x,Dn	-**0*	B	16x43n:6+2*x
C<=(ea)<=(ea)[F]		W	16x53n:6+2*x
um x Bits		L	16x63n:8+2*x
ROR.s Dm,Dn	-**0*	B	16m07n:6+2*d
(ea)[0]=>(ea)=>C		W	16m17n:6+2*d
um (Dm) Bits		L	16m27n:8+2*d

Assembler-Syntax & Operationsbeschreibung	XNZVC	s	Code : Cyc
ROR.s #x,Dn	-**Ø*	B	16xØ3n:6+2*x
(ea)[Ø]=>(ea)=>C		W	16x13n:6+2*x
um x Bits		L	16x23n:8+2*x
ROXL.s Dm,Dn	***Ø*	B	16m46n:6+2*d
X/C<=(ea)<=X		W	16m56n:6+2*d
um (Dm) Bits		L	16m66n:8+2*d
ROXL.s #x,Dn	***Ø*	B	16x42n:6+2*x
X/C<=(ea)<=X		W	16x52n:6+2*x
um x Bits		L	16x62n:8+2*x
ROXR.s Dm,Dn	***Ø*	B	16mØ6n:6+2*d
X=>(ea)=>X/C		W	16m16n:6+2*d
um (Dm) Bits		L	16m26n:8+2*d
ROXR.s #x,Dn	***Ø*	B	16xØ2n:6+2*x
X=>(ea)=>X/C		W	16x12n:6+2*x
um x Bits		L	16x22n:8+2*x
RTE	*****		Ø47163: 2Ø
((SP)+)->SR,((SP)+)->PC			
RTR	*****		Ø47167: 2Ø
((SP)+)->CCR,((SP)+)->PC			
RTS	-----		Ø47165: 16
((SP)+)->PC			
SBCD Dn,Dm	*U*U*	B	1Øm4Øn: 6
(Dm)-(Dn)-X->Dm		W	----
		L	----

Assembler-Syntax & Operationsbeschreibung	XNZVC	s	Code : Cyc
SBCD -(An),-(Am)	*U*U*	B	1Øm41n: 18
(-(Am))-(-(An))-X->(Am)		W	----
		L	----
STOP #xxx	*****		Ø47162: 4
xxx->SR,Anhalten			
SUBX.s Dn,Dm	*****	B	11m4Øn: 4
(Dm)-(Dn)-X->Dm		W	11m5Øn: 4
		L	11m6Øn: 8
SUBX.s -(An),-(Am)	*****	B	11m41n: 18
(-(Am))-(-(An))-X->(Am)		W	11m51n: 18
		L	11m61n: 3Ø
SWAP Dn	-**ØØ	B	----
(Dn)[1F...1Ø]<->(Dn)[F...Ø]		W	Ø441Øn: 4
		L	----
TRAP #xx	-----		Ø471xx: 34
Trap Nummer xx			
TRAPV	-----		Ø4716: 34/4
Wenn V=1 dann Trap			
UNLK	-----		Ø4713n: 12
(An)->SP,(SP)+ ->An			

Anmerkungen

zu ABCD/SBCD : Die Rechnung erfolgt im BCD-Code
zu Bcc/TRAPV : Bei den Taktzyklen gilt der erste Wert, wenn ein Sprung erfolgt, ansonsten der zweite
zu DBcc : Siehe auch "Bcc". Die Werte gelten, falls "cc" falsch ist. Sonst lauten die Werte "--/12"

zu den Schiebe- und Rotierbefehlen der Form "Dm,Dn" : Bei den Taktzyklen ist d := (Dm)

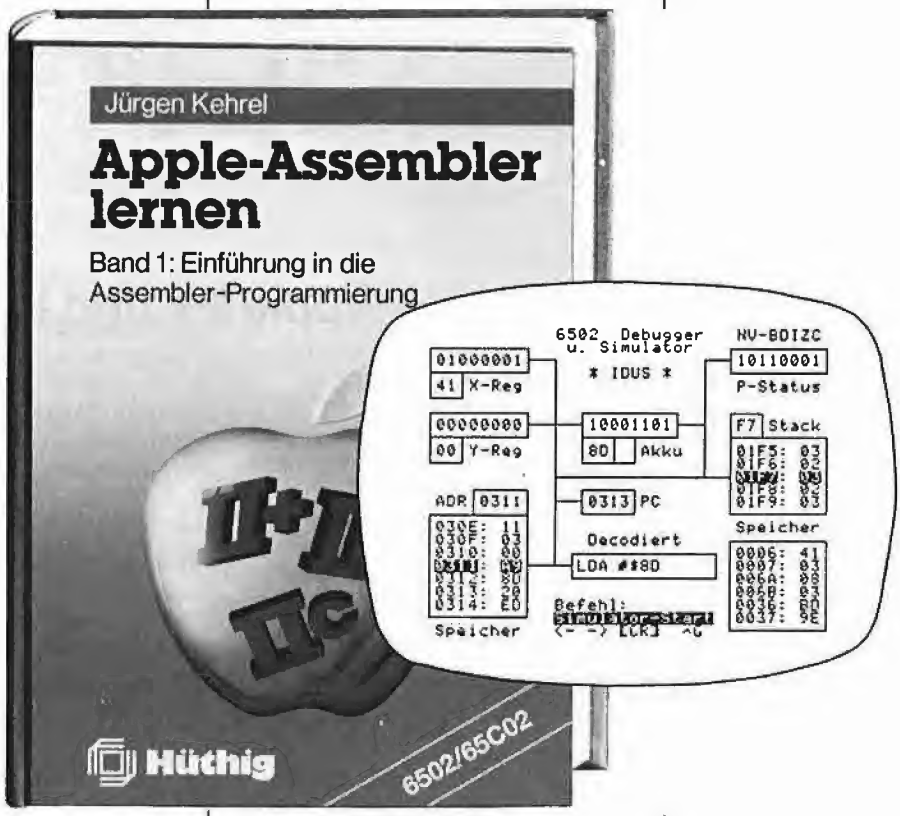
Die Zeichen in der Spalte "Flags" bedeuten:

- : Flag wird nicht verändert
Ø : Flag wird auf Ø gesetzt
* : Flag wird wie üblich verändert
U : Flag hat undefinierten Wert

Tabelle 5: Die Bedingungscodes vom 68000

Mnemonic	Condition	Bedingung (deutsch)	Code	Test	
T	True	wahr	ØØØØ	1	
F	False	unwahr	ØØØ1	Ø	
HI	HIgh	größer	ØØ1Ø	-C&-Z	-\
LS	Low or Same	kleiner oder gleich	ØØ11	C v Z	\ ohne
CC	Carry Clear	Übertrag zurückgesetzt	Ø1ØØ	-C	/ Vorzeichen
CS	Carry Set	Übertrag gesetzt	Ø1Ø1	C	-/
NE	Not Equal	ungleich	Ø11Ø	-Z	
EQ	EQual	gleich	Ø111	Z	
VC	oVerflow Clear	Überlauf zurückgesetzt	1ØØØ	-V	
VS	oVerflow Set	Überlauf gesetzt	1ØØ1	V	
PL	PLus	positiv	1Ø1Ø	-N	
MI	MInus	negativ	1Ø11	N	
GE	Greater or Equal	größer oder gleich	11ØØ	N&V v -N&-V	-\
LT	Less Than	kleiner	11Ø1	N&-V v -N&V	\ mit
GT	Greater Than	größer	111Ø	N&V&-Z v -N&-V&-Z	/ Vorzeichen
LE	Less or Equal	kleiner oder gleich	1111	Z v N&-V v -N&V	-/

FACHBUCHTIP Hüthig



Apple-Assembler lernen

von Jürgen Kehrel

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

1985, 234 S., kart., DM 38,—
ISBN 3-7785-1151-3
Begleitdiskette zu Bd. 1:
DM 44,—
ISBN 3-7785-1243-9

Band 2: Nutzung besonderer Apple-Eigenschaften

1986, 276 S., kart., DM 38,—
ISBN 3-7785-1170-X
Begleitdiskette zu Bd. 2:
DM 44,—
ISBN 3-7785-1244-7

Das zweibändige Werk „Apple Assembler lernen" ist ein kompletter Kurs in der Assemblerprogrammierung des 6502 oder 65C02 auf dem Apple II, der nicht da aufhört, wo andere Einführungen auf „weiterführende Literatur" verweisen. Starkes Gewicht wird auf die praktische Anwendung gelegt. Deshalb gehört zum Kurs ein vollwertiger 2-Pass Assembler, der als einer von wenigen die erweiterten Befehle der neuen IIc und IIe Prozessoren 65C02 verarbeitet und der auch lange Programme von mehr als 1 000 Zeilen in wenigen Sekunden übersetzt. Zu seinen professionellen Eigenschaften gehören neben 14 Pseudo-Opcodes, die die Arbeit mit Strings und Tabellen zu einem Kinderspiel werden lassen, ein Zeileneditor mit viel Komfort und die Fähigkeit, Quellcode in verschiedenen Formaten zu schreiben und zu lesen. Ein interaktiver Debugger und Simulator hilft Ihnen, eigene und fremde Maschinenprogramme zu verstehen. Mit seinen vielfältigen und mächtigen Möglichkeiten läßt er Sie hinter die Kulissen Ihres Rechners schauen. Sie können „sehen", was abläuft, Ihre Vorstellungskraft wird angeregt und nicht nur einfach Ihr Gedächtnis strapaziert.

Alle Programme sind 100 % kompakter Maschinencode, nicht einfach compiliertes Basic. Selbstverständlich lernt der Leser sämtliche Maschinenbefehle des Apple und die wichtigen Grundalgorithmen. Aber auch der Umgang mit den eingebauten ROM-Routinen wird ausführlich geübt. Grafik, Sound Stringverwaltung, Fließkommaarithmetik werden ebenso behandelt oder das Zusammenwirken von Applesoft und Assemblerprogrammen.

BESTELLCOUPON

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

Buchtitel

Name

Straße

Ort

Unterschrift

Wie man Programme „entschützt“

Ein „Hardbreaker“ für Apple II+ und IIe

von Willi Porten

Dieser Artikel soll aufzeigen, wie man ein Programm, welches auf einem Apple II+ oder IIe mit 48K läuft, entschützen kann. Voraussetzung hierzu ist eine 16K-Karte (im IIe bereits eingebaut) und eine kleine Schaltung.
Dabei können mit dieser von mir entwikkelten Methode fast alle RAM-residenten Programme zu jedem beliebigen Zeitpunkt unterbrochen, auf Diskette gespeichert und dann später genau da wieder gestartet werden, wo sie vorher unterbrochen wurden. Ausgenommen sind hier Programme, die den Inhalt des kompletten Monitor-ROMs überprüfen, die glücklicherweise jedoch sehr selten sind. Außerdem kann man damit auch Grafik-Bilder aus Programmen retten und dann später mit Hilfe anderer Programme ausdrucken.
„RAM-resident“ bedeutet hier, daß das Programm, wenn es einmal im Speicher ist, keine anderen Programmteile nachladen darf. In diesem Sinne sind Programme, die mit Overlays arbeiten, nicht RAM-resident.
Bevor Sie diesen Artikel weiterlesen, sollten Sie sich über die rechtliche Seite des Duplizierens von Programmen informieren (s. „Software und Kopierrecht“, Peeker 4/85, S. 6).

1. Das Prinzip

Wie kann man es nun schaffen, jedes beliebige Programm zu jedem Zeitpunkt zu unterbrechen und dann später genau dort weiterlaufen zu lassen, wo es vorher unterbrochen wurde? Nach einigem Umhersuchen stieß ich auf die Lösung dieses Problems: den „Nonmaskable Interrupt“ (NMI).
Wenn bei der 6502-CPU ein NMI-Signal auftritt, so beendet der Prozessor erst einmal den Befehl, welcher gerade bearbeitet wird. Die CPU überprüft dabei nicht – wie beim IRQ –, ob das Interrupt-Disable-Flag gesetzt ist (SEI, CLI), sondern rettet in jedem Fall den Programmzähler und das Status-Register auf den Stack. Danach ruft der Prozessor die Routine auf, welche durch den NMI-Vektor $FFFA/$FFFB festgelegt wurde. Somit kann man, wenn man den NMI-Eingang der CPU benutzt, jedes Programm anhalten und sich merken, wo genau und mit welchen Register-Inhalten das Programm unterbrochen wurde. Dieses Verfahren hat im Gegensatz zum Programmabbruch mittels des Reset-Vektors (Softbreaker) den Vorzug, daß sich der Programmzähler nach einem NMI auf dem Stack befindet. Wenn jetzt der gesamte Programmspeicher auf Diskette gerettet wird, kann später dieses Programm an genau der gleichen Stelle, an der es unterbrochen wurde, fortgesetzt werden.

2. Die zusätzliche Hardware

Um nun diesen NMI benutzen zu können, ist es notwendig, eine kleine Zusatzschaltung im Wert von ca. DM 3,- aufzubauen, was aber vollkommen unkompliziert ist. **Abb. 1** zeigt den entsprechenden Schaltplan.
Der Schaltungsaufwand beschränkt sich auf zwei Teile:

Zum einen muß der mechanische Taster entprellt werden, um nicht mehrere NMI-Signale bei einem Tastendruck zu erzeugen. Dies wird in gewohnter Weise durch die beiden rückgekoppelten NAND-Glieder erreicht. Zum anderen muß der NMI-Pegel durch den hochohmigen Ausgang vom Bus abgetrennt werden, um nicht in Konflikt mit anderen Peripheriegeräten zu

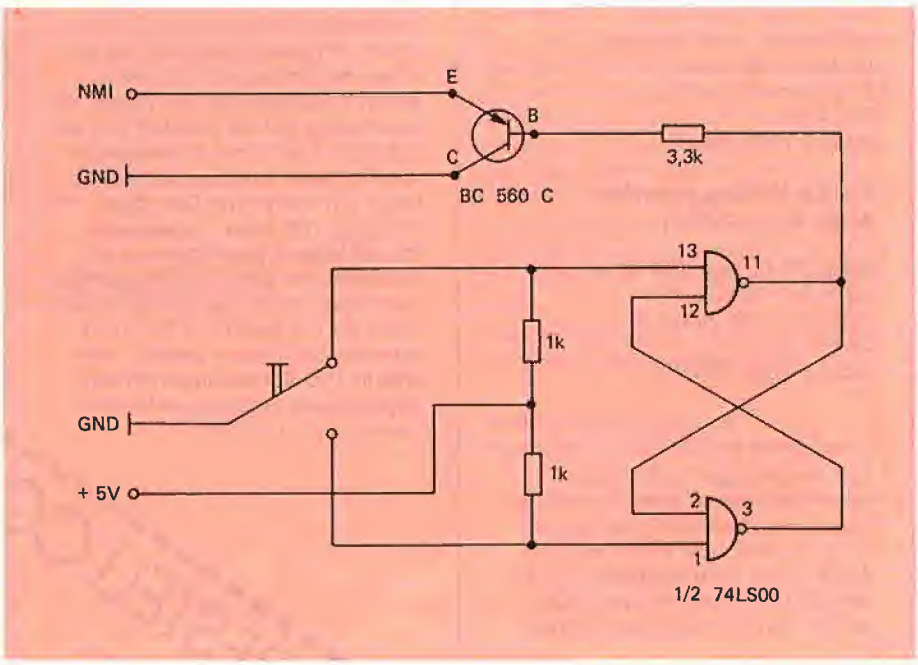


Abb. 1

kommem, die ebenfalls NMI-Impulse erzeugen.
Die Stromversorgung und der NMI-Eingang können entweder direkt auf dem Motherboard oder einer beliebigen Interface-Karte abgegriffen werden. Wer darauf nicht löten will (um z.B. den Garantieanspruch nicht zu verlieren), sollte sich eine Experimentierkarte für den Apple kaufen. Ansonsten kann der Aufbau auf einer kleinen Lochraster-Platine erfolgen.

3. Das Programm HARDBREAKER

Das Programmsystem **HARDBREAKER** funktioniert nun wie folgt (s.a. Kurzanleitung):
Nach dem Booten der HARDBREAKER-Diskette (auf welcher sich die Programme „HARDBREAKER", „HARDBREAKER.INIT", „SAVEMEM", „NMI", „LCMOVE" und „GEN.EXEC" befinden) wird das Programm HARDBREAKER gestartet. Dann wählt man aus dem Menü Punkt „1" aus.
Der daraufhin ausgeführte EXEC-File kopiert nun das Applesoft-BASIC und den Monitor in die Language-Card. Nun verändert er den NMI-Vektor, der danach auf eine Routine zeigt, welche die Bank 1 der Language-Card einschaltet und nach $D000, also der NMI-Routine, springt. Da diese Routine bei $FCCA liegt, wird so die Kassettenrekorder-Routine des Monitors unbrauchbar.
Danach sollte die Language-Card blockiert werden, da einige Programme die Karte auf jeden Fall benutzen, wenn sie vorhanden ist. Dies ist aber von uns nicht erwünscht, da diese Programme dann oft unseren NMI-Vektor und die NMI-Routine zerstören. Deshalb sollte die Language-Card hardwaremäßig geschützt werden. Ohne diese Blockierung der Language-Card schaltet z. B. das Apple-DOS-3.3 direkt nach dem Booten die Language-Karte ab und das Monitor-ROM ein. Wenn aber das ROM eingeschaltet ist, kann der Hardbreaker nicht mehr funktionieren. Abb. 1 zeigt die Schaltung zum Schutz der Language-Card, die nur den Device-Select der Language-Card abschaltet, so daß die $CO8x-Softswitches unwirksam werden. Beim BASIS 108 ist diese Schaltung nicht nötig, da der BASIS einen zusätzlichen Softswitch ($COOF) hat, um $CO8x unwirksam zu schalten. Wer auf seiner Language-Card nicht löten möchte, dem sei gesagt, daß dieser zusätzliche Aufwand für viele Programme nicht notwendig ist.

Als nächsten Schritt nimmt man die HARDBREAKER-Diskette aus dem Laufwerk und bootet durch Betätigung irgendeiner Taste das Programm, welches entschützt werden soll. Nachdem man sicher ist, daß das komplette Programm geladen wurde, nimmt man die Diskette aus dem Laufwerk, schiebt wieder die HARDBREAKER-Diskette hinein (auf der noch mindestens 203 Sektoren frei sein müssen), legt den Schalter der Language-Card von Blockiert auf Aktiv um und drückt den NMI-Taster. Dadurch wird die NMI-Routine ausgeführt, welche nun, nachdem die Register A, X, Y und der Stackpointer gerettet wurden, die Bereiche von $0000 bis $0CFF und $9600 bis $BFFF in der Language-Card abspeichert.
Die Diskette wird nun gebootet, und im Menü des HARDBREAKER-Programms wählt man „2" (SAVEMEM) aus. Dieser EXEC-File legt den 48K-Speicher des Programmes als FILE.1 bis FILE.4 ab, indem er die durch das DOS zerstörten Bereiche aus der 16K-Karte nimmt. Danach wird das Programm **GEN.EXEC** gestartet, in welchem abgefragt wird, wie das gerettete Programm heißen soll.
Jetzt erfolgt eine Abfrage, auf welcher Bildschirmseite das Programm unterbrochen wurde. Falls man dies nicht genau weiß, probiert man einfach eine Option aus. Sollte das Programm auf einer exotischen Bildschirmseite starten (z.B. Lores-Grafik mit 4 Zeilen Text), so kann man GEN.EXEC leicht modifizieren. Hat man falsch gewählt, kann das Programm mit RUN GEN.EXEC neu gestartet werden. Wenn das Programm dann mit einer Fehlermeldung endet, so hat dies nichts zu sagen, da die Files von FILE.x schon beim vorherigen Male in NAME.x umbenannt wurden. Wenn GEN.EXEC beendet wurde, ist das gerettete Programm in den Files Name.1 bis Name.4 abgespeichert. Mit „EXEC Name" kann es jederzeit gestartet werden.

Abb. 2. Experimentierplatine

Kurzhinweise

1. Zweck:
Hard- und Software zum „Entschützen" von Programmen
2. Konfiguration:
II+ (mit LC) oder IIe, Zusatzschaltung; DOS 3.3
3. Test:
siehe Kurzanleitung
4. Sammeldisk:
HARDBREAKER
(Applesoft-Startprogramm)
GEN.EXEC
(Applesoft-Programm zur Generierung der EXEC-Startdatei)
T.NMI
T.LCMOVE
(Big-Mac-Quelltexte)
NMI
LCMOVE
(Maschinenprogramme)
HARDBREAKER.INIT
HARDBREAKER.INIT.BASIS (für BASIS 108)
SAVEMEM
(EXEC-Dateien)

Kurzanleitung

1. Diskette mit HARDBREAKER als HELLO-Programm initialisieren.
2. Folgende Dateien zusätzlich auf diese Diskette kopieren:
   a) GEN.EXEC
   b) HARDBREAKER.INIT
      (bei einem BASIS 108 die Datei HARDBREAKER.INIT.BASIS kopieren und in HARDBREAKER.INIT umbenennen)
   c) SAVEMEM
   d) NMI
   e) LCMOVE
3. HARDBREAKER-Diskette booten und <1> aus Menü wählen.
4. Programmdiskette booten.
5. HARDBREAKER-Diskette einlegen.
6. NMI-Schalter betätigen und dann <2> aus Menü wählen.
7. Gewünschten Namen eingeben.

Das "entschützte" Programm wird dann durch:
EXEC Name
gestartet.

HARDBREAKER

```
100 HIMEM: 3328
110 REM ********************
120 REM *   HARDBREAKER    *
130 REM *                  *
140 REM * W. Porten, 1985 *
150 REM ********************
160 D$ = CHR$ (13) + CHR$ (4)
170 HOME : PRINT "WELCHES PROGRAMM SOLL GESTARTET WERDEN?"
180 VTAB 5: PRINT "<1> HARDBREAKER.INIT"
190 PRINT : PRINT "<2> SAVEMEM": PRINT
200 HTAB 2: GET N$: IF N$ = "2" THEN 230
210 IF N$ < > "1" THEN 200
220 PRINT D$"EXECHARDBREAKER.INIT": END
230 PRINT D$"EXECSAVEMEM": END
```

GEN.EXEC

```
100 REM ********************
110 REM *    GEN.EXEC      *
120 REM *                  *
130 REM * W. Porten, 1985 *
140 REM ********************
150 D$ = CHR$ (13) + CHR$ (4)
160 HOME
170 INPUT "NAME DES PROGRAMMES: >";N$: IF N$ = "" THEN 160
180 IF LEN (N$) > 20 THEN PRINT : PRINT "DER NAME DARF
    HOECHSTENS 20 ZEICHEN": PRINT "HABEN!": PRINT : GOTO 170
190 A = ASC ( LEFT$ (N$,1)): IF A < 65 OR A > 90 THEN PRINT :
    PRINT "DER NAME MUSS MIT EINEM BUCHSTABEN": PRINT "BEGINNEN!":
    PRINT : GOTO 170
200 PRINT : PRINT "AUF WELCHEM SCREEN SOLL DAS PROGRAMM"
210 PRINT : PRINT "GESTARTET WERDEN?"
220 PRINT : PRINT : PRINT "<1>  TEXT    ($0400-$07FF)"
230 PRINT : PRINT "<2>  TEXT2  ($0800-$0BFF)"
240 PRINT : PRINT "<3>  HGR    ($2000-$3FFF)"
250 PRINT : PRINT "<4>  HGR2   ($4000-$5FFF)"
260 PRINT : PRINT
270 PRINT ">";
280 GET NU$:NU = VAL (NU$): IF NU < 1 OR NU > 4 THEN 280
290 PRINT NU
300 PRINT D$"OPEN"N$: PRINT D$"WRITE"N$
310 PRINT "NOMONCIO"
320 PRINT "CALL-151"
330 PRINT "C081 C081"
340 PRINT "F800<F800.FFFFM"
350 ON NU GOSUB 460,470,480,490
360 PRINT "BLOAD"N$",2"
370 PRINT "BLOAD"N$",3"
380 PRINT "BLOAD"N$",4"
390 PRINT "C08B C08B"
400 PRINT "BLOAD"N$",1"
410 PRINT "BLOADLCMOVE,A$F700"
420 PRINT "F700G"
430 PRINT D$"CLOSE"N$
440 FOR I = 1 TO 4: PRINT D$"RENAMEFILE."I","N$"."I: NEXT I
450 END
460 RETURN
470 PRINT "C055": RETURN
480 PRINT "C054 C057 C052 C050": RETURN
490 PRINT "C055 C057 C052 C050": RETURN
```

NMI

BSAVE NMI, A$D000, L$00B5

```
                        1   ********************
                        2   *       NMI        *
                        3   *                  *
                        4   * W. Porten, 1985 *
                        5   ********************
                        6   *
                        7   * Dieses Programm rettet Teile
                        8   * des 48K-Bereiches in die
                        9   * Language-Card und bootet dann
                       10   *
                       11            ORG  $D000
                       12
                       13   START2   EQU  $F700
                       14   BOOT     EQU  $FA62
                       15   LDA      EQU  $AD
                       16
                       17   * Bank 1 schreibfähig schalten
                       18
D000: 2C 8B C0         19            BIT  $C08B
D003: 2C 8B C0         20            BIT  $C08B
                       21
                       22   * Register retten
                       23
D006: 8D 1D D1         24            STA  ASAVE
D009: 8E 1E D1         25            STX  XSAVE
D00C: 8C 1F D1         26            STY  YSAVE
D00F: BA               27            TSX
D010: 8E 20 D1         28            STX  SSAVE
                       29
                       30   * Programm kopieren und starten
                       31
D013: A2 00            32            LDX  #0
D015: BD 21 D0         33   LOOP     LDA  WEITER,X
D018: 9D 00 F7         34            STA  START2,X
D01B: E8               35            INX
D01C: D0 F7            36            BNE  LOOP
D01E: 4C 00 F7         37            JMP  START2
                       38
                       39   * Jetzt geht es im gemeinsamen
                       40   * Bereich von Bank 0 und Bank 1
                       41   * der Language-Card weiter
                       42
                       43   * TRANSFER D000<0000,0CFF
                       44
                       45   WEITER
                       46
                       47   OFFSET   EQU  START2-WEITER
                       48
D021: AD 00 00         49   LOAD     DFB  LDA,0,0    *LDA $0000
                       50   *
                       51   * notwendig, da sonst der
                       52   * 2-Byte-Zero-Page-Befehl vom
                       53   * Assembler generiert wird
                       54   *
D024: 8D 00 D0         55   STORE    STA  $D000
D027: EE 01 F7         56            INC  LOAD+1+OFFSET
D02A: EE 04 F7         57            INC  STORE+1+OFFSET
D02D: D0 F2            58            BNE  LOAD
D02F: EE 02 F7         59            INC  LOAD+2+OFFSET
D032: EE 05 F7         60            INC  STORE+2+OFFSET
D035: AD 02 F7         61            LDA  LOAD+2+OFFSET
D038: C9 0D            62            CMP  #$0D
D03A: D0 E5            63            BNE  LOAD
                       64
                       65   * TRANSFER DD00<9600,98FF
                       66
D03C: AD 00 96         67   LOAD1    LDA  $9600
D03F: 8D 00 DD         68   STORE1   STA  $DD00
D042: EE 1C F7         69            INC  LOAD1+1+OFFSET
D045: EE 1F F7         70            INC  STORE1+1+OFFSET
D048: D0 F2            71            BNE  LOAD1
D04A: EE 1D F7         72            INC  LOAD1+2+OFFSET
D04D: EE 20 F7         73            INC  STORE1+2+OFFSET
D050: AD 1D F7         74            LDA  LOAD1+2+OFFSET
D053: C9 99            75            CMP  #$99
D055: D0 E5            76            BNE  LOAD1
                       77
                       78   * Bank 0 schreibfähig schalten
                       79
D057: 2C 83 C0         80            BIT  $C083
D05A: 2C 83 C0         81            BIT  $C083
                       82
                       83   * TRANSFER D000<9900,BFFF
                       84
```

```
D05D: AD 00 99  85  LOAD2   LDA  $9900
D060: 8D 00 D0  86  STORE2  STA  $D000
D063: EE 3D F7  87          INC  LOAD2+1+OFFSET
D066: EE 40 F7  88          INC  STORE2+1+OFFSET
D069: D0 F2     89          BNE  LOAD2
D06B: EE 3E F7  90          INC  LOAD2+2+OFFSET
D06E: EE 41 F7  91          INC  STORE2+2+OFFSET
D071: AD 3E F7  92          LDA  LOAD2+2+OFFSET
D074: C9 C0     93          CMP  #$C0
D076: D0 E5     94          BNE  LOAD2
                95
                96  * ($E000) retten, da dies, falls
                97  * mit Diversi-DOS gearbeitet wird,
                98  * zerstört wird
                99
D078: AD 00 E0 100          LDA  $E000
D07B: 8D FB F7 101          STA  ESAVE+OFFSET
               102
               103  * im LC-Monitor den NMI-Vektor
               104  * wieder herstellen
               105
D07E: A9 FB    106          LDA  #$FB
D080: 8D FA FF 107          STA  $FFFA
D083: A9 03    108          LDA  #$03
D085: 8D FB FF 109          STA  $FFFB
               110
               111  * auf Kaltstart schalten
               112
D088: 8D F3 03 113          STA  $03F3
D08B: 8D F4 03 114          STA  $03F4
               115
D08E: A9 4C    116          LDA  #$4C
D090: 8D FB 03 117          STA  $03FB
D093: A9 65    118          LDA  #$65
D095: 8D FC 03 119          STA  $03FC
D098: A9 FF    120          LDA  #$FF
D09A: 8D FD 03 121          STA  $03FD
               122
               123  * Monitor patchen: FE14: BIT $C082
               124
D09D: A9 2C    125          LDA  #$2C
D09F: 8D 14 FE 126          STA  $FE14
D0A2: A9 82    127          LDA  #$82
D0A4: 8D 15 FE 128          STA  $FE15
D0A7: A9 C0    129          LDA  #$C0
D0A9: 8D 16 FE 130          STA  $FE16
               131
D0AC: A9 FA    132          LDA  #>BOOT
D0AE: 48       133          PHA
D0AF: A9 62    134          LDA  #<BOOT
D0B1: 48       135          PHA
               136
D0B2: 4C 14 FE 137          JMP  $FE14
               138
               139  * in den gepatchten Monitor
               140  * springen:
               141  * im RAM $FE14: BIT $C082
               142  * im ROM $FE17: RET
               143
               144  WEND    DS   $100-WEND+WEITER-5
               145
               146  ESAVE   DS   1
               147  ASAVE   DS   1
               148  XSAVE   DS   1
               149  YSAVE   DS   1
               150  SSAVE   DS   1
```

289 Bytes

LCMOVE

BSAVE LCMOVE, A$F700, L$007C

```
                1  ********************
                2  *     LCMOVE       *
                3  *                  *
                4  * W. Porten, 1985 *
                5  ********************
                6  *
                7  * Dieses Programm kopiert die in
                8  * die LC geretteten Teile
                9  * zurück in den 48K-Bereich und
               10  * startet dann das gerettete
               11  * Programm da, wo es unterbrochen
               12  * wurde.
               13  *
               14          ORG  $F700
               15
               16  STA     EQU  $8D
               17
               18
               19  * TRANSFER 0000<D000.DCFF
               20
F700: AD 00 D0 21  LOAD    LDA  $D000
F703: 8D 00 00 22  STORE   DFB  STA,0,0    *STA $0000
               23  * notwendig, da sonst der
               24  * 2-Byte-Zero-Page-Befehl vom
               25  * Assembler generiert wird
               26  *
F706: EE 01 F7 27          INC  LOAD+1
F709: EE 04 F7 28          INC  STORE+1
F70C: D0 F2    29          BNE  LOAD
F70E: EE 02 F7 30          INC  LOAD+2
F711: EE 05 F7 31          INC  STORE+2
F714: AD 02 F7 32          LDA  LOAD+2
F717: C9 DD    33          CMP  #$DD
F719: D0 E5    34          BNE  LOAD
               35
               36  * TRANSFER 9600<DD00.DFFF
               37
F71B: AD 00 DD 38  LOAD1   LDA  $DD00
F71E: 8D 00 96 39  STORE1  STA  $9600
F721: EE 1C F7 40          INC  LOAD1+1
F724: EE 1F F7 41          INC  STORE1+1
F727: D0 F2    42          BNE  LOAD1
F729: EE 1D F7 43          INC  LOAD1+2
F72C: EE 20 F7 44          INC  STORE1+2
F72F: AD 1D F7 45          LDA  LOAD1+2
F732: C9 E0    46          CMP  #$E0
F734: D0 E5    47          BNE  LOAD1
               48
               49  * Bank 0 schreibfähig schalten
               50
F736: 2C 83 C0 51          BIT  $C083
F739: 2C 83 C0 52          BIT  $C083
               53
               54  * TRANSFER 9900<D000,F6FF
               55
F73C: AD FB F7 56          LDA  ESAVE
F73F: 8D 00 E0 57          STA  $E000
               58
F742: AD 00 D0 59  LOAD2   LDA  $D000
F745: 8D 00 99 60  STORE2  STA  $9900
F748: EE 43 F7 61          INC  LOAD2+1
F74B: EE 46 F7 62          INC  STORE2+1
F74E: D0 F2    63          BNE  LOAD2
F750: EE 44 F7 64          INC  LOAD2+2
F753: EE 47 F7 65          INC  STORE2+2
F756: AD 44 F7 66          LDA  LOAD2+2
F759: C9 F7    67          CMP  #$F7
F75B: D0 E5    68          BNE  LOAD2
               69
               70  * Monitor patchen: FE0F: BIT $C082
               71
F75D: A9 2C    72          LDA  #$2C
F75F: 8D 0F FE 73          STA  $FE0F
F762: A9 82    74          LDA  #$82
F764: 8D 10 FE 75          STA  $FE10
F767: A9 C0    76          LDA  #$C0
F769: 8D 11 FE 77          STA  $FE11
               78
F76C: AE FF F7 79          LDX  SSAVE
F76F: 9A       80          TXS
F770: AC FE F7 81          LDY  YSAVE
F773: AE FD F7 82          LDX  XSAVE
F776: AD FC F7 83          LDA  ASAVE
               84
F779: 4C 0F FE 85          JMP  $FE0F
               86
               87  * in den gepatchten Monitor
               88  * springen:
               89  * im RAM $FE0F: BIT $C082
               90  * im Rom $FE12: RETI
               91
               92  ESAVE   EQU  $F7FB
               93
               94  ASAVE   EQU  $F7FC
               95  XSAVE   EQU  $F7FD
               96  YSAVE   EQU  $F7FE
               97  SSAVE   EQU  $F7FF
```

124 Bytes

HARDBREAKER.INIT

```
NOMONC,I,0
CALL-151
C081 C081
D000<D000.FFFFM
FFFA:CA FC
FCC9:60 2C 8B C0 4C 00 D0
C089 C089
BLOADNMI,A$D000
C080
3D0G
NEW
1HOME:?"HARDBREAKER":?"VON W. PORTEN":?"DEN LC-SCHALTER
AUF BLOCKIERT STELLEN,
"2?"DIE ZU ENTSCHUETZENDE DISK":?
"EINLEGEN UND EINE TASTE DRUECKEN!"
3NEW
RUN
FORI=1T0129:I=PEEK(-16384):NEXT
PR#6
```

HARDBREAKER.INIT.BASIS

```
Dieses Programm sollte anstelle von HARDBREAKER eingegeben
werden, falls Sie einen BASIS 108 mit APPLE II+ Autostart-
Monitor-ROM besitzen.


NOMONC,I,0
CALL-151
C081 C081
D000<D000.FFFFM
FFFA:CA FC
FCC9:60 8D 0E C0 2C 8B C0 4C 00 D0
C089 C089
BLOADNMI,A$D000
C080
C00F:00
3D0G
NEW
1HOME:?"HARDBREAKER (BASIS 108)":?"VON W. PORTEN":?
2?"DIE LANGUAGE CARD IST BLOCKIERT.":?:?"BITTE DIE ZU
ENTSCHUETZENDE DISK":?"EINLEGEN UND EINE TASTE DRUECKEN!"
3NEW
RUN
FORI=1T0129:I=PEEK(-16384):NEXT
PR#6


Damit der Hardbreaker auch auf einem BASIS 108 mit BASIS-Monitor-
ROM läuft, sind folgende Änderungen vorzunehmen:
- In NMI
  Zeile 126: STA $FE15 statt STA $FE14
  Zeile 128: STA $FE16 statt STA $FE15
  Zeile 130: STA $FE17 statt STA $FE16
  Zeile 137: JMP $FE15 statt JMP $FE14
- In LCMOVE:
  Zeile 73: STA $FE10 statt STA $FE0F
  Zeile 75: STA $FE11 statt STA $FE10
  Zeile 77: STA $FE12 statt STA $FE11
  Zeile 85: JMP $FE10 statt JMP $FE0F
- In HARDBREAKER.INIT.BASIS:
  FFFA:90 FB statt FFFA:CA FC
  FB90:8D 0E C0 2C 8B C0 4C 00 D0 statt
  FCC9:60 8D 0E C0 2C 8B C0 4C 00 D0
```

SAVEMEM

```
NOMONC,I,0
CALL-151
BSAVEFILE.2,A$D00,L$5300
BSAVEFILE.3,A$6000,L$3600
C088
BSAVEFILE.1,A$D000,L$1000
C080
BSAVEFILE.4,A$D000,L$2800
C082
FP
RUNGEN.EXEC
```

Peeker-Sammeldisketten #20 und #21

Disk#20

(UCSD-Apple-128K-Pascal-1.2-Diskette; Heft 8/1986; Achtung: Umfangreiches Softwarepaket, deshalb Einzelpreis DM 48,–; Fortsetzungspreis DM 38,–; (wird nicht automatisch an Fortsetzungsbezieher verschickt)

(1) = Zweck; (2) = Heft/Seitenzahl; (3) = Gerätekonfiguration; (4) = Betriebssystem; (5) = Programmstart; (6) = Sonstiges

PICEDIT:
SYSTEM.MISCINFO
SYSTEM.CHARSET
ASCII.FONT
GERMAN.FONT
MATH.FONT
GREECE.FONT
SUPER.SUB.FONT
MENU.GRAF
PRINTER.INFO
EPSON
IMAGEWRITER
SYSTEM.ATTACH
CRUNCHER.CODE
SYSTEM.STARTUP
SYSTEM.LIBRARY
ARROWS.KEYS
DESIGNER.CODE

(1) PIC-EDIT von Jürgen Geiß ist das Gegenstück zum Turtle-Graphics-Library-Paket, das bekanntlich über den Hüthig Software Service für DM 98,– erhältlich ist. PIC-EDIT ist ein universeller Grafik-Editor, der dem Macpaint-Programm nachempfunden ist und über ungewöhnlich leistungsfähige Befehle verfügt; (2) Heft 8/86, S. 6; (3) Apple IIc oder IIe mit erweiterter 80-Zeichenkarte; (4) UCSD-Apple-Pascal 1.2; (5) Auf der Sammeldiskette #20 befinden sich die obenstehenden Dateien außer SYSTEM.APPLE und SYSTEM.PASCAL, die Sie aus urheberrechtlichen Gründen selbst auf hierfür reservierte Dummy-Files kopieren müssen. Dabei muß es sich um 128K-Pascal-1.2-Files handeln.

Die Diskette läuft sofort mit Epson-Druckern. Für den Imagewriter muß im Filer die Datei IMAGEWRITER mit den Transfer-Befehl auf PRINTER.INFO 1 kopiert werden.

Nach dem Kopieren der zwei Pascal-Systemdateien kann die Diskette direkt gebootet werden.

Disk #21

(DOS-3.3-Diskette; Einzelpreis DM 28,–; Fortsetzungspreis DM 20,–)

Heft 8/1986
A 005 LOGIK
T 004 MLOGIC
Übungsprogramme zur Aussagenlogik; 8/86, S. 32.
RUN LOGIK (von DOS 3.3)
MLOGIK muß zunächst mit APDOS auf MBASIC-Diskette konvertiert werden.

T 019 T.NETZWERK
B 003 NETZWERK
Vernetzung von Apple-Rechnern; 8/86, S. 50.
BRUN NETZWERK (von DOS 3.3)

T 006 T.UNIFORMAT
B 002 UNIFORMAT
Formatierbefehl für Unidisk; 8/86, S. 56.
BLOAD UNIFORMAT; CALL 768

Heft 9/1986
A 003 HARDBREAKER
A 006 GEN.EXEC
T 009 T.NMI
B 003 NMI
T 007 T.LCMOVE
B 002 HARDBREAKER.INIT
T 003 HARDBREAKER.INIT.BASIS
T 002 SAVEMEM
NMI-Interrupt-Routinen für entsprechende Experimentierplatine, 9/86, S. 34.
RUN HARDBREAKER (von DOS 3.3)

T 052 FILER.PAS
Turbo-Pascal-Dateikopierprogramm, 9/86, S. 40. Muß zunächst mit APDOS auf Turbo-Disk konvertiert und dann compiliert werden.

T 026 DREIECK.TEXT
Dreiecksberechnung in UCSD-Pascal, 9/86, S. 40. Muß zunächst mit GETPAS auf UCSD-Disk konvertiert und dann compiliert werden.

T 024 STRINGUTILS.I
T 013 STRINGDEMOS.P
Kyan-Pascal-String-Include-Datei, 9/86, S. 50. Muß erst mit CONVERT auf ProDOS-Disk konvertiert und dann compiliert/assembliert werden.

A 002 EINZEL.TASC
B 021 EINZEL.RUNTIME
B 061 EINZEL.OBJ
A 056 EINZEL.KALK
T 002 EINZEL.KONST
T 006 EINZEL.KLEIN
T 006 EINZEL.GROSS
Bucheinzelkalkulation, 9/86, S. 56.
RUN EINZEL.TASC (von DOS 3.3)

A 003 FUNKTIONEN START
A 011 FUNKTIONEN ANLEITUNG
B 002 FUNKTIONEN EXC
A 057 FUNKTIONEN
Funktionsplotter für Apple II+, S. 62.
RUN FUNKTIONEN START
(von DOS 3.3)

Hüthig Software Service
Postfach 10 28 69 · 6900 Heidelberg

ProDOS-Editor 1.0

Applesoft-Editor unter ProDOS-Betriebssystem

von U. Stiehl

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

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

Unter anderem sind folgende Features implementiert worden:

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

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

Gerätevoraussetzung: Apple II+, IIe oder IIc, 40 Zeichen/Zeile

Hinweis: Der Applesoft-Editor für DOS 3.3 befindet sich auf Sammeldisk # 16 für Fortsetzungsbezieher. Siehe Peeker 4/86, S. 13.

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

Preiswerte Begleitdisketten

Bd. 1: DM 28,–; Bd. 2: DM 28,–

DM 28,–

DM 28,– (Neue Diskette für 3. Aufl.)

Hüthig Software Service
Postfach 10 28 69 · 6900 Heidelberg

Filer für CP/M

Ein Dateikopierprogramm in Turbo-Pascal

von Gerhard Runge

1. Dateien-Backup mit einem Laufwerk

Auch wer unter CP/M arbeitet, steht hin und wieder vor dem Problem, Dateien mit nur einem Diskettenlaufwerk kopieren zu müssen. Sei es, daß man nur ein Laufwerk besitzt, weil man z.B. noch eine RAM-Disk hat, oder daß das zweite Laufwerk gerade in Reparatur ist. Das zum CP/M-System gehörige Kopierprogramm ‚PIP.COM' ist hier leider völlig überfordert. Von kleinen Dateien kann man auch mit Hilfe des Programms ‚DDT.COM' und des SAVE-Befehls eine Sicherungskopie anfertigen, aber Dateien, die größer sind als die TPA, bleiben unkopierbar. Alles in allem ist diese Methode sehr behelfsmäßig und mühsam.

Läßt sich das Kopierproblem bei einem zur Reparatur geschickten Laufwerk in der Regel durch einfaches Warten erledigen, so zwingt der Umgang mit zwei verschiedenen Laufwerken zum Nachdenken. Dieser Fall dürfte gerade bei Benutzern von Computern der Apple-II-Familie eher die Regel als die Ausnahme sein. Oft will man aus Kompatibilitätsgründen sein altes 35-Spur-Laufwerk als Bootlaufwerk behalten, aber auf ein modernes doppelseitiges Laufwerk nicht verzichten. Man kann bei unterschiedlichen Laufwerken wohl wieder mit ‚PIP' kopieren, jedoch das Hin- und Herschaufeln von Dateien einschließlich des bei jedem Diskettenwechsel notwendigen ‚Ctrl-C' (und dem dadurch verursachten Ausstieg aus ‚PIP') erscheint mir reichlich umständlich. Andererseits ist es wenig einleuchtend, seine Sicherungskopien nicht auf dem Laufwerk mit der größten Kapazität zu ziehen.

2. Das Programm Filer

Eine Lösung für dieses Problem bietet das im folgenden abgedruckte Programm, das in Turbo-Pascal geschrieben wurde. Es ist in der Lage, Dateien von jedem angeschlossenem Laufwerk auf jedes andere zu kopieren, wobei sogar der Userbereich gewechselt werden kann. Es nutzt mit Hilfe von Zeigervariablen immer den gesamten zur Verfügung stehenden freien Speicher. Unter 56K-CP/M (Apple-II+-CP/M 2.20B) können z.B. pro Diskettenzugriff ca. 37K kopiert werden. ‚Filer' arbeitet schneller und komfortabler als ‚PIP', so daß es auch zum normalen Kopieren mit zwei Laufwerken gut benutzt werden kann. Die Bedienung ist sehr einfach, da der Benutzer per Menü geführt wird. Dateinamen müssen nicht eingegeben werden, sie werden angezeigt. Durch Drücken von ‚RETURN', ‚Y' oder ‚J' wird die Datei zum Kopieren vorgemerkt. Auch Dateien mit dem SYS-Attribut werden angezeigt und kopiert, ebenso R/O-Dateien. An allen wichtigen Stellen ist ein Programmabbruch mit ‚ESC' möglich.
Ein Nachteil des Programms soll hier jedoch nicht verschwiegen werden. Um einen möglichst kompakten Code zu erhalten und die Übertragbarkeit zu anderen Computern zu ermöglichen, wurde konsequent nur mit Turbo-Pascal und BDOS-Aufrufen gearbeitet. Alle Diskettenoperationen werden somit nur von CP/M erledigt – mit allen damit verbundenen Nachteilen. Ein Diskettenwechsel wird von CP/M erkannt und muß mit ‚Ctrl-C' quittiert werden, da sonst ein BDOS-Error auftritt. Das vorliegende Filerprogramm erledigt dies automatisch (BDOS-Funktion 13), es wird jedoch immer eine Systemdiskette im Bootlaufwerk erwartet. Wer nur ein Laufwerk hat, kann nur auf vorbereitete Disketten kopieren, die das CP/M-System 2.2 enthalten. Für alle anderen Laufwerke genügt es, eine dem Laufwerk entsprechend formatierte Diskette zu verwenden. Dafür enthält das Programm keine absoluten Adressen und keinerlei Tricks, die von fremden Computern nicht nachgeahmt werden können. Es wird nichts weiter als CP/M 2.2 und Turbo-Pascal 2.0 erwartet. Möglicherweise funktioniert der ‚Filer' auch mit anderen CP/M-Versionen, sicher aber nicht mit älteren Pascal-Versionen, da die Funktionen *Dispose* und *Maxavail* benötigt werden. Grundsätzlich läßt sich die Speicherplatzverwaltung auch mit *Mark*, *Release* und *Memavail* lösen, doch leider hat mein Turbo-Pascal (und offenbar auch die meisten anderen) hier einen Fehler.

3. Arbeitsweise des Filers

Nun ein paar Worte zur Arbeitsweise des Programms. Zu Beginn werden Ziel- und Quellaufwerk sowie der gewünschte Userbereich abgefragt. Wird als Antwort nur ‚RETURN' eingegeben, so nimmt das Programm die voreingestellten Werte an. Dann kommt die Aufforderung, seine Disketten in die vorgewählten Laufwerke zu legen. Sind Quelle und Ziel identisch, so wird nur die Quelldiskette angefordert.

3.1. Dateiauswahl

Als nächstes muß ein Dateinamen eingegeben werden. Dies dient zur Vorauswahl von Dateien, die man kopieren möchte. Bei dieser Eingabe sind die bekannten Jokerzeichen ‚?' und ‚*' erlaubt. Wird nur ‚RETURN' eingegeben, so entspricht das der Eingabe ‚*.*' und liefert ein komplet-

tes Inhaltsverzeichnis der eingelegten Diskette. Dies kann speziell bei doppelseitigen Disketten mit vergrößertem Directory sehr lästig werden, wenn viele kleine Files vorhanden sind, die nicht kopiert werden sollen. Bei anderen Eingaben werden nur die passenden Dateien angezeigt und dem Benutzerwunsch entsprechend in eine Liste der zu kopierenden Dateien aufgenommen. Diese Vorgehensweise hat den Vorteil, daß immer der ganze freie Speicher genutzt werden kann und somit auch mehrere Dateien gleichzeitig ohne Rückfragen (und Diskettenwechsel wie z.B. beim ‚FID' für Apple-DOS) im Speicher stehen können. Die Anzahl der Diskettenwechsel ist nur vom vorhandenen Speicherausbau (und der Konstanten *Puffergroesse*) und der Gesamtzahl der zu kopierenden Blöcke, nicht aber von der Dateienanzahl, abhängig.

3.2. Puffergröße

Die Konstante ‚Puffergroesse' bestimmt den Anteil der Datei, der pro Diskettenzugriff gelesen wird. Es wird immer *dieser* Speicherplatz reserviert, egal wie groß die Datei (bzw. der noch zu lesende Rest) wirklich ist. Die Speicherplatzausnutzung ist um so besser, je kleiner die Konstante ‚Puffergroesse' gewählt wird. Andererseits bedingt eine Verkleinerung dieser Konstante eine höhere Anzahl von Diskettenzugriffen. Wählt man ‚Puffergroesse = 128', so wird der freie Speicher auch bei vielen kleinen Dateien bis zum letzten Byte genutzt, der Kopiervorgang wird jedoch unerträglich langsam. Da CP/M immer nur einen ganzen Block adressiert, ist es ein guter Kompromiß, diesen auch komplett einzulesen. Vergrößert man ‚Puffergroesse' auf mehr als zwei bis drei Blöcke, so geht der Vorteil der höheren Lese- und Schreibgeschwindigkeit meist durch den häufigeren Diskettenwechsel wieder verloren. Genau einen Block pro Diskettenzugriff einzulesen, erscheint mir daher speziell bei Laufwerken mit mehr als 40 Spuren und Blöcken von 2K oder gar 4K eine gute Lösung zu sein. Leere Dateien werden als solche erkannt und ignoriert. Sie beanspruchen keinen Pufferspeicher und werden auf der Zieldiskette gar nicht in das Directory eingetragen.

3.3. Programmlisting

Zum Programmlisting selbst bleibt nicht mehr viel zu sagen. Die Aufgabe jeder PROCEDURE ist mit einem Kommentar erläutert, desgleichen die der BDOS-Aufrufe. BDOS-Aufrufe sind unter Turbo-Pascal besonders leicht zu realisieren, da diese voll unterstützt werden. Beim Procedureaufruf wird einfach die Unterprogrammnummer übergeben. Benötigt das BDOS-Unterprogramm weitere Werte, so können auch diese übergeben werden. Liefert die BDOS-Routine ein Ergebnis, so kann dies entweder im Akkumulator oder im HL-Register stehen. Soll der Akkumulator ausgewertet werden, so benutzt man ‚Ergebnis:=Bdos(nr,wert)', soll dagegen das HL-Register weiterverarbeitet werden, gilt ‚Ergebnis:=Bdoshl(nr,wert)'.

3.4. BDOS-Aufrufe

In diesem Filerprogramm wurden die BDOS-Aufrufe überall dort verwendet, wo die restlichen Pascalfunktionen nicht mehr ausreichen. Speziell das Auslesen des Inhaltsverzeichnisses einer Diskette geschieht auf diese Art. Die Procedure *Create_fcb* legt einen File-Control-Block an. Die Anfangsadresse des File-Control-Blocks muß den BDOS-Funktionen 11 bzw. 12 übergeben werden, damit CP/M weiß, nach welchem Dateinamen auf der Diskette gesucht wird. Wohin das Ergebnis dieser Suche geschrieben werden soll, wird dem System mit BDOS(26,*Zieladresse*) mitgeteilt. CP/M schreibt dabei immer einen kompletten 128-Byte-Sektor des Directories in den Puffer, der bei Zieladresse beginnt. Im A-Register liefern die Funktionen 11 oder 12 einen Wert zwischen 0 und 3, der auf den gesuchten Namen im Puffer zeigt. Dazu muß man wissen, daß jeder Name im Directory 32 Bytes belegt und somit bis zu 4 Namen im Puffer stehen können. Wird der Wert 255 geliefert, so bedeutet dies, daß der gesuchte Name nicht gefunden wurde. Mit dem Jokerzeichen ‚?' kann auch nach unbekannten Dateinamen gesucht werden – wie es auch hier geschieht. Ist der Dateiname ermittelt, kann die Datei mit den normalen Pascalprozeduren gelesen und geschrieben werden. Weiterhin wird die BDOS-Funktion 32 benutzt, die in der Lage ist, den aktuellen Userbereich anzuzeigen und zu ändern. Dabei werden Userbereiche von 0 bis 31 akzeptiert. Dies versetzt den Anwender in die Lage, Dateien in geheime Bereiche zu kopieren, die von der Tastatur aus nicht mehr erreicht werden können. Der CCP nimmt nämlich im USER-Befehl nur Argumente bis einschließlich 15 an.

4. Kompatibilität

Filer wurde auf einem Apple II+ unter CP/M 2.2 entwickelt. Es sollte jedoch ohne Änderungen auf jeden anderen CP/M-Computer übertragbar sein, da keinerlei rechnerspezifische Funktionen verwendet wurden. Ebensowenig interessieren Spur- und Sektoranzahl auf der Diskette oder die Anzahl der möglichen Directoryeinträge. Wichtig ist allein, daß das verwendete CP/M auf die vorhandene Laufwerkskonfiguration eingestellt ist. Der Anwender muß lediglich vor dem Compilieren die Konstante *Max* entsprechend der Anzahl seiner Laufwerke setzen. Selbstverständlich sollte in der Compileroption ‚O' im Turbo-Menü die Endadresse so hoch wie möglich gesetzt werden, um einen optimalen Pufferspeicher zu erhalten.

Anm. d. Bearbeiters:

1. Auch nach Compilieren mit Turbo-Pascal 3.0 funktionierte das Programm einwandfrei. Ebenso arbeitet Filer mit meinem speziellen CP/M für die Z80+-Card problemlos zusammen. Ich kann jedoch keine Aussagen darüber machen, ob es mit anderen CP/M-Versionen genauso funktioniert. Im Zweifelsfall hilft nur Ausprobieren.

2. Wie installiert man Filer?

2.1. Von der Sammeldisk: Die Peeker-Sammeldisketten sind grundsätzlich im DOS-3.3-Format beschrieben. Das heißt, daß Sie den File ‚FILER.PAS' von der Sammeldiskette zuerst mit APDOS (auf der CP/M-Master-Diskette) auf eine CP/M-Diskette übertragen müssen. Weiter unter 2.3.
2.2. Selbsteintippen: Wie gewohnt mit dem Turbo-Editor eingeben und als File ‚FILER.PAS' abspeichern.
2.3. Schließlich im Turbo-Menü die Option ‚C' aufrufen und compilieren.
2.4. Starten mit ‚FILER'.

Literatur:

Rodnay Zaks: CP/M Handbuch, Sybex-Verlag
Bernd Pol : Vom Umgang mit CP/M, IWT-Verlag
Turbo Pascal 2.0 Handbuch, Heimsoeth Software

FILER.PAS

```
program filer;

{-------------------------------------------------}
{ Dateikopierprogramm, auch fuer ein Laufwerk!    }
{ Benutzt die Erweiterungen von Turbo-Pascal 2.Ø  }
{ Laeuft unter CP/M 2.2                           }
{                                                 }
{                                                 }
{ von Gerhard Runge                               }
{                                                 }
{                                                 }
{                                                 }
{-------------------------------------------------}


const
      max = 1 ;
          {Anzahl der Laufwerke -1 A:= Ø, B:= 1 usw.}
      puffergroesse = 2Ø48;
          {Sollte ein Vielfaches der CP/M-Blockgroesse sein}
      datpuffer = 16;
          {puffergroesse/128}

type
filename      = string[12];

zeiger        = ↑pufferfeld;   {Puffer fuer einen Lese-}
pufferfeld    = record         {und Schreibzugriff}
                  datname:filename;
                  next  :zeiger;
                  sec   :integer;
                  puffer:array [1..puffergroesse] of byte;
                end;

dirzeiger     = ↑dirfeld;   {Liste aller zu kopierenden}
dirfeld       = record      {Dateien}
                  nextname:dirzeiger;
                  aktname :filename;
                end;

var
dirstart, dirweiter                 :dirzeiger;   {Pointer}
start, weiter, letzter              :zeiger;      {Pointer}
name, neuname, suchname, eingabe    :filename;
taste, laufwerk                     :char;
neufile, altfile                    :file;
lang, anzahl, userbereich, fcblock  :integer;
versatz, b, a, c, d, f, i, anf      :integer;
filepuffer                          :array[Ø..127] of byte;
fcbpuffer                           :array[Ø..35] of byte;
zuser, qusery, ziel, quelle         :byte;
raum                                :real; {freier Speicher}
fertig                              :boolean;


procedure tastendruck;
{wartet auf einen Tastendruck}
begin
  read(kbd, taste);
  taste := upcase(taste);
end;

procedure dirrelease;
{Uebergibt einen Dateinamen aus der Directory-Liste
  und gibt den vom Namen belegten Speicherplatz wieder frei}
begin
  dirweiter := dirstart;
  if dirweiter <> nil then
     begin
       dirstart := dirweiter↑.nextname;
       name := dirweiter↑.aktname;
       dispose(dirweiter);
       fertig := false;
     end
     else fertig := true;
end;

procedure release;
{simuliert die Funktionen MARK und RELEASE mittels DISPOSE}
begin
  weiter := start;
  while weiter <> nil do
      begin
        letzter := weiter↑.next;
        dispose(weiter);
        weiter := letzter
      end;
  start := nil;
end;

procedure einloggen(laufwerk:byte);
{Setzt das System zurueck und macht <laufwerk> zum aktuellen}
{Laufwerk  entspricht <CTRL-C> und fordert daher eine}
{Systemdisk in A:}
begin
  if ((laufwerk >= Ø) and (laufwerk <= max))then
    begin
      bdos(13);           {Anmelden einer Diskette}
      bdos(14,laufwerk);  {Laufwerk wird aktuelles Laufwerk}
    end
  else
    bdos(13);
end;

function  name_format(name:filename):filename;
  {xmacht aus einem beinahe beliebigem}
  {String einen erlaubten Dateinamen}
var wort   :string[3];
    dat    :filename;
begin
   if name = '' then name := '*.*';
   for a := 1 to length(name) do name[a] := upcase(name[a]);
   a := pos('.',name);
   if a < 1 then a := 9;
   wort := copy(name,a+1,3);
   b := pos('*',wort);
   if b <> Ø
      then for c := (b-1) to 2 do wort := copy(wort,1,c)+'?';
   if a > 8 then a := 9;
   dat := copy(name,1,a-1);
   a := pos('*',dat);
   if a <> Ø
      then for c := (a-1) to 7 do dat := copy(dat,1,c)+'?';
   name := dat+'.'+wort;
   for i := 1 to length(name) do
       if pos(' ',name) <> Ø then delete(name,pos(' ',name),1);
   name_format := name;
end;

procedure nameeinlesen;
{liest einen String, verwandelt ihn in fuer Dateinamen}
{gueltige Form und schreibt ihn an die alte Stelle}
begin
  write('Filename eingeben (<CR> = *.*) : ');
  read(eingabe);
  name := name_format(eingabe);
  i := length(eingabe);
  while i > Ø do
  begin
    write(#8,' ',#8);
    i := i-1;
  end;
  writeln(name);
end;

procedure speicherplatz;
{bestimmt den noch zur Verfuegung stehenden Speicherplatz}
begin
  raum := maxavail;
  if raum < Ø then raum := maxavail + 65535.Ø;
end;


procedure dir(suchname:filename);
{durchsucht das Directory nach dem angegebenen Dateinamen
  und speichert ihn, falls erwuenscht, in einer Liste }

  procedure create_fcb;
    var i,k: integer;
    {initialisiert CP/M-File-Control-Block}
    begin
      fcbpuffer[Ø] := Ø; fcbpuffer[32] := Ø;
      for i := 1 to 11 do fcbpuffer[i] := $2Ø;
      for i := 12 to 15 do fcbpuffer[i] := Ø;
      i := 1; k:= pos('.',suchname)+1;
      while suchname[i] <> '.' do
         begin
           fcbpuffer[i] := ord(suchname[i]);
           i := i+1
         end;
      for i := Ø to 2 do
          fcbpuffer[9+i] := ord(suchname[k+i]);
    end;

  procedure name_einfuegen;
    begin
      new(dirweiter);
      dirweiter↑.aktname := name;
      dirweiter↑.nextname := dirstart;
      dirstart := dirweiter;
      fertig := false;
    end;
```

```
  procedure namefragen;
     begin
       name := '';
       for i := anf+1 to anf+11 do
          name := name+chr((filepuffer[i]) and 127);
       insert('.',name,9);
       for i := 1 to 12 do
          if pos(' ',name) <> Ø
             then delete(name,pos(' ',name),1);
       write(name:14,' kopieren?          J/N? ');
       tastendruck;
       case taste of
          'J','Y',#13 : begin
                          name_einfuegen;
                          writeln('Ja');
                        end;
         else writeln('Nein');
       end;  {end of case};
     end;

begin        {dir}
  fertig := true;
  create_fcb;
  fcblock := addr(fcbpuffer);
  {fcblock = Adresse Filecontrolblock}
  versatz := bdos(26,(addr(filepuffer)));
  {Puffer fuer Diskettenoperationen angeben}
  versatz := bdos($11,fcblock);
  {Directory von Anfang an lesen}
  if taste = #27 then versatz := 255;
  if versatz = 255 then
     begin
       writeln;
       writeln('Keine Datei zu kopieren!    ');
     end;
while (versatz < $ff) do
     begin
       anf := versatz*32;
       namefragen;
       versatz := bdos($12,fcblock); {Directory weiterlesen}
       if taste = #27 then versatz := 255;
     end;
end;  {dir}


procedure gleiches_laufwerk(name:filename);

{veranlasst gegebenenfalls einen Aufruf zum Diskettenwechsel,}
{setzt das System zurueck, aktualisiert das richtige}
{Laufwerk und setzt den gewuenschten Userbereich.}

begin
  if ziel = quelle then
    begin
      write(#7);
      writeln(name,'Diskette in Laufwerk ',chr(ziel+65),
              ': einlegen und Taste druecken!');
       read(kbd,taste);
       writeln;
     end;
   if name = 'Ziel' then
      begin
        einloggen(ziel);
        userbereich := bdos(32,zuser);
        {neuen User einstellen}
      end
   else begin
        einloggen(quelle);
        userbereich := bdos(32,quser);
   end;
end;

procedure start_lesen;
{bestimmt die Laenge einer zu lesenden Datei}
begin
  assign(altfile,name);
  {$I-}
  reset(altfile);
  f := ioresult;
  {$I+}
  if f <> Ø then writeln('Fehler Quelldisk!? ',f)
        else
          begin
            lang := filesize(altfile);
            writeln('Lese: ',name);
          end;
end;
```

```
procedure lesen;
{liest die durch <puffergroesse> bestimmte Anzahl von Bytes
 (oder mindestens den Rest) der Datei in den Speicher.}
begin
  if datpuffer <= lang then anzahl := datpuffer
                       else anzahl := lang;
new(weiter);
blockread(altfile,weiter↑.puffer,anzahl);
weiter↑.sec := anzahl;
weiter↑.datname := name;
lang := lang-anzahl;
if start = nil then start := weiter
               else letzter↑.next := weiter;
letzter := weiter;
letzter↑.next := nil;
speicherplatz;
end;

procedure neue_datei;
{eroeffnet eine neue Datei zum Schreiben.
  Falls der Dateiname schon existiert, kann ein neuer Name
  angegeben werden. <CR> ueberschreibt die alte Datei
  gleichen Namens.}
begin
  suchname := weiter↑.datname;
  repeat
    assign(neufile,suchname);
    {$I-}
    reset(neufile);
    f := ioresult;
    {$I+}
    if f = Ø then
      begin
        writeln(suchname,' bereits vorhanden.  ');
        write(#7,'Neuen Filenamen eingeben oder Diskette');
        write(#7,' wechseln (<CR> ueberschreibt ');
        writeln(#7,suchname,'):');
        readln(neuname);
        if length(neuname) > Ø
           then neuname := name_format(neuname);
        f := 1;
        if length(neuname) > 1 then suchname := neuname;
        einloggen(ziel);
      end;
    if f > 1 then
      begin
        writeln('IO-Fehler ',f);
        {Kann nur bei Hardwarefehlern vorkommen}
        halt;
      end;
  until f = 1;
  assign(neufile,suchname);
  rewrite(neufile);
  writeln('Schreibe: ',suchname);
  suchname := weiter↑.datname;
end;

procedure copy_dat;
{Der Programmteil, welcher wirklich kopiert.}
begin
  suchname := ' @anfang@';
  {Kann in Wirklichkeit nicht vorkommen, markiert den Anfang}
  lang := Ø;
  repeat
     speicherplatz;
     while((raum > (datpuffer*128.Ø)) and (fertig = false)) do
       begin
         if lang = Ø then
             begin
               dirrelease;
               if fertig = false then start_lesen
             end;
         while ((lang>Ø) and (raum>(datpuffer*128.Ø))) do lesen;
       end;
     gleiches_laufwerk('Ziel');
     if suchname <> ' @anfang@' then
       begin
         reset(neufile);
         seek(neufile,filesize(neufile));
       end;
     weiter := start;
     while weiter <> nil do
       begin
         if suchname = ' @anfang@' then neue_datei;
         if (suchname <> weiter↑.datname) then
            begin
              close(neufile);
              neue_datei;
            end;
```

```
        blockwrite(neufile,weiter↑.puffer,weiter↑.sec);
        weiter := weiter↑.next;
      end;
    close(neufile);
    release;
    if fertig = false then gleiches_laufwerk('Quell');
  until fertig = true;
  close(neufile);
end;

procedure userfragen;
{erfragt die gewuenschten Userbereiche fuer Ziel und Quelle}
begin
  write('Userbereich? ':2Ø);
  read(eingabe);
  val(eingabe,userbereich,c);
  if c = Ø then case userbereich of
           Ø..31 : ;
           else  userbereich := Ø;
           end   {end of case}
       else userbereich := Ø;
  i := length(eingabe);
  while i > Ø do
     begin
       write(#8,' ',#8);
       i := i-1;
     end;
  writeln(userbereich);
end;

begin                                  {Hauptprogramm}
  dirstart := nil;
  start := nil;
  ziel := max;
  quelle := Ø;
  repeat                               {Hauptschleife}
    laufwerk := chr(bdos(25)+65);      {aktuelle Laufwerk}
    userbereich := bdos(32,$ff);       {aktuelle User}
    fertig := true;
    clrscr;
    speicherplatz;
    writeln('Kopierprogramm fuer Files:    CP/M 2.2',
           '3.8.85':19,'G.Runge':22);
    write('Aktives Laufwerk ',laufwerk,':');
    write('Userbereich: ':24,userbereich);
    writeln(raum:18:Ø,' Bytes verfuegbar');
    write  ('In Laufwerk A: muss sich immer eine ');
    writeln('Systemdiskette befinden!');
    for i := 1 to 8Ø do write('-');
    if raum < ((128.Ø * datpuffer)+13ØØ) then
       begin
          writeln('Programmabbruch: zu wenig Speicher!');
          halt;
       end;
    repeat
      write('Quelle:     Welches Laufwerk? ');
      tastendruck;
      if ord(taste) > 64 then quelle := ord(taste)-65;
      write(chr(quelle+65),': ');
      userfragen;
      quser := userbereich;
      write('Ziel  :     Welches Laufwerk? ');
      tastendruck;
      if ord (taste) > 64 then Ziel := ord(taste)-65;
      write(chr(ziel+65),': ');
      userfragen;
      zuser := userbereich;
    until((quelle >= Ø) and (quelle <= max))
         and ((ziel >= Ø) and(ziel <= max));
    if quelle <> ziel then writeln('Disketten einlegen! ');
    gotoxy(66,3); write('<ESC> = Abbruch') ;gotoxy(1,8);
    gleiches_laufwerk('Quell');
    writeln;
    if taste <> #27 then
       begin
         nameeinlesen;
         dir(name)
       end
       else while fertig = false do dirrelease;
    if fertig = false then copy_dat;
    writeln;
    write('Programmende mit <ESC> ');
    tastendruck;
  until taste = #27;
  userbereich := bdos(32,Ø); {immer zurueck in Userbereich Ø}
end.
```

PEEKER Börse

Gelegenheitsanzeigen

Sie können unter dieser Rubrik zu einem besonders günstigen Preis

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

Musteranzeige privat

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

Beispiel:

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

nur DM 18,81 inkl. MwSt.

Musteranzeige gewerblich

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

Beispiel:

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

nur DM 62,70 inkl. MwSt.

PEEKER

Peeker-Börse

AUFTRAG FÜR KLEINANZEIGEN

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

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

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

PEEKER

Produkt-Karte

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

Anzeige über _____

_____

bitte ich um detaillierte Information.

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

Menge	Produkt und Bestellnummer	à DM	gesamt DM

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

_____

Unterschrift (für Jugendliche unter 18 Jahren der Erziehungsberechtigte)

PEEKER

Info-Karte

_____

_____

_____

_____

_____

_____

_____

Ich interessiere mich für Beiträge über

☐ Apple ☐ Mac ☐ 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

Umfrage-Karte

Karte bitte vollständig ausfüllen

Vorname, Name

Firma

Straße

PLZ/Ort

Telefon mit Vorwahl

POSTKARTE

Peeker-Börse
Anzeigen-Service

Dr. Alfred Hüthig Verlag

Postfach 10 28 69

6900 Heidelberg 1

POSTKARTE

Inserent

Straße

PLZ/Ort

POSTKARTE

Peeker
Redaktion

Dr. Alfred Hüthig Verlag

Postfach 10 28 69

6900 Heidelberg 1

Produkt-Karte

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

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

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

Damit erleichtern Sie dem Hersteller eine gezielte Beantwortung Ihrer Anfrage

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

PEEKER

...und plötzlich steht die Polizei im Kinderzimmer!

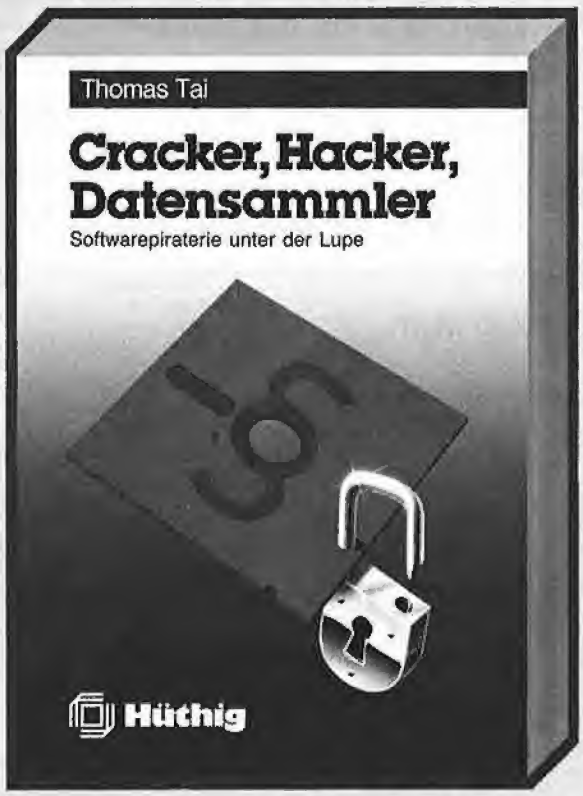
Raubkopierer gehen einem gefährlichen Hobby nach. Sie fühlen sich als die Robin Hood's der Computerszene, aber sie richten Millionenschaden an!

Thomas Tai kennt die Kopierszene. Er weiß, warum kopiert wird. Und was die Software-Häuser dagegen tun. Oft endet das, was als vermeintlich harmloses Hobby anfing, mit Hausdurchsuchung und Gerichtsverfahren.

Er schrieb eine Mahnung an alle Raubkopierer auszusteigen. An authentischen Fällen zeigt er die rechtliche Situation. Sagt, was bei einer Abmahnung oder Schlimmerem zu tun ist. Denn die „Gegenseite" reagiert oft übertrieben und stellt astronomische Schadensersatzforderungen.

Jeder, der einen Computer hat, kann mit Raubkopien in Verbindung kommen. Deshalb ist dieses Buch für alle wichtig. Auch für Eltern, die sich wundern, warum Klaus-Peter neuerdings so viel Post bekommt.

Das Buch ist brandneu und kostet 24,– DM.



Diese Buchhandlungen haben das Buch vorrätig!

Hüthig

Dreiecksberechnungen

in UCSD-Pascal

von Samuel Schmid

Das hier vorgestellte Pascal-Programm erlaubt dem Benutzer, Seiten, Winkel, Flächen und andere Daten von beliebigen Dreiecken schnell und einfach zu berechnen und die Dreiecke anschließend auf dem Bildschirm grafisch darzustellen. Die Berechnung der Dreiecke erfolgt über die vier allgemein bekannten Kongruenzsätze in der Art „...Dreiecke sind kongruent, wenn...".

1. Bedienung

Nach dem Programmstart erscheint das Menü und fragt den Benutzer, nach welchem Kongruenzsatz das Dreieck berechnet werden soll. Will man z.B. ein Dreieck berechnen, von dem alle drei Seiten bekannt sind, so gibt man „1" für den ersten Kongruenzsatz (Seite – Seite – Seite) ein. Danach wartet das Programm auf die Eingabe der entsprechenden Daten, welche in einem beliebigen einheitlichen Maß eingegeben werden. War die Eingabe korrekt (keine der Eingaben darf 0 sein), so erscheinen kurz darauf die berechneten Daten.
Will man das Dreieck sehen, so gibt man auf die Frage, ob das Dreieck grafisch dargestellt werden soll, „J" für „Ja" ein. Das Dreieck erscheint nun auf dem Bildschirm. Die Eckpunkte sind mit den dazugehörigen Buchstaben bezeichnet. Das Dreieck wird nicht mit den eingegebenen oder berechneten Seitenlängen dargestellt. Lediglich die Verhältnisse stimmen, das dargestellte ist also dem berechneten Dreieck ähnlich. Dies ist deshalb der Fall, da die Dreiecke so vergrößert oder verkleinert werden, daß sie auf den Bildschirm passen. Ließ man das Dreieck grafisch darstellen, so hat man jetzt die Möglichkeit, durch Drücken der Tasten „T" und „G" zwischen Text- und Grafikbildschirm hin- und herzuschalten oder mit „E" wieder zum Menü zu gelangen. Das Programm beginnt immer wieder von vorn. Es kann jedoch durch die Eingabe von „0" im Menü verlassen werden.
Das Programm wird im Editor des Pascal-Systems eingegeben. Ist die Eingabe beendet, so verläßt man den Editor und compiliert das Programm. Gestartet wird das Programm entweder mittels „R" für „Run" oder mittels „X" für „Execute".

2. Änderungen

Wer das Programm in eine andere Pascal-Version umschreiben oder erweitern will, sollte folgendes beachten:
– Für eine Grafik-Dump-Routine kann man beispielsweise die Prozedur „toggle" (= „hin- und herschalten") um einen Befehl erweitern, der die Druckroutine, welche vorzugsweise in Form einer „Case"-Anweisung oder einer Prozedur geschrieben wird, aufruft.
– Die im Programm immer wieder auftretende Konstante „dg" wird benötigt, um die durch den Computer in Radiant (2 * Pi) berechneten Winkelfunktionen in die allgemein verwendeten Altgrade (Kreiswinkel = 360 Grad), welche sich auch für die grafische Darstellung besser eignen, umzurechnen. Will man weitere Berechnungen in die Sätze einfügen, so muß diese Konstante in alle Formeln, welche den Sinus, Cosinus oder Arcustangens benötigen, eingefügt werden. Die durch „asin", „acos" oder „tan" gelieferten Werte sind schon in Altgrade umgerechnet.
– Die Variablen „s0", „s1", und „s2" werden verwendet, um das Dreieck so zu vergrößern oder zu verkleinern, daß es auf den Bildschirm paßt.
– Das Programm wurde ohne die Verwendung des Goto-Befehls realisiert. Wird jedoch dieser Befehl in einer Erweiterung oder Abänderung des Programmes benötigt, so muß im Programmkopf die Compiler-Anweisung „(*$G+*)" stehen.
– Eine schlechte Eigenschaft von Pascal 1.1 und 1.2 ist es, daß eine Eingabeanweisung der Form „readln" mit einer realen Variablen als Argument bei fehlerhafter Eingabe nicht korrigierbar ist und bei Eingabe eines Buchstabens sogar das ganze Programm abbricht. Eine Möglichkeit wäre also, die Eingabe mittels des „ioresult"-Befehls durch das Programm selbst zu prüfen und gegebenenfalls eine Fehlermeldung auszugeben.

DREIECK.TEXT

```
PROGRAM dreiecksberechnungen;
{von Samuel Schmid, Bern, 1985}
uses TURTLEGRAPHICS, TRANSCEND;


Const   dg                                      = 57.29577951;
Var     s1,s2                                   : integer;
        ch                                      : char;
        a,b,c,al,be,ga,ha,fl,ir,ur,ar,d,s,s0    : real;

FUNCTION asin(x: real): real;  {asin=arcussinus}
Begin
  if abs(x)=1 then asin:=90*x; {asin(1)=90, asin(-1)=-90}
    if abs(x)<1 then asin:=atan(x/sqrt(-x*x+1))*dg;
End {asin};

FUNCTION acos(x: real): real;  {acos=arcuscosinus}
Begin
  if abs(x)<1 then acos:=(-atan(x/sqrt(-x*x+1))+1.5707633)*dg
    else acos:=0;
    if x=-1 then acos:=180;
End {acos};
```

```
FUNCTION tan(x: real): real;
Begin
  if (cos(x/dg)<>0) then tan:=(sin(x/dg)/cos(x/dg));
End {tan};

PROCEDURE satz1; {Seite Seite Seite}
Begin
  write(chr(12)); {Bildschirm loeschen}
  gotoxy(30,2);  writeln('SSS');
    repeat
      gotoxy(0,7);
        write('Seite a :'); readln(a);
        write('Seite b :'); readln(b);
        write('Seite c :'); readln(c);
      a:=abs(a); b:=abs(b); c:=abs(c);
    until ((a+b>=c) and (a>0) and (b>0) and (c>0));
        s:=(a+b+c)/2; {s=halber Dreiecksumfang}
        al:=2*acos(sqrt(s*(s-a)/(b*c)));
        be:=asin(sin(al/dg)*b/a);
        ga:=180-al-be;
        fl:=a*b/2*sin(ga/dg);
        ha:=fl*2/a;
        ir:=fl/s;
        ur:=(a*b*c)/(4*fl);
        ar:=fl/(s-a);
End {satz1};
```

```
PROCEDURE satz2; {Winkel Seite Winkel}
Begin
  write(chr(12)); gotoxy(30,2); writeln('WSW');
    repeat
      gotoxy(0,7);
        write('Alpha :');    readln(al);
        write('Seite b :');  readln(b);
        write('Gamma :'); readln(ga);
      ga:=abs(ga); al:=abs(al); b:=abs(b);
    until ((ga>0) and (al>0) and (b>0));
        be:=180-al-ga;
        a:=b*sin(al/dg)/sin(be/dg);
        c:=b*sin(ga/dg)/sin(be/dg);
        fl:=a*b/2*sin(ga/dg);
        ha:=fl*2/a;
        s:=(a+b+c)/2;
        ir:=fl/s;
        ur:=(a*b*c)/(4*fl);
        ar:=fl/(s-a);
End {satz2};

PROCEDURE satz3; {Seite Winkel Seite}
Begin
  write(chr(12)); gotoxy(30,2); writeln('SWS');
    repeat
      gotoxy(0,7);
        write('Seite c :'); readln(c);
        write('Alpha :');   readln(al);
        write('Seite b :'); readln(b);
      c:=abs(c); al:=abs(al); b:=abs(b);
    until ((c>0) and (al>0) and (b>0));
        a:=sqrt(sqr(b)+sqr(c)-2*b*c*cos(al/dg));
        be:=asin(b*sin(al/dg)/a);
        ga:=180-al-be;
        fl:=a*b*sin(ga/dg)/2;
        ha:=fl*2/a;
        s:=(a+b+c)/2;
        ir:=fl/s;
        ur:=(a*b*c)/(4*fl);
        ar:=fl/(s-a);
End {satz3};

PROCEDURE satz4; {Seite Seite Winkel}
Begin
  write(chr(12)); gotoxy(30,2); writeln('SSW');
    repeat
      gotoxy(0,7);
        writeln('Seite b > Seite a');
        write('Seite b :'); readln(b);
        write('Seite a :'); readln(a);
        write('Beta :');    readln(be);
      b:=abs(b); a:=abs(a); be:=abs(be);
    until ((b>a) and (b>0) and (a>0) and (be>0));
        al:=asin(a*sin(be/dg)/b);
        ga:=180-al-be;
        c:=b*sin(ga/dg)/sin(be/dg);
        s:=(a+b+c)/2;
        fl:=a*b/2*sin(ga/dg);
        ha:=fl*2/a;
        ir:=fl/s;
        ur:=(a*b*c)/(4*fl);
        ar:=fl/(s-a);
End {satz4};

PROCEDURE toggle; {Wechseln zwischen Text und Grafik}
Begin
  repeat
    repeat
      gotoxy(0,23); read(ch);
    until ch in ['t','T','g','G','e','E'];
      case ch of
        't','T': textmode;
        'g','G': grafmode;
        'e','E': exit(toggle);
      end {case};
  until false {Endlosschleife}
End {toggle};

PROCEDURE runde; {rundet die Winkel}
Begin
  if ga-trunc(ga)>0.5 then ga:=trunc(ga+0.5);
  if al-trunc(al)>0.5 then al:=trunc(al+0.5);
  if be-trunc(be)>0.5 then be:=trunc(be+0.5);
End {runde};

PROCEDURE grafout;
Begin
  initturtle; s1:=1; s2:=1;
  chartype(10); {weisse Buchstaben, schwarzer Grund}
    if ga<=90 then begin
      repeat {Dreieck vergroessern}
        if (a*s2<250) and (ha*s2<170) then s2:=s2+1;
      until ((a*(s2+1)>=250) or (ha*(s2+1)>=170));
      repeat {Dreieck verkleinern}
        if (a*s2/s1>250) or (ha*s2/s1>170) then s1:=s1+1;
      until ((a*s2/s1<=250) and (ha*s2/s1<=170));
    end {if ga}
      else begin
        d:=sqrt(sqr(b)-sqr(ha))+a;
        repeat {Dreieck vergroessern}
          if (d*s2<250) and (ha*s2<170) then s2:=s2+1;
        until ((d*(s2+1)>=250) or (ha*(s2+1)>=170));
        repeat {Dreieck verkleinern}
          if (d*s2/s1>250) or (ha*s2/s1>170) then s1:=s1+1;
        until ((d*s2/s1<=250) and (ha*s2/s1<=170));
      end {else};
    s0:=s1/s2; moveto(10,10); pencolor(white);
      moveto(10+trunc(a/s0),10);
      turn(180-trunc(ga)); move(trunc(b/s0)); moveto(10,10);
        pencolor(none); moveto(8,1);              wchar('B');
        moveto(8+trunc(a/s0),1);                  wchar('C');
        moveto(12+trunc(a/s0),10); turnto(0);
        turn(180-trunc(ga)); move(trunc(b/s0));   wchar('A');
      toggle;
End {grafout};

PROCEDURE textout;
Begin
  write(chr(12)); gotoxy(30,3); writeln('ERGEBNISSE');
  gotoxy(0,7);
    {Daten werden mit zwei Dezimalstellen ausgegeben}
    writeln('Seite a=',a:5:2);
    writeln('Seite b=',b:5:2);
    writeln('Seite c=',c:5:2);
    writeln('Winkel alpha=',al:5:2);
    writeln('Winkel beta=',be:5:2);
    writeln('Winkel gamma=',ga:5:2);
    writeln('Hoehe auf a=',ha:5:2);
    writeln('Flaeche=',fl:5:2);
    writeln('Inkreisradius=',ir:5:2);
    writeln('Umkreisradius=',ur:5:2);
    writeln('Ankreisradius=',ar:5:2);
    writeln;
    write('Soll das Dreieck grafisch dargestellt werden (J/N)');
    read(ch); if ch in ['j','J'] then grafout;
End {textout};

PROCEDURE dreiecke;
Begin
  write(chr(12));
    gotoxy(25,2); writeln('DREIECKSBERECHNUNGEN');
    gotoxy(20,3); writeln('von Samuel Schmid, Bern 1985');
      gotoxy(30,8);  writeln('(1) SSS');
      gotoxy(30,9);  writeln('(2) WSW');
      gotoxy(30,10); writeln('(3) SWS');
      gotoxy(30,11); writeln('(4) SSW');
        gotoxy(25,15); write('Eingabe : (0=Ende)');
          repeat
            read(ch);
          until ch in ['0'..'4'];
            case ch of
              '0': exit(program);
              '1': satz1;
              '2': satz2;
              '3': satz3;
              '4': satz4;
            end {case};
          textout; textmode;
End {dreiecke};

BEGIN {hauptprogramm}
  repeat
    dreiecke;
  until false {Endlosschleife}
END {hauptprogramm}.
```

Hinweis: Für einen geordneten Exit Endlosschleife durch kleines Menü ersetzen.

Stringbefehle für Kyan-Pascal

von Matthias Meyer

1. Probleme der Parameterübergabe

Im Peeker 3/86, S.40, konnten Sie nachlesen, daß es um die Stringverarbeitung in Kyan-Pascal schlecht bestellt ist. Im Gegensatz zu UCSD- und Turbo-Pascal gibt es nämlich in Kyan-Pascal leider keinen vordefinierten Datentyp STRING, so daß man bisher mit einer einmal festgelegten und nicht mehr zu ändernden Länge MAXSTRING leben mußte.
Unglücklicherweise waren die mitgelieferten Befehle zur Stringverarbeitung auch noch in Pascal geschrieben, was das ansonsten effiziente Kyan-Pascal in Sachen Textverarbeitung zum „Fat- and Slow-Produkt" werden ließ. Diesem Mißstand soll nun mit sechs neuen Befehlen abgeholfen werden: LENGTH, POS, CONCAT, COPY, DELETE und INSERT, die in Anlehnung an (1) entwickelt wurden.

Doch bevor es soweit war, sollten erst einmal noch einige Hürden zu überwinden sein. Zunächst stellte es sich als unmöglich heraus, komplexe Datentypen (ARRAY OF CHAR) mit unterschiedlichem Aufbau (verschiedene Stringlängen) an dieselbe Prozedur oder Funktion zu übergeben. Da es jedoch erfreulicherweise in Pascal möglich ist, an eine Prozedur oder Funktion anstelle einer Variablen deren Speicheradresse zu übergeben (2), schien dieses Problem damit eigentlich schon gelöst zu sein. Doch leider stellte sich heraus, daß diese Übergabe aus unerfindlichen Gründen nur dann erfolgen darf, wenn der Datentyp der Variablen, auf die sich der Variablenzeiger richten soll, explizit in der Prozedur oder Funktion angegeben wird, obwohl dieser zur Bestimmung des Variablenzeigers völlig uninteressant ist. Aber genau hier war dann doch noch die undichte Stelle im System:

Wenn es schon nicht möglich sein sollte, den ganzen String an die Prozedur oder Funktion zu übergeben, so doch wenigstens das erste Zeichen. Wenn man in der Prozedur dann anstelle des ersten Zeichens dessen Adresse anfordert, hat man automatisch die Anfangsadresse des ganzen Strings. Einen Haken hat die Sache aber: Die Länge der Zeichenkette ist unbekannt. Deshalb muß nun zu jeder Stringoperation auch noch ein Längenbyte übergeben werden.

Wir fassen also zusammen:
– Zur Übergabe eines Strings an einen Stringbefehl muß dessen vordefinierte Länge bekannt sein und übergeben werden.
– Anstelle des ganzen Strings wird nur das erste Zeichen übergeben. Das erste Zeichen eines Strings fordert man an, indem man direkt nach dem Variablennamen den Index [1] schreibt.

Die neuen Stringbefehle wurden so weit wie möglich an die Syntax der UCSD-Pascal-Befehle angelehnt. Leider war es aus verschiedenen Gründen nicht möglich, die Funktionen CONCAT und COPY als ebensolche zu implementieren. Schon ein einfacher Versuch, eine String-Variable konstanter Länge als Funktionswert zurückzubekommen, endete mit einem nicht dokumentierten „47 error" des Compilers. Offensichtlich sind bei Kyan-Pascal komplexe Variablen als Funktionsrückgabeparameter nicht vorgesehen, denn der von Kyan Software mitgelieferte, in Pascal geschriebene CONCAT-Befehl ist auch als Prozedur anstelle einer Funktion implementiert worden.

2. Die implementierten Befehle*

Um die nun folgenden Beschreibungen der einzelnen Befehle möglichst kurz zu halten, folgen hier zunächst einige Definitionen:

– Ein String ist in Kyan-Pascal ein ARRAY [1..n] OF CHAR und somit kein vordefinierter Datentyp.
– Die Übergabe einen Stings an einen Stringbefehl erfolgt, wenn nicht anders vermerkt, durch die Übergabe der ersten Zeichens des Strings.
– Wenn nachfogend von Werten gesprochen wird, so ist damit eine INTEGER-Variable bzw. eine INTEGER-Konstante gemeint.
– Unter der Maximallänge eines Strings verstehen wir einen Wert, der die maximale Anzahl der Zeichen, die der String aufnehmen kann, angibt.
– Mit der Länge eines Strings hingegen ist ein Wert gemeint, der die Anzahl der zu verarbeitenden Zeichen des Strings angibt.

LENGTH-Funktion

Sie ermittelt die Länge der String-Eintragung eines Übergabestrings.
Eingabe: Die Funktion erwartet als Übergabeparameter die Maximallänge des Strings, dann den String selbst.
Ausgabe: Die Funktion verhält sich im Programm wie eine INTEGER-Variable.

POS-Funktion

Sie ermittelt die Position, ab der ein String S1 in einem String S2 vollständig enthalten ist.
Eingabe: Die Funktion erwartet als Übergabeparameter die Länge von S1, gefolgt von S1, dann die Maximallänge von S2, gefolgt von S2.
Ausgabe: Wie bei LENGTH.

Prozedur CONCAT

Sie erzeugt einen String S3, der aus zwei Strings S1 und S2 zusammengesetzt wird.
Eingabe: Die Prozedur erwartet als Übergabeparameter die Länge von S1, gefolgt von S1, dann die Länge von S2, gefolgt von S2, dann die Maximallänge von S3, gefolgt von S3.
Ausgabe: Die Prozedur wird im Programm als eigenständiger Befehl aufgerufen und

* Zum besseren Verständnis kann man die Stringdemo-Zählleisten aus Peeker 3/1986, S. 43-44 heranziehen.

überschreibt den Ausgabestring vollständig. Nicht benutzte Bytes werden mit Leerzeichen aufgefüllt.

Prozedur COPY
Sie erzeugt einen String S2, der einen Teilstring von S1 darstellt.
Eingabe: Die Prozedur erwartet als Übergabeparameter einen Wert mit der Anzahl der zu kopierenden Zeichen, gefolgt von dem Zeichen von S1, ab welchem kopiert werden soll, dann die Maximallänge von S2, gefolgt von S2.
Ausgabe: S2 darf mit S1 identisch sein, sonst wie bei CONCAT.

Prozedur DELETE
Sie löscht einen Teil aus einem String S. Der verbleibende Stringrest wird so verschoben, daß die verbleibenden zwei Stringteile direkt aufeinander folgen.
Eingabe: Die Prozedur erwartet als Übergabeparameter die Maximallänge von S, dann einen Wert mit der Position, ab welcher gelöscht werden soll, gefolgt von einem Wert mit der Anzahl der zu löschender Zeichen, schließlich den String S.
Ausgabe: Wie bei CONCAT.

Prozedur INSERT
Sie fügt einen Teil aus einem String S1 in einen String S2 ab einer angegebenen Position ein. Der verbleibende Stringrest von S2 wird so verschoben, daß er an die eingefügten Zeichen aus String S1 direkt anschließt.
Eingabe: Die Prozedur erwartet als Übergabeparameter die Maximallänge von S2, dann einen Wert mit der Position, ab welcher S1 in S2 eingefügt werden soll, gefolgt von einem Wert mit der Anzahl einzufügender Zeichen, nachfolgend das Zeichen von S1, ab welchem kopiert werden soll, schließlich den String S2.
Ausgabe: Wie bei CONCAT.

Zwei abschließende Tips

– Um die Compilierung zu verkürzen, ist es empfehlenswert, den Include-File direkt in das Quellprogramm einzufügen, da die vielen Directory-Zugriffe bei Verwendung mehrerer Include-Files die Compilierung doch recht eindrücklich verlangsamen.
– Beim Eintippen der Assembler-Quellprogramme sind Verwechslungen der temporären Variablen _T+n usw. sehr leicht möglich. Bitte besonders beachten.

Literatur

(1) Apple Pascal Language Reference Manual, S. 22-25.
(2) Kyan Pascal User's Manual Version 2.0, S. V-18 ff.

Kurzhinweise

1. Zweck:
Schnelle Assemblerroutinen für Stringbefehle in Kyan-Pascal 2.0.
2. Konfiguration:
II+/e/c; ProDOS; Kyan-Pascal ab Version 2.0 aufwärts.
3. Compiliertes Programm STRINGDEMOS aus Kyan-Menü heraus starten.
4. Sammeldisk:
STRINGUTILS.I
(Include-File mit sechs neuen Stringbefehlen)
STRINGDEMOS.P
(Pascal-Quelltext mit Anwendungsbeispielen)
5. Sonstiges:
Diese Dateien müssen zunächst mit CONVERT oder DOSTOPRO von der DOS-3.3-Sammeldisk auf Ihre Kyan-ProDOS-Arbeitsdiskette konvertiert werden. Danach STRINGDEMOS.P zu STRINGDEMOS compilieren.

STRINGUTILS.I

```
FUNCTION LENGTH(L:INTEGER;VAR S:CHAR):INTEGER;
BEGIN
LENGTH:=Ø;
#A
        LDY  #7
        LDA  (_SP),Y
        STA  _T          ;S-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+1        ;S-Zeiger.H
        INY
        LDA  (_SP),Y
        TAY              ;Länge von S
        DEY              ;Index auf String
        LDA  #$2Ø        ;Space
FLLOOP  CMP  (_T),Y      ;Vergleiche Zeichen
        BNE  FLEXIT      ;Kein Space -> Fertig
        DEY              ;Weitertesten
        BPL  FLLOOP      ;bis Stringanfang
FLEXIT  INY              ;LENGTH zählt ab 1
        TYA
        LDY  #5
        STA  (_SP),Y     ;Funktionswert
#
END;


FUNCTION POS(L1:INTEGER; VAR S1:CHAR;
             L2:INTEGER; VAR S2:CHAR):INTEGER;
BEGIN
POS:=Ø;
#A
        STX  _T
;
        LDY  #7
        LDA  (_SP),Y
        STA  _T+1        ;S2-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+2        ;S2-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+3        ;Länge von S2
;
        LDY  #11
        LDA  (_SP),Y
        STA  _T+4        ;S1-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+5        ;S1-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+6        ;Länge von S1
;
        SEC
        LDA  _T+3        ;Länge von S2
        SBC  _T+6        ;Länge von S1
        BCC  PEXIT1      ;Fehler, wenn L1>L2.
        STA  _T+3        ;Testlänge von S2
        INC  _T+3        ;+1 -Korrektur
;
        LDX  #Ø          ;Zähler für POS -1
PLOOPØ  LDY  #Ø          ;Zähler für
PLOOP1  LDA  (_T+4),Y    ;Zeichen aus S1
        CMP  (_T+1),Y    ;Zeichen aus S2
        BNE  PNEXT1
        INY
        CPY  _T+6        ;Länge von S1
        BCC  PLOOP1
        BCS  PDONE1
;
PNEXT1  INX
        CPX  _T+3        ;Länge von S2
        BCS  PEXIT1      ;fertig, wenn X>=L2.
        INC  _T+1        ;S1-Zeiger.L
        BNE  PLOOPØ
        INC  _T+2        ;S1-Zeiger.H
        BNE  PLOOPØ      ;Verweige immer.
;
PDONE1  INX              ;POS-Werte von 1 an
        TXA
        LDY  #5
        STA  (_SP),Y     ;Funktionswert
;
PEXIT1  LDX  _T
#
END;
```

```
PROCEDURE CONCAT(L1:INTEGER; VAR S1:CHAR;
                 L2:INTEGER; VAR S2:CHAR;
                 L3:INTEGER; VAR S3:CHAR);
BEGIN
#A
        STX  _T
.
,
        LDY  #5
        LDA  (_SP),Y
        STA  _T+1        ;S3-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+2        ;S3-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+3        ;Länge von S3
;
        LDY  #9
        LDA  (_SP),Y
        STA  _T+4        ;S2-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+5        ;S2-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+6        ;Länge von S2
;
        LDY  #13
        LDA  (_SP),Y
        STA  _T+7        ;S1-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+8        ;S1-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+9        ;Länge von S3
;
        LDY  #Ø
CLOOP1  LDA  (_T+7),Y    ;Von S1 nach
        STA  (_T+1),Y    ;S3 kopieren.
        INY
        CPY  _T+9        ;Länge von S1
        BCS  C1DONE
        CPY  _T+3        ;Länge von S3
        BCC  CLOOP1
;
C1DONE  SEC
        LDA  _T+3        ;Länge von S3
        SBC  _T+9        ;Länge von S1
        BCC  C2DONE      ;fertig, wenn L1>=L3.
        BEQ  C2DONE
        STA  _T+1Ø       ;Restlänge von S3
;
        CLC
        LDA  _T+1        ;S3-Zeiger.L
        ADC  _T+9        ;Länge von S1
        STA  _T+1
        LDA  _T+2        ;S3-Zeiger.H
        ADC  #Ø          ;Carry-Flag
        STA  _T+2
;
        LDY  #Ø
CLOOP2  LDA  (_T+4),Y    ;Von S2 nach
        STA  (_T+1),Y    ;S3 kopieren
        INY
        CPY  _T+6        ;Länge von S2
        BCS  C2DONE
        CPY  _T+1Ø       ;Restlänge von S3
        BCC  CLOOP2
;
C2DONE  SEC
        LDA  _T+1Ø       ;Restlänge von S3
        SBC  _T+6        ;Länge von S2
        BCC  C3DONE      ;fertig, wenn S2>=
        BEQ  C3DONE      ;Restlänge von S3.
        STA  _T+1Ø       ;Neue Restlänge S3
;
        LDA  #$2Ø        ;Space
        LDX  #Ø
CLOOP3  STA  (_T+1),Y
        INY
        INX
        CPX  _T+1Ø       ;Neue Restlänge S3
        BCC  CLOOP3
;
C3DONE  LDX  _T
#
END;
```

```
PROCEDURE DELETE(L1,L2,L3:INTEGER; VAR S:CHAR);
BEGIN
#A
        LDY  #5
        LDA  (_SP),Y
        STA  _T+1       ;S-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+2       ;S-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+3       ;Länge für DELETE
        LDY  #9
        LDA  (_SP),Y
        STA  _T+4       ;Position für DELETE
        DEC  _T+4  .    ;-1 Korrektur
        LDY  #11
        LDA  (_SP),Y
        STA  _T+5       ;Länge von S
;
        SEC
        SBC  _T+4       ;Position für DELETE
        BCC  DEXIT1     ;fertig, wenn >=L.
        BEQ  DEXIT1
;
        STA  _T+6       ;Restlänge von S
;
        CLC
        LDA  _T+1       ;S-Zeiger.L
        ADC  _T+4       ;Position für DELETE
        STA  _T+1
        LDA  _T+2       ;S-Zeiger.H
        ADC  #Ø         ;Carry-Flag
        STA  _T+2
;
        SEC
        LDA  _T+6       ;Restlänge von S
        SBC  _T+3       ;Länge für DELETE
        BCC  DOVFL1     ;Überlauf
        BEQ  DOVFL1
        STA  _T+7       ;Restlänge DELETE
;
        CLC

PROCEDURE COPY(L1:INTEGER; VAR S1:CHAR;
               L2:INTEGER; VAR S2:CHAR);
BEGIN
#A
        LDY  #5
        LDA  (_SP),Y
        STA  _T+1       ;S2-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+2       ;S2-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+3       ;Länge von S2
;
        LDY  #9
        LDA  (_SP),Y
        STA  _T+4       ;S1-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+5       ;S1-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+6       ;Länge von S1
;
        LDY  #Ø
CPLOOP  LDA  (_T+4),Y   ;Zeichen aus S1
        STA  (_T+1),Y   ;nach S2 kopieren
        INY
        CPY  _T+6       ;Länge von S1
        BCS  CPDONE
        CPY  _T+3       ;Länge von S2
        BCC  CPLOOP
;
CPDONE  LDA  _T+6       ;Länge von S1
        CMP  _T+3       ;Länge von S2
        BCS  CPEXIT     ;fertig, wenn >=S2.
;
        LDA  #$2Ø       ;Space
CPFILL  STA  (_T+1),Y   ;S2 auffüllen
        INY
        CPY  _T+3       ;Länge von S2
        BCC  CPFILL
;
CPEXIT  NOP
#
END;
```

```
        LDA  _T+1       ;S-Zeiger.L
        ADC  _T+3       ;Länge für DELETE
        STA  _T+8       ;S-Zeiger danach.L
        LDA  _T+2       ;S-Zeiger.H
        ADC  #Ø         ;Carry
        STA  _T+9       ;S-Zeiger danach.H
;
        LDY  #Ø
DCOPY1  LDA  (_T+8),Y   ;S-Zeichen danach
        STA  (_T+1),Y   ;S-Zeichen davor
        INY
        CPY  _T+7       ;Restlänge DELETE
        BCC  DCOPY1
;
DFILLØ  LDA  #$2Ø       ;Space
DFILL1  STA  (_T+1),Y   ;S-Leerzeichen
        INY
        CPY  _T+6       ;Restlänge von S
        BCC  DFILL1
        BCS  DEXIT1
;
DOVFL1  LDY  #Ø
        BEQ  DFILLØ
;
DEXIT1  NOP
#
END;


PROCEDURE INSERT(L1,L2,L3:INTEGER; VAR S1,S2:CHAR);
BEGIN
#A
        LDY  #5
        LDA  (_SP),Y
        STA  _T+1       ;S2-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+2       ;S2-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+3       ;S1-Zeiger.L
        INY
        LDA  (_SP),Y
        STA  _T+4       ;S1-Zeiger.H
        INY
        LDA  (_SP),Y
        STA  _T+5       ;Länge INSERT
        LDY  #11
        LDA  (_SP),Y
        STA  _T+6       ;Position INSERT
        DEC  _T+6       ;-1 Korrektur
        LDY  #13
        LDA  (_SP),Y
        STA  _T+7       ;Länge von S2
;
        SEC
        SBC  _T+6       ;Position INSERT
        BCC  IEXIT1     ;fertig, wenn >=L2
        BEQ  IEXIT1
;
        STA  _T+8       ;Restlänge von S2
;
        CLC
        LDA  _T+1       ;S2-Zeiger.L
        ADC  _T+6       ;Position INSERT
        STA  _T+1
        LDA  _T+2       ;S2-Zeiger.H
        ADC  #Ø         ;Carry-Flag
        STA  _T+2
;
        SEC
        LDA  _T+8       ;Restlänge von S2
        SBC  _T+5       ;Länge INSERT
        BCC  IOVFL1     ;Überlauf
        BEQ  OVERWR
        STA  _T+9       ;Restlänge INSERT
;
        CLC
        LDA  _T+1       ;S2-Zeiger.L
        ADC  _T+5       ;Länge INSERT
        STA  _T+1Ø      ;S2-Zeiger danach.L
        LDA  _T+2       ;S2-Zeiger.H
        ADC  #Ø         ;Carry-Flag
        STA  _T+11      ;S2-Zeiger danach.H
;
        LDY  _T+9       ;Restlänge INSERT
        DEY             ;-1 Korrektur
ICOPY1  LDA  (_T+1),Y   ;S2-Zeichen davor
        STA  (_T+1Ø),Y  ;S2-Zeichen danach
```

```
        DEY
        BPL  ICOPY1
;
OVERWR  LDY  _T+5       ;Länge INSERT
        DEY             ;-1 Korrektur
ICOPY2  LDA  (_T+3),Y   ;S1-Zeichen
        STA  (_T+1),Y   ;in S2 einfuegen
        DEY
        BPL  ICOPY2
        BMI  IEXIT1
;
IOVFL1  LDA  _T+8       ;Restlänge von S2
        STA  _T+5       ;Länge INSERT
        BNE  OVERWR     ;Verzweige immer
;
IEXIT1  NOP
#
END;
```

STRING.DEMOS.P

```
PROGRAM STRINGDEMOS(INPUT,OUTPUT);

VAR A,B,C:ARRAY [1..16] OF CHAR;
    D,E,F:ARRAY [1..40] OF CHAR;
    G,H,I:ARRAY [1..64] OF CHAR;
    C1,C2:CHAR;

#I STRINGUTILS.I

BEGIN

 {Die Funktion LENGTH

  Die Funktion LENGTH erzeugt einen INTEGER-Wert,
  welcher die Länge einer Zeichenkette
  (ARRAY OF CHAR) angibt.

  Syntax:

  LENGTH (Längenbyte der Zeichenkette,
  Name der Zeichenkette [1])

  Beispiel:}

  A:='1234567         ';
  WRITELN;
  WRITELN(A);
  WRITELN('LENGTH=',LENGTH(16,A[1]));


 {Die Funktion POS

  Die Funktion POS erzeugt einen INTEGER-Wert,
  welcher die Position angibt, ab der eine
  Zeichenkette S1 in einer Zeichenkette S2 enthalten ist.

  Syntax:

  POS (Längenbyte der Zeichenkette S1, Name von S1 [1],
       Längenbyte der Zeichenkette S2, Name von S1 [1]);

  Beispiel:}

  D:='Take the bottle with a metal cap        ';
  B:='tal             ';
  WRITELN;
  WRITELN(D);
  WRITELN(B);
  WRITELN('POS=',POS(LENGTH(16,B[1]),B[1],40,D[1]));


 {Die Prozedur CONCAT

  Die Prozedur CONCAT erzeugt eine Zeichenkette S3,
  welche aus den beiden Zeichenketten S1 und S2
  zusammengesetzt wird.

  Syntax

  CONCAT (Längenbyte von S1, Name von S1 [1],
          Längenbyte von S2, Name von S2 [1],
          Längenbyte von S3, Name von S3 [1]);

  Beispiele:}

  C:='12345           ';
  E:='Dies hier ist ein sehr langer String.    ';
  CONCAT(LENGTH(16,C[1]),C[1],LENGTH(40,E[1]),E[1],64,G[1]);
  WRITELN;
  WRITELN(C);
  WRITELN(E);
  WRITELN(G);


 {Die Prozedur COPY

  Die Prozedur COPY erzeugt eine Zeichenkette S2, die einen
  Teil aus der Übergabe-Zeichenkette S1 enthält.

  Syntax:

  COPY (Anzahl der zu uebertragenden Zeichen
        aus S1, Name von S1
        [Position in S1, ab der kopiert werden soll],
        Längenbyte von S2, Name der Zeichenkette S2 [1]);

  Beispiel:}

  WRITELN;
  C1:='s';
  F:='Keep something here                      ';
  WRITELN(F);
  COPY ( 9,F[POS(1,C1,40,F[1])],40,F[1]);
  WRITELN(F);


 {Die Prozedur DELETE

  Die Prozedur DELETE löscht einen Teil
  aus einer Zeichenkette S.

  Syntax:

  DELETE (Längenbyte der Zeichenkette, Startposition
          zum Löschen, Anzahl der Zeichen zu Löschen,
          Name von S [1]);

  Beispiel:}

  I:= 'This string has far too many
       characters in it                  ';
  {Aus satztechnischen Gründen wurde Zeile geteilt!}
  WRITELN;
  WRITELN(I);
  DELETE(64,16,8,I[1]);
  WRITELN(I);


 {Die Prozedur INSERT

  Die Prozedur INSERT fuegt einen Teil aus einer
  Zeichenkette S1 in eine bestehende Zeichenkette S2 ein.

  Syntax:

  INSERT (Längenbyte der Zeichenkette S2, Startposition
          zum Einfügen, Anzahl der Zeichen einzufügen,
          Name von S1 [Anfangsposition von S1],
          Name von S2 [1]);

  Beispiel:}

  D:='Demonstriere Einfügung                   ';
  B:='diese           ';
  WRITELN;
  WRITELN(D);
  WRITELN(B);
  C2:='E';
  INSERT(40,POS(1,C2,40,D[1]),LENGTH(16,B[1])+1,B[1],D[1]);
  WRITELN(D)

END.
```

Hinweis: Wenn Sie das Programm abtippen, so geben Sie bitte bei den Kommentaren keine Umlaute ein.

Programming Toolkits

für Kyan-Pascal 2.0

Toolkit I: System Utilities: Club-Preis DM 118,-; Normalpreis DM 148,- (lieferbar)

Toolkit II: Mouse Text: Club-Preis DM 118,-, Normalpreis DM 148,- (lieferbar)

Toolkit III: Advanced Graphics: Club-Preis DM 118,-, Normalpreis DM 148,- (Mitte Juni)

Toolkit IV: Turtle Graphics: Club-Preis DM 68,-, Normalpreis DM 88,- (lieferbar)

Toolkit V: Mouse Graphics: Club-Preis DM 158,-, Normalpreis DM 198,- (Mitte Juli)

Alle Utilities werden als teils beidseitig bespielte Disketten geliefert, die neben den Include-Files (meist Quelltexte) diverse Demos enthalten. Die Anleitungen selbst sind Loseblattlieferungen (z.B. bei den System Utilities 52 Druckseiten), die für den grauen Ordner von Kyan 2.0 bestimmt sind. Zum Club-Preis werden nur Mitglieder des Kyan-Clubs beliefert.

Toolkit I: System Utilities

Diese Utilities decken verschiedene Bereiche ab:
1. ProDOS-Utilities: Delete, Rename, Copy, Set-Pefix, Get-Prefix, Lock, Unlock, Make-Directory, Get-Directory, Remove-Directory, Scan-File, Format, Print-File, Bsave, Bload, Set-Time, Get-Time, Set-Date, Get-Date, Get-Clock, Set-Clock, Find-Clock.
2. Maus und Joystick: Find-Mouse, Init-Mouse, End-Mouse, Home-Mouse, Mouse-Click, Mouse-held, Mouse-moved, Mouse-x, Mouse-y, Set-Mouse-xy, Set-x-Bounds, Set-y-Bounds, Print-Mouse-Char, Button-0, Button-1, Joystick-x, Joystick-y.
3. Bildschirmsteuerung (funktionieren teilweise nur auf IIe/c): Clear-Screen, Clear-Line, Clear-End-of-Line, Clear-End-of-Page, Inverse, Normal, Tab, Scroll-up, Scroll-down, Col-80, On-40, On-80, Screen-Bottom, Screen-Top, Screen-Full, Cursor-x, Cursor-y, Get-Char, Machine-Identification.
4. Zufallszahlen: Random im Bereich min..max, Rnd im Bereich 0..1, Seeding.
5. Zahlenkonvertierungsroutinen: Real – String, String – Real, Integer – String, String – Integer.
6. Sortieren (alphabetisch und numerisch) sowie Mischen (bis zu 5 Files).
7. Line Parsing Routine.

Toolkit II: Mouse Text

Diese Utilities umfassen mehrere Dutzend Befehle für Fenstertechnik usw.
1. Cursor-Befehle: Set-Cursor, Obscure-Curser, Hide-Cursor, Show-Cursor.
2. Interrupts: Check-Events, Get-Event, Post-Event, Set-Key-Event, Flush-Event, Peek-Event.
3. Menü-Befehle: Init-Menu, Set-Menu, Menu-Select, Menu-Key, High-Light-Menu, Disable-Menu, Disable-Item, Check-Item, Set-Mark.
4. Kontrollbefehle: Find-Control, Set-Control-Max, Track-Thumb, Update-Thumb, Activate-Control.
5. Fensterbefehle: Init-Window-Margin, Close-Window, Open-Window, Find-Window, Front-Window, Select-Window, Drag-Window, Grow-Window, Screen-to-Window, Window-to-Screen, Close-all, Window-Char, Window-String, Window-Block, Window-Text, Window-Op.

Toolkit IV: Turtle Graphics

Diese Diskette enthält diverse Hires- und Ton-Routinen.
1. Turtle-Befehle: Init-Turtle, Turtle-x, Turtle-y, Turtle-Angle, Graf-Mode, Text-Mode, Pen-Color, Turn, Turn-to, Move, Move-to, View-Port, Full-Port, Fill-Port, Fill-Area, Save-Hires, Load-Hires.
2. Ton-Befehle: Beep, Note, Clock, Phaser.
3. Balkendiagramme usw.: Bar-Chart, Pie-Chart, Plot-x-y.

Hüthig Software Service
Postfach 10 28 69 Heidelberg

Double-Hires-Tools

von Matthias Meyer

NEU

Im September 1986 erscheinen zwei preisgünstige Programmpakete für doppelt-hochauflösende Grafik auf dem Apple IIc und IIe (mit 64K-Karte):

DHGR-Tool für Applesoft

Diskette und Manual, Einführungspreis DM 28,–

Diese Ampersand-Programmsammlung für Double-Hires und -Lores läuft unter Applesoft, und zwar sowohl unter DOS 3.3 als auch unter ProDOS. Unter anderen wurden folgende Befehle implementiert:

&1 und &2 wählen 1. und 2. Zeichensatz,
&CLEAR löscht die DHGR-Seite,
&COLOR= und &HCOLOR= wählen Double-Lores/Hires-Farben,
&DRAW und &XDRAW zeichnen DHGR-Shapes,
&DRAW AT zeichnet Grafikbeschriftungen (ASCII-Strings),
&GR, &HGR, &H, &TEXT, &T usw. schalten verschiedene Grafik- und Text-Modi ein,
&HLIN und &VLIN plotten waagrechte und senkrechte Double-Lores-Linien,
&HPLOT und &XHPLOT plotten DHGR-Linien,
&SCALE= und &ROT bestimmen Größe und Rotation von Shapes,
&LOAD und &SAVE laden und speichern Grafikseiten,
&HELP zeigt alle Befehle an,
und anderes mehr.

DHGR-Tool für Kyan-Pascal

Diskette und Manual, Einführungspreis DM 28,–

Das Kyan-Pascal-Tool umfaßt ähnliche Prozeduren wie die obigen Ampersand-Routinen, wobei jedoch noch einige Befehle, z. B. Procedure Swaphires, Procedure Background usw., sowie einige Datentypen, z. B. Shape, Chrset usw. zusätzlich aufgenommen worden sind.

Bei dem Kyan-Tool sind die Zeichensätze und die „Lookup"-Tabellen für die sehr schnellen Plotbefehle auf die 64K-Karte gelegt worden, und das Hauptmodul selbst befindet sich in der Bank 2 der Language-Card, ohne Kix-Reboot zu zerstören. Damit eignet sich dieses Kyan-Modul besser als andere Kyan-Grafik-Programme zur Einbindung in eigene Anwendungsprogramme.

Hüthig Software Service · Postfach 10 28 69 · 6900 Heidelberg 1

Buchkalkulation

Ein Applesoft-Kalkulationsprogramm für Verlage

von Ulrich Stiehl

Kalkulator (calculus = Rechenstein auf dem Rechenbrett, calculator = Rechenmeister) und Computer (to compute → computer → computator = Berechner) gehen beide auf Wörter mit derselben Grundbedeutung zurück. Es verwundert deshalb nicht, daß die Kalkulation als die ureigenste Aufgabe des Computers angesehen wird. Ergo computemus!
Das nachfolgende Kalkulationsprogramm EINZEL.KALK ist primär für Buchverlage gedacht, doch kann es zugleich als Muster für Kalkulationsprogramme in anderen Branchen dienen. Als eines meiner ersten Apple-Programme (das allerdings später überarbeitet wurde) benutzt EINZEL.KALK weder Peeks noch Pokes; selbst PRINT USING wurde durch eine Applesoft-Routine ersetzt.

1. Wie werden Bücher kalkuliert?

Aus Platzgründen ist es an dieser Stelle nicht möglich, auf die Feinheiten der Buchkalkulation einzugehen. Interessierte Leser seien deshalb auf die zwei einzigen seit dem Krieg erschienenen Monographien verwiesen, die sich mit der Buchkalkulation befassen, nämlich
– Preispolitik und Kalkulation im Buchverlag, Poeschel-Verlag, Stuttgart 1964 (vergriffen)
– Die Buchkalkulation. Mit 75 Musterkalkulationen, Harrassowitz-Verlag, 3., verb. Aufl. 1983 (enthält im Anhang kostentheoretische Erläuterungen zu dem gelisteten Programm)

1.1. Preisbindung

Die Kalkulation (= im engeren Sinne das Berechnen von Preisen) bezieht sich im Verlagswesen grundsätzlich auf die sog. *Ladenpreise* (Gegensatz: Listenpreise usw.), die als Endabnehmerpreise der vertikalen Preisbindung unterliegen, die aufgrund des Kartellgesetzes für typische Verlagserzeugnisse, d.h. für Bücher und Zeitschriften, nicht aber für Software und AV-Medien, zulässig ist (UWG §16). Preisbindung impliziert Gleichbehandlung der Endabnehmer – man erhält überall in Deutschland das gleiche Buch zum gleichen Preis – und Festpreispolitik über einen längeren Zeitraum – man kann nicht heute den Preis herauf- und morgen wieder heruntersetzen. Obgleich die vertikale Preisbindung eigentlich im Widerspruch zur freien Marktwirtschaft steht, wurde sie vom Gesetzgeber aus kulturellen Gründen ausdrücklich sanktioniert. Es verwundert deshalb nicht, daß die Bundesrepublik das Land mit der relativ größten Distributionsdichte (ca. 6.000 Buchverkaufsstellen) und der relativ größten Jahrestitelproduktion (ca. 50.000 Titel) ist. So gibt es beispielsweise in den USA relativ weniger Buchhandlungen und zugleich relativ mehr Analphabeten als in der Bundesrepublik. Der Gesetzgeber hat sich also einiges gedacht, als er die Preisbindung einführte, denn die Verlagswirtschaft wird im Gegensatz zu anderen kulturwirtschaftlichen Einrichtungen weder subventioniert (wie das Theater usw.) noch in staatlicher Regie geführt (wie Bibliotheken usw.).

1.2. Kalkulationselemente

Ein vereinfachtes Kalkulationsschema sieht etwa wie folgt aus:

```
  (1) Herstellungsstückkosten
+ (2) Honorarstückkosten
-------------------------------------
=     Einstandspreis
+ (3) Verlagsstückkosten
-------------------------------------
=     Selbstkostenpreis
+ (4) Verlagsstückgewinn
-------------------------------------
=     Nettopreis                  Np
+ (5) Stückrabatt
-------------------------------------
=     Ladenpreis ohne MwSt       NLp
+ (6) Mehrwertsteuer
-------------------------------------
=     Ladenpreis                  Lp
```

(1) **Herstellungskosten** sind die technischen Produktionskosten, die sich in auflagenvariable Einzelkosten (Papierkosten, Einbandmaterialkosten, Fortdruckkosten usw.) sowie auflagenfixe Einzelkosten (Satzkosten, Korrekturkosten, Montagekosten usw.) aufteilen lassen. Im Gegensatz zur landläufigen Meinung verteuert nicht der Einband das Buch, sondern die zu geringe Druckauflage, weil dann die Fixkosten auf eine zu kleine Auflage verteilt werden müssen. Herstellungskosten gehören im Verlagswesen übrigens zu den sog. Fremdkosten, weil die Produktion fast ausnahmslos in Fremdbetrieben (Druckereien, Buchbindereien usw.) erfolgt.
(2) **Honorarkosten** können als Pauschalhonorar auflagenfixe oder häufiger als Absatzhonorar auflagenvariable Kosten sein, die dann in der Regel als Prozentsatz vom Laden- oder Nettopreis berechnet werden. Neben dem verlorenen Pauschalhonorar gibt es noch die anrechenbare (anr.) Honorargarantie, die mit dem Absatzhonorar verrechnet wird. Auch die Honorarkosten sind sog. Fremdkosten, weil Autoren im Hinblick auf den Verlag fast ausnahmslos selbständig schreiben und somit keine Verlagsmitarbeiter sind.
(3) **Verlagskosten** sind überwiegend Gemeinkosten für Gehälter, Steuern, Miete usw., doch gibt es auch Einzelkosten, z.B. Kommissionsgebühren im Falle einer sog. Fremdauslieferung.
Hinsichtlich (4) **Verlagsgewinn**, (5) **Rabatt** und (6) **Mehrwertsteuer** ergeben sich – ähnlich wie bei den Gemeinkosten – keine wesentlichen Unterschiede zu anderen Branchen.

1.3. Kalkulationsziele

Es gibt im Verlagswesen theoretisch sechs Kalkulationsziele oder Größen, die es zu errechnen gilt:
– *Ladenpreis:* Wie hoch muß der Ladenpreis (DS-Ladenpreis = Deckungsspanne-Ladenpreis) sein, wenn Verkaufsauflage, Vollkosten und Gewinn vorgegeben sind? (Berechnung der Preisobergrenze)
– *Mindestladenpreis:* Wie niedrig darf der Ladenpreis sein, wenn Verkaufsauflage und Vollkosten (MLp I = Mindestladenpreis I) oder Teilkosten (MLp II) vorgegeben sind? (Berechnung der Preisuntergrenze)
– *Auflage:* Wie hoch muß die Verkaufsauflage (DS-Verkaufsauflage) sein, wenn Ladenpreis, Vollkosten und Gewinn vorgegeben sind? (Berechnung der Auflagenobergrenze)
– *Deckungsauflage:* Wie niedrig darf die Verkaufsauflage sein, wenn Ladenpreis und Vollkosten (Deckungsauflage I) oder Teilkosten (Deckungsauflage II) vorgegeben sind? (Berechnung der Auflagenuntergrenze)
– *Kosten:* Wie hoch dürfen oder wie niedrig müssen bestimmte Kosten sein, wenn Ladenpreis, Verkaufsauflage und die anderen Kosten vorgegeben sind? (Höchstkostenberechnung)
– *Deckungsbeitrag:* Wie hoch ist der Dekkungsbeitrag und die als Prozentsatz vom Umsatz ausgedrückte Deckungsspanne, wenn Ladenpreis, Verkaufsauflage und Teilkosten vorgegeben sind? (Überschußberechnung)

2. Das Kalkulationsprogramm

Das Kalkulationsprogramm EINZEL.KALK ist ein Einzelkalkulationsprogramm zur Berechnung der verschiedenen Kalkulationsgrößen für einen einzelnen in Planung befindlichen Buchtitel (Titelkalkulation als Vorkalkulation im Gegensatz zur Betriebsabrechnung als Nachkalkulation usw.).

2.1. Eingabedaten

Das Programm unterscheidet zwischen den *konstanten* Eingabedaten, die bei jeder Buchkalkulation benötigt werden, nämlich
– Mehrwertsteuersatz: Prozentsatz
– Vertriebskostensatz/Np: Prozentsatz
– Werbekostensatz/Np: Prozentsatz
– Gemeinkostensatz/Np: Prozentsatz
– Gewinnsatz/Np: Prozentsatz
und den *variablen* Eingabedaten, die nur von Fall zu Fall mit Werten belegt werden, nämlich:
– Ladenpreis: DM-Betrag
– Verkaufsauflage: Stückzahl
– Freiexemplare: Stückzahl
– Rabattsatz: Prozentsatz
– Vertriebsstückkosten: DM-Betrag
– Fester Werbeetat: DM-Betrag
– Fixe Herstellkosten: DM-Betrag
– Var. Herstellstückkosten: DM-Betrag
– Honorarsatz/Lp: Prozentsatz
– Honorarsatz/NLp: Prozentsatz
– Honorarsatz/Np: Prozentsatz
– Honorarstückkosten: DM-Betrag
– Honorarpauschale: DM-Betrag
– Honorarsatz/Lp (anr.): Prozentsatz
– Honorarsatz/NLp (anr.): Prozentsatz
– Honorarsatz/Np (anr.): Prozentsatz
– Honorarstückkosten (anr.): DM-Betrag
– Honorargarantie (anr.): DM-Betrag
– Sonstige Kosten: DM-Betrag
– Sonstige Erlöse: DM-Betrag
Die Druckauflage taucht bei den variablen Eingabedaten nicht auf, weil sie sich aus der Differenz zwischen Verkaufsauflage und Freiexemplaren errechnet.

2.2. Ausgabedaten

Bei den Ausgabedaten versteht sich die nachfolgende Aufstellung von selbst:

```
  Umsatz
- Vertriebskosten
- Werbekosten
- Herstellungskosten
- Honorarkosten
- Sonstige Kosten
+ Sonstige Erlöse
-----------------------------
= Deckungsbeitrag
  * Deckungsspanne in Prozent
- Gemeinkosten
-----------------------------
= Gewinn
  * Gewinnspanne in Prozent
=============================
  Investitionskosten
  Absatzhonorar (anrechenbar)
```

Die Investitionskosten sind die vor dem Verkauf des ersten Exemplares anfallenden Gesamtausgaben einschließlich einer möglichen Honorargarantie, die später mit dem anrechenbaren Absatzhonorar aufsaldiert wird.

```
Deckungsspanne-Ladenpreis
Deckungsspanne-Verkaufsauflage
Mindest-Ladenpreis  I
Mindest-Ladenpreis II
Deckungsauflage  I
Deckungsauflage II
```

Diese Ausgabedaten sind im obigen Abschnitt 1.3 definiert worden. Zum besseren Verständnis rechne man im Bedarfsfall einige fiktive Zahlenbeispiele durch. Dies ist jedoch noch nicht alles, denn es können zusätzlich Alternativkalkulationen durchgespielt sowie verschiedene Grafiken geplottet werden.

2.3. Programmaufbau

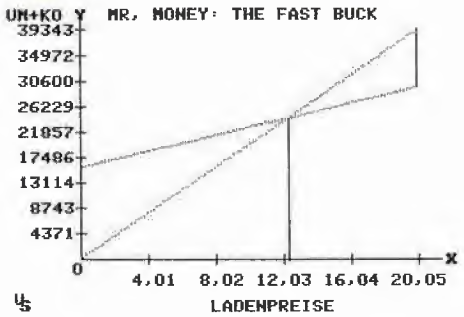
Hinweis: Die Buchstaben (a) bis (m) beziehen sich auf die gestrichelten Trennlinien in dem nachfolgend gelisteten Programm EINZEL.KALK.
Beim Programmstart (a) werden zunächst die Hires-Shapes für die Grafikbeschriftung (k) erzeugt und dann die Tabellenbeschriftung (l) für den Apple IIe/c (EINZEL.KLEIN) oder für den alten Apple II+ (EINZEL.GROSS) sowie die konstanten Eingabedaten EINZEL.KONST (m) eingelesen. Dann erscheint das Hauptmenü (b). Zunächst gibt man die neuen Daten ein (c) und kann dann zwischen Eingabekontrolle (d), Eingabekorrektur (e), Bildschirm- oder Druckerberechnung (f), Änderung- und Abspeicherung von Eingabekonstanten (h), Plotten und Abspeichern der Kurvengrafiken (i) und Alternativkalkulation (j) wählen.
Das Kernstück des Programms ist der Rechenteil (g), der beim Plotten der Umsatz- und Kostenkurven über 200mal aufgerufen wird. Man ändere deshalb ggf.

Programmzeile 528, falls gestrichelte Kurven ausreichen.

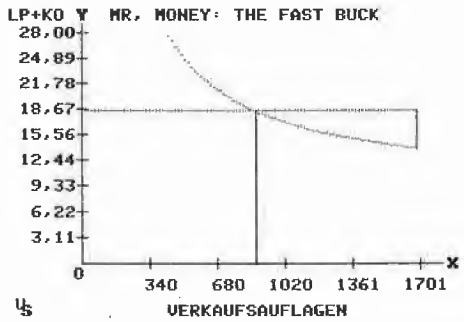
Das *Applesoft*-Programm EINZEL.KALK benutzt folgende Speicherverteilung:
$0800 ff.: EINZEL.KALK
$4000 ff.: HGR-Seite 2
$6000 ff.: HGR-Shapes
$6400 ff.: Variablen
Bei einem Applesoft-Programm dieser Größe bietet sich eine *Compilierung* an, die wegen der vielen GOTOs und GOSUBs das Programm 5-10mal schneller macht. Für die Compilierung mit dem Microsoft-TASC-Compiler müssen die Zeilen 632-832 gestrichen und folgende Zeilen geändert werden:

```
1ØØ PRINT CHR$ (21): HOME
452 HOME : HGR : POKE - 16302,Ø:
    POKE 232,Ø: POKE 233,9:
    HCOLOR= 7: SCALE= 1: ROT= Ø
528 Y = U / 1Ø5:SD = 1: REM *** LOT-FLAG
562 IF X$ = "3" THEN POKE - 16304,Ø:
    POKE - 16297,Ø: POKE - 16300,Ø:
    POKE - 16302,Ø: GOTO 554
566 PRINT : PRINT CHR$ (4)
    "BSAVE"X$",A$2ØØØ,L$1FFØ": GOTO 168
```

Die Hires-Shapes müssen gesondert abgespeichert und zweckmäßigerweise mit der TASC-RUNTIME zu EINZEL.RUNTIME vereinigt werden, wobei zur Compilierung als „Origin" für die Runtime $0D00 anzugeben ist. Die Peeker-Sammeldisk enthält bereits das fertig compilierte Programm. Die TASC-Version hat folgende Speicherverteilung:
$0800 ff. EINZEL.TASC (Start-Programm)
$0900 ff.: EINZEL.RUNTIME, und zwar
$0900-$0CE7: Shapes
$0D00-$1CAF: Runtime
$2000 ff.: HGR-Seite 1
$4000 ff.: EINZEL.OBJ



Beispiel für Gesamtkostenkurve



Beispiel für Stückkostenkurve

2.4. Zu den Abbildungen

Die Ladenpreisgrafik (hier Gesamtkostenkurve) und Auflagengrafik (hier Stückkostenkurve) ergeben sich, wenn man neben den Konstanten folgende Werte eingibt:
Ladenpreis: DM 28,-
Verkaufsauflage: 3000 Ex.
Freiexemplare: 50 Ex.
Rabattsatz: 30%
Fixe Herstellkosten: DM 8000,-
Var. Herstellkosten: DM 2,50

Kurzhinweise

1. Zweck:
Buchkalkulationsprogramm; als Einzelproduktkalkulation für andere Branchen adaptierbar.
2. Konfiguration:
Apple II+/e/c; DOS 3.3; das nicht-compilierte Applesoft-Programm läuft auch unter ProDOS
3. Test:
RUN EINZEL.TASC oder RUN EINZEL.KALK
4. Sammeldisk:
EINZEL.TASC
EINZEL.RUNTIME
EINZEL.OBJ
(TASC-Module)
EINZEL.KALK
(Applesoft-Programm)
EINZEL.KONST
EINZEL.KLEIN
EINZEL.GROSS
(Daten-Textfiles)

EINZEL.KALK

```
-----------------------------------------------------------(a)
1ØØ PRINT CHR$ (21): HOME : LOMEM: 256ØØ: PRINT "ZEICHENSATZ
    ERZEUGEN...": GOSUB 632: REM AB $6ØØØ
1Ø2 ONERR GOTO 1Ø8
1Ø4 DIM A(5),A$(5),S(19),S$(19),T(19),T$(19)
1Ø6 GOTO 118
1Ø8 PRINT : PRINT CHR$ (4)"CLOSE": PRINT CHR$ (4)"PR#Ø"
11Ø TEXT : HOME : PRINT "SIE HABEN FEHLER-NR. " PEEK (222)"
    GEMACHT"
112 HTAB 1: VTAB 5: PRINT "W = WEITER ";: GET X$: IF X$ < > "W"
    THEN 112
114 GOTO 168
116 REM *** A$(1...5)=KONSTANTEN;
    S$(Ø...S)=EINGABE;T$(Ø...T)=AUSGABE
118 TEXT : HOME : PRINT "KLEINSCHREIBUNG J/N ";
12Ø GET X$: IF X$ < > "J" AND X$ < > "N" THEN 12Ø
122 IF X$ = "J" THEN X$ = "EINZEL.KLEIN": GOTO 126
124 X$ = "EINZEL.GROSS"
126 PRINT : PRINT CHR$ (4)"OPEN"X$: PRINT CHR$ (4)"READ"X$
128 FOR Z = 1 TO 5: INPUT A$(Z): NEXT :S = 19: FOR Z = Ø TO S:
    INPUT S$(Z): NEXT :T = 19: FOR Z = Ø TO T: INPUT T$(Z): NEXT
13Ø PRINT CHR$ (4)"CLOSE"
132 PRINT CHR$ (4)"OPEN EINZEL.KONST": PRINT CHR$ (4)"READ
    EINZEL.KONST": FOR Z = 1 TO 5: INPUT A(Z): NEXT : PRINT CHR$
    (4)"CLOSE"
134 REM *** PRINT USING
136 K2 = 1ØØ:K7 = 9999999:K6 = 999999: GOTO 154
138 K = INT (X): IF ABS (X) > K7 THEN X$ = "ZU GROSS": RETURN
14Ø IF K = X THEN X$ = STR$ (X) + ".ØØ": RETURN
142 K = INT (X * K2 + .5) / K2:X$ = STR$ (K): IF K = INT (K) THEN
    X$ = X$ + ".ØØ": RETURN
144 IF K < 1 AND K > Ø THEN X$ = "Ø" + X$
146 IF K > - 1 AND K < Ø THEN X$ = "-Ø." + MID$ (X$,3,2)
148 IF MID$ (X$, LEN (X$) - 2,1) = "." THEN RETURN
15Ø X$ = X$ + "Ø": RETURN
152 REM *** DIVERSE TEXTE
154 R$ = "1 = ZURUECK "
156 RØ$ = "                                                  "
158 R1$ = "-----------------------------------------"
16Ø R3$ = "IST DRUCKER EINGESCHALTET ??"
162 R7$ = "DATEN UNVOLLSTAENDIG"
164 E$ = "..... ": REM *** AUSPUNKTIERUNG
-----------------------------------------------------------(b)
166 REM *** MENUE ***
168 TEXT : HOME : INVERSE :
    PRINT "          EINZELKALKULATION        ":
    NORMAL : PRINT
17Ø PRINT "   VON U. STIEHL * VERSION V. 1.9.82   "
172 PRINT : PRINT :
    PRINT "  1 NEUEINGABE"
174 PRINT "  2 EINGABEKONTROLLE"
176 PRINT "  3 EINGABEKORREKTUR"
178 PRINT :
    PRINT "  4 BILDSCHIRMBERECHNUNG"
18Ø PRINT "  5 DRUCKERBERECHNUNG"
182 PRINT :
    PRINT "  6 GRAFIK"
184 PRINT "  7 KONSTANTENLISTE"
186 PRINT "  8 ALTERNATIVKALKULATION"
188 PRINT :
    PRINT "  9 ENDE"
```

```
190 INVERSE : FOR X = 1 TO 18: HTAB 1: VTAB X: PRINT " ";: HTAB
    39: VTAB X: PRINT " ";: NEXT : FOR X = 1 TO 39: VTAB 19: HTAB
    X: PRINT " ";: NEXT : NORMAL
192 HTAB 1: VTAB 21: CALL - 958: HTAB 2: VTAB 21: PRINT "( )":
    HTAB 3: VTAB 21: GET X$:X = VAL (X$): IF X < 1 OR X > 9 THEN
    192
194 ON X GOTO 198,218,226,250,250,420,392,570,412
-----------------------------------------------------------------(c)
196 REM *** NEUEINGABE
198 HOME : HTAB 37: PRINT "*";: HTAB 1: INPUT "TITEL: ";T$:T$ =
    LEFT$ (T$,29): IF T$ = "" THEN T$ = " "
200 FOR X = 1 TO LEN (T$):Y = ASC ( MID$ (T$,X,1)): IF Y < 32 OR
    Y > 93 THEN PRINT "NUR GROSSBUCHSTABEN ";: GET X$: GOTO 198
202 NEXT
204 HOME : PRINT T$: PRINT R1$
206 FOR Z = 0 TO 5: PRINT S$(Z): NEXT
208 FOR Z = 0 TO 5: VTAB Z + 3: HTAB 30: INPUT "";X$:S(Z) = VAL
    (X$): IF S(Z) > K6 OR S(Z) < 0 THEN S(Z) = 0
210 VTAB Z + 3: HTAB 30:X = S(Z): GOSUB 138: PRINT MID$ (E$, LEN
    (X$) + 1,10)X$: NEXT
212 HTAB 1: VTAB 23: PRINT R$;: GET X$: IF X$ < > "1" THEN 212
214 GOTO 168
-----------------------------------------------------------------(d)
216 REM *** EINGABEKONTROLLE
218 HOME : PRINT T$: PRINT R1$
220 FOR Z = 0 TO 5: PRINT S$(Z);:X = S(Z): GOSUB 138: PRINT MID$
    (E$, LEN (X$) + 1,10)X$: NEXT

222 PRINT R$;: GET X$: GOTO 168
-----------------------------------------------------------------(e)
224 REM *** EINGABEKORREKTUR
226 HOME : PRINT T$: PRINT R1$: FOR Z = 0 TO 5: PRINT S$(Z);:X =
    S(Z): GOSUB 138: PRINT MID$ (E$, LEN (X$) + 1,10)X$: NEXT
228 VTAB 23: HTAB 1: PRINT R0$;: VTAB 23: HTAB 1: PRINT "WELCHE
    ZEILE (0 - "5"): ";: INPUT "";X$
230 Z = INT ( VAL (X$)): IF X$ = "0" OR X$ = "00" THEN 234
232 IF Z > 5 OR Z < 1 THEN 228
234 IF F = 1 THEN VTAB Z + 3: HTAB 4: INVERSE : PRINT "=>":
    NORMAL : VTAB 23: HTAB 1: PRINT R0$;: VTAB 23: HTAB 1: PRINT
    R$;: GET X$: GOTO 576
236 VTAB Z + 3: HTAB 1: PRINT R0$;: VTAB Z + 3: HTAB 1: PRINT
    S$(Z);: INPUT "";X$:X = VAL (X$): IF X = 0 AND X$ < > "0"
    THEN 240
238 S(Z) = X: IF S(Z) > K6 OR S(Z) < 0 OR X$ = "0" THEN S(Z) = 0
240 VTAB Z + 3: HTAB 1: PRINT R0$;: VTAB Z + 3: HTAB 1: PRINT
    S$(Z);:X = S(Z): GOSUB 138: PRINT MID$ (E$, LEN (X$) +
    1,10)X$;
242 VTAB 23: HTAB 1: PRINT "1 = ZURUECK  *  9 = ERNEUTE AENDERUNG
    ";: GET X$: IF X$ = "1" GOTO 168
244 IF X$ < > "9" THEN 242
246 GOTO 228
-----------------------------------------------------------------(f)
248 REM *** BILDSCHIRM- UND DRUCKERBERECHNUNG
250 HOME
252 REM *** X VON ON X GOTO
254 IF X < > 5 THEN F = 0: GOTO 268
256 F = 1: PRINT CHR$ (7): HOME : INVERSE : PRINT R3$: NORMAL :
    VTAB 3: PRINT R$: VTAB 5: PRINT "3 = AUSDRUCK": VTAB 7: GET
    X$: IF X$ < > "3" GOTO 168
258 HOME : PRINT : PRINT CHR$ (4)"PR#1": PRINT
260 PRINT T$: PRINT R1$: FOR Z = 0 TO 5: IF S(Z) = 0 THEN NEXT :
    GOTO 264
262 PRINT S$(Z);:X = S(Z): GOSUB 138: PRINT MID$ (E$, LEN (X$) +
    1,10)X$: NEXT
264 FOR Z = 1 TO 5: PRINT A$(Z);:X = A(Z): GOSUB 138: PRINT MID$
    (E$, LEN (X$) + 1,10)X$: NEXT : PRINT : HOME
-----------------------------------------------------------------(g)
266 REM *** DB-BERECHNUNG
268 GOSUB 270: GOSUB 290: GOTO 382
270 A1 = A(1) + K2:B = S(0) * S(1)
272 T(0) = B * (K2 - S(3)) / A1:T(1) = T(0) * A(2) / K2 + S(1) *
    S(4):T(2) = T(0) * A(3) / K2 + S(5):T(3) = S(6) + (S(1) +
    S(2)) * S(7)
274 T(13) = B * S(13) / K2 + B * S(14) / A1 + T(0) * S(15) / K2 +
    S(1) * S(16):T(4) = T(13): IF S(17) > T(13) THEN T(4) = S(17)
276 T(4) = T(4) + B * S(8) / K2 + B * S(9) / A1 + T(0) * S(10) /
    K2 + S(1) * S(11) + S(12)
278 REM *** E = EINZELKOSTEN
280 T(5) = S(18):T(6) = S(19):E = T(1) + T(2) + T(3) + T(4) +
    T(5) - T(6):T(7) = T(0) - E:T(9) = T(0) * A(4) / K2:T(10) =
    T(7) - T(9)
282 T(8) = 0:T(11) = 0: IF T(0) > 0 THEN T(8) = T(7) * K2 /
    T(0):T(11) = T(10) * K2 / T(0)
284 T(12) = S(5) + S(12) + S(17) + S(18) - S(19) + T(3)
286 RETURN
288 REM *** TERME
290 B1 = S(5) + S(12) + S(18) - S(19):B2 = (K2 - S(3)) / A1 /
    K2:B3 = K2 - A(2) - A(3) - S(10):B4 = S(8) / K2 + S(9) /
    A1:B5 = S(13) / K2 + S(14) / A1 + B2 * S(15):B6 = S(11) +
    S(4)
292 B7 = B2 * (B3 - A(4) - A(5)):B8 = B1 + T(3) + S(1) * B6:B9 =
    B1 + S(6) + S(2) * S(7):BA = S(0) * B2 * B3 - S(0) * B4 -
    B6:BB = B2 * (B3 - A(4)):BC = S(0) * B4 + B6 + S(7): FOR Z =
    14 TO 19:T(Z) = 0: NEXT : IF T(12) < = 0 THEN RETURN
294 REM *** DS-LP
296 NN = S(1) * (B7 - B4 - B5):ZZ = B8 + S(1) * S(16): IF NN < =
    0 OR ZZ < = 0 GOTO 306
298 T(14) = ZZ / NN: IF S(1) * (S(16) + T(14) * B5) > = S(17)
    GOTO 306
300 NN = S(1) * (B7 - B4):ZZ = B8 + S(17): IF NN < = 0 OR ZZ < =
    0 THEN T(14) = 0: GOTO 306
302 T(14) = ZZ / NN
304 REM *** MLP-I
306 NN = S(1) * (BB - B4 - B5):ZZ = B8 + S(1) * S(16): IF NN < =
    0 OR ZZ < = 0 GOTO 316
308 T(16) = ZZ / NN: IF S(1) * (S(16) + T(16) * B5) > = S(17)
    GOTO 316
310 NN = S(1) * (BB - B4):ZZ = B8 + S(17): IF NN < = 0 OR ZZ < =
    0 THEN T(16) = 0: GOTO 316
312 T(16) = ZZ / NN
314 REM *** MLP-II
316 NN = S(1) * (B2 * B3 - B4 - B5):ZZ = B8 + S(1) * S(16): IF NN
    < = 0 OR ZZ < = 0 GOTO 326
318 T(17) = ZZ / NN: IF S(1) * (S(16) + T(17) * B5) > = S(17)
    GOTO 326
320 NN = S(1) * (B2 * B3 - B4):ZZ = B8 + S(17): IF NN < = 0 OR ZZ
    < = 0 THEN T(17) = 0: GOTO 326
322 T(17) = ZZ / NN
324 REM *** DS-VA
326 NN = S(0) * (B7 - B5) - BC - S(16):ZZ = B9: IF NN < = 0 OR ZZ
    < = 0 GOTO 336
328 T(15) = ZZ / NN: IF T(15) * (S(16) + S(0) * B5) > = S(17)
    GOTO 336
330 NN = S(0) * B7 - BC:ZZ = B9 + S(17): IF NN < = 0 OR ZZ < = 0
    THEN T(15) = 0: GOTO 336
332 T(15) = ZZ / NN
334 REM *** DA-I
336 IF S(1) = 0 GOTO 348
338 NN = BA - S(0) * B5 - S(16):ZZ = B1 + T(3) + T(9): IF NN < =
    0 OR ZZ < = 0 GOTO 358
340 T(18) = ZZ / NN: IF T(18) * (S(16) + S(0) * B5) > = S(17)
    GOTO 358
342 NN = BA:ZZ = B1 + T(3) + T(9) + S(17): IF NN < = 0 OR ZZ < =
    0 THEN T(18) = 0: GOTO 358
344 T(18) = ZZ / NN: GOTO 358
346 REM *** DA-I = VA
348 NN = S(0) * (BB - B5) - BC - S(16):ZZ = B9: IF NN < = 0 OR ZZ
    < = 0 GOTO 358
350 T(18) = ZZ / NN: IF T(18) * (S(16) + S(0) * B5) > = S(17)
    GOTO 358
352 NN = S(0) * BB - BC:ZZ = B9 + S(17): IF NN < = 0 OR ZZ < = 0
    THEN T(18) = 0: GOTO 358
354 T(18) = ZZ / NN
356 REM *** DA-II
358 IF S(1) = 0 GOTO 370
360 NN = BA - S(0) * B5 - S(16):ZZ = B1 + T(3): IF NN < = 0 OR ZZ
    < = 0 GOTO 378
362 T(19) = ZZ / NN: IF T(19) * (S(16) + S(0) * B5) > = S(17)
    GOTO 378
364 NN = BA:ZZ = B1 + T(3) + S(17): IF NN < = 0 OR ZZ < = 0 THEN
    T(19) = 0: GOTO 378
366 T(19) = ZZ / NN: GOTO 378
368 REM *** DA-II = VA
370 NN = BA - S(7) - S(0) * B5 - S(16):ZZ = B9: IF NN < = 0 OR ZZ
    < = 0 GOTO 378
372 T(19) = ZZ / NN: IF T(19) * (S(16) + S(0) * B5) > = S(17)
    GOTO 378
374 NN = BA - S(7):ZZ = B9 + S(17): IF NN < = 0 OR ZZ < = 0 THEN
    T(19) = 0: GOTO 378
376 T(19) = ZZ / NN
378 RETURN
380 REM *** AUSGABEDATEN
382 PRINT T$: PRINT R1$: FOR Z = 0 TO T: IF T(Z) = 0 THEN NEXT :
    GOTO 386
384 PRINT T$(Z);:X = T(Z): GOSUB 138: PRINT MID$ (E$, LEN (X$) -
    3,10)X$: NEXT
386 IF F = 1 THEN F = 0: PRINT CHR$ (12): PRINT CHR$ (4)"PR#0"
388 PRINT R$;: GET X$: GOTO 168
-----------------------------------------------------------------(h)


390 REM *** KONSTANTENLISTE
392 HOME : INVERSE : PRINT "                KONSTANTENLISTE
    ": NORMAL : PRINT : PRINT
394 FOR Z = 1 TO 5: HTAB 1: VTAB Z + 3: PRINT A$(Z);:X = A(Z):
    GOSUB 138: PRINT MID$ (E$, LEN (X$) + 1,10)X$
396 HTAB 30: VTAB Z + 3: INPUT "";X$:X = VAL (X$): IF X > 0 AND X
    < K6 OR X$ = "0" THEN A(Z) = X
398 X = A(Z): HTAB 30: VTAB Z + 3: GOSUB 138: PRINT MID$ (E$, LEN
    (X$) + 1,10)X$: NEXT Z
400 PRINT : PRINT "SPEICHERN J/N ";
402 GET X$: IF X$ < > "J" AND X$ < > "N" THEN 402
404 IF X$ = "J" THEN PRINT : PRINT CHR$ (4)"OPEN EINZEL.KONST":
```

```
    PRINT CHR$ (4)"WRITE EINZEL.KONST"
4Ø6 IF X$ = "J" THEN FOR Z = 1 TO 5: PRINT A(Z): NEXT : PRINT
    CHR$ (4)"CLOSE"
4Ø8 GOTO 168
41Ø REM *** ENDE
412 HTAB 1: VTAB 23: INPUT "WIRKLICH ENDE JA/N ";X$
414 IF X$ < > "JA" THEN 192
416 HOME : PRINT "ENDE...": END
-----------------------------------------------------(i)

418 REM *** GRAFIK
42Ø HOME : GOSUB 27Ø: GOSUB 29Ø
422 VTAB 1: HTAB 16: INVERSE : PRINT "GRAFIKEN": VTAB 3: PRINT "
    FUER AUFLAGE-LADENPREIS-ALTERNATIVEN ": NORMAL
424 PRINT : FOR Z = 14 TO 15: PRINT T$(Z);:X = T(Z): GOSUB 138:
    PRINT MID$ (E$, LEN (X$) - 3,1Ø)X$: NEXT
426 PRINT : IF T(14) = Ø AND T(15) = Ø THEN PRINT R7$: PRINT :
    PRINT R$;: GET X$: GOTO 168
428 IF T(14) = Ø THEN PRINT "LADENPREIS-GRAFIK NICHT MOEGLICH"
43Ø IF T(15) = Ø THEN PRINT "VERKAUFSAUFLAGEN-GRAFIK NICHT
    MOEGLICH"
432 PRINT : PRINT R$: PRINT : PRINT "3 = LADENPREIS-GRAFIK":
    PRINT : PRINT "5 = VERKAUFSAUFLAGEN-GRAFIK": PRINT
434 GET X$: IF X$ = "1" THEN 168
436 REM *** F = Ø = DS-LP * F = 1 = DS-VA
438 IF X$ = "3" AND T(14) < > Ø THEN F = Ø: GOTO 444
44Ø IF X$ = "5" AND T(15) < > Ø THEN F = 1: GOTO 444
442 GOTO 434
444 UU = S(F):U = T(14 + F):S(F) = U: GOSUB 27Ø:UM = T(Ø):
    REM *** UM=UMSATZ-OBERGRENZE FÜR Y-ACHSE * U=DS-LP/DS-VA
446 PRINT : PRINT "G = GESAMTKOSTENKURVE": PRINT : PRINT "S =
    STUECKKOSTENKURVE": PRINT
448 GET X$: IF X$ < > "G" AND X$ < > "S" THEN 448
45Ø V = Ø: IF X$ = "S" THEN V = 1:UM = S(Ø)
452 HOME : HGR2 : POKE 232,Ø: POKE 233,96: HCOLOR= 7: SCALE= 1:
    ROT= Ø
454 REM *** ROHGRAFIK
456 HPLOT 46,9 TO 46,158: HPLOT 46,158 TO 27Ø,158:
    REM *** KOORDINATENKREUZ
458 FOR Y = 14 TO 142 STEP 16: HPLOT 43,Y TO 49,Y: NEXT :
    REM *** Y-UNTERTEILUNG
46Ø FOR Y = 88 TO 256 STEP 42: HPLOT Y,155 TO Y,161: NEXT :
    REM *** X-UNTERTEILUNG
462 X$ = "Ø":SL = 1:SH = 6:SV = 21:BH = 5:BV = 1: GOSUB 516:BH =
    6: GOSUB 516: REM *** ORIGO
464 X$ = "Y":SL = 1:SH = 7:SV = 1:BH = 1:BV = Ø: GOSUB 516:BH =
    2: GOSUB 516:BH = Ø: GOSUB 516: REM *** Y-MARKIERUNG
466 X$ = "UM+KO": IF V = 1 THEN X$ = "LP+KO"
468 SL = 5:SH = 1:SV = 1:BV = Ø:BH = Ø: GOSUB 516:BH = 1: GOSUB
    516: REM *** Y-WERTE
47Ø X$ = "X":SL = 1:SH = 4Ø:SV = 2Ø:BH = Ø:BV = 3: GOSUB 516:BH =
    1: GOSUB 516:SH = 39:SV = 2Ø:BH = 6: GOSUB 516:
    REM *** X-MARKIERUNG
472 IF F = 1 THEN X$ = "VERKAUFSAUFLAGEN":SL = LEN (X$):SV =
    24:SH = 15:BV = Ø:BH = Ø: GOSUB 516:BH = 1: GOSUB 516: GOTO
    476
474 X$ = "LADENPREISE":SL = LEN (X$):SV = 24:SH = 19:BV = Ø:BH =
    Ø: GOSUB 516:BH = 1: GOSUB 516
476 X$ = " " + T$:SL = LEN (X$):SV = 1:SH = 9:BV = Ø:BH = Ø:
    GOSUB 516:BH = 1: GOSUB 516: REM *** TITEL
478 X$ = "U":SL = 1:SH = 1:SV = 23:BH = 4:BV = 4: GOSUB 516:BH =
    5: GOSUB 516:X$ = "S":SL = 1:SH = 2:SV = 24:BH = Ø:BV = Ø:
    GOSUB 516:BH = 1: GOSUB 516:BH = 2: GOSUB 516
48Ø REM *** 5 X-ACHSEN-ZAHLEN (LP/VA)
482 SV = 22:SH = 1Ø:BV = Ø:Y = U / 5: FOR Z = 1 TO 5:X = Z * Y:
    IF F = Ø THEN GOSUB 138: GOTO 488
484 X$ = STR$ ( INT (X)):SL = LEN (X$): IF SL > 5 THEN X$ = LEFT$
    (X$,SL - 3) + "T"
486 GOTO 492
488 SL = LEN (X$): IF SL = 6 THEN X$ = LEFT$ (X$,SL - 1)
49Ø SL = LEN (X$): IF SL > 6 THEN X$ = LEFT$ (X$,SL - 3)
492 SL = LEN (X$): IF SL < 6 THEN X$ = " " + X$: GOTO 492
494 BH = Ø: GOSUB 516:BH = 1: GOSUB 516:SH = SH + 6: NEXT Z
496 REM *** 9 Y-ACHSEN-ZAHLEN (UM=UMSATZ; BEI V=1 LP)
498 SV = 18:SH = 1:BV = 3:Y = UM / 9: FOR Z = 1 TO 9:X = Z * Y:
    IF V = Ø THEN X$ = STR$ ( INT (X)): GOTO 5Ø6
5ØØ GOSUB 138:SL = LEN (X$): IF SL = 7 THEN X$ = LEFT$ (X$,SL -
    1)
5Ø2 SL = LEN (X$): IF SL > 6 THEN X$ = LEFT$ (X$,SL - 3)
5Ø4 GOTO 5Ø8
5Ø6 SL = LEN (X$): IF SL > 6 THEN X$ = LEFT$ (X$,SL - 3) + "T"
5Ø8 SL = LEN (X$): IF SL < 6 THEN X$ = " " + X$: GOTO 5Ø8
51Ø BH = Ø: GOSUB 516:BH = 1: GOSUB 516:SV = SV - 2: NEXT Z
512 GOTO 528
514 REM *** ZEICHENROUTINE
516 FOR X = 1 TO SL
518 SD = ASC ( MID$ (X$,X,1)) - 31: IF SD = 17 THEN SD = 48: REM
    *** 0 STATT Ø
52Ø IF SD = 15 THEN SD = 13: REM , STATT .
522 DRAW SD AT (X + SH - 2) * 7 + BH,6 + (SV - 1) * 8 + BV
524 NEXT : RETURN

526 REM *** UMSATZ- UND KOSTENKURVEN
    *** In der nächsten Zeile "U / 21Ø" durch "U / 1Ø5"
    ersetzen, wenn doppelt so schnell geplottet werden soll!
528 Y = U / 21Ø:SD = 1: REM *** LOT-FLAG
53Ø V1 = 21Ø / U:V2 = - 144 / UM:
    REM *** 256-46=21Ø=X-PUNKTE * 14-158=-144=Y-PUNKTE
532 FOR Z = .1 TO U STEP Y:S(F) = Z: GOSUB 27Ø
534 IF V = 1 THEN T(Ø) = T(Ø) / S(1):E = E / S(1)
536 X1 = V1 * Z + 46:Y1 = V2 * T(Ø) + 158:Y2 = V2 * E + 158
538 IF X1 < 46 OR X1 > 256 THEN 546
54Ø IF Y1 > 13 AND Y1 < 159 THEN HPLOT X1,Y1: REM *** UMSATZ-PLOT
542 IF Y2 > 13 AND Y2 < 159 THEN HPLOT X1,Y2: REM *** KOSTEN-PLOT
544 IF SD = 1 AND Y1 < Y2 AND Y1 > 13 AND Y1 < 159 THEN SD = Ø:
    HPLOT X1,Y1 TO X1,159: REM *** BREAK-EVEN-LOT
546 NEXT
548 IF X1 < 46 OR X1 > 256 OR Y1 < 14 OR Y2 < 14 OR Y1 > 158 OR
    Y2 > 158 THEN 552
55Ø HPLOT X1,Y1 TO X1,Y2
552 S(F) = UU:F = Ø:V = Ø:SD = Ø: REM *** FLAGS ZURÜCKSETZEN
554 GET X$
556 TEXT : HOME : PRINT R$: PRINT : PRINT "3 = ERNEUT ANSEHEN":
    PRINT : PRINT "5 = SPEICHERN": PRINT
558 GET X$: IF X$ < > "1" AND X$ < > "3" AND X$ < > "5" THEN 558
56Ø IF X$ = "1" THEN 42Ø
562 IF X$ = "3" THEN POKE - 163Ø4,Ø: POKE - 16297,Ø: POKE -
    16299,Ø: GOTO 554
564 HOME : PRINT : PRINT CHR$ (4)"CATALOG": INPUT "DATEINAME:
    ";X$: PRINT "VERTIPPT J/N ";: GET U1$: IF U1$ < > "N" THEN
    556
566 PRINT : PRINT CHR$ (4)"BSAVE"X$",A$4ØØØ.L$1FFØ": GOTO 168
-----------------------------------------------------(j)
568 REM *** ALTERNATIVKALKULATION
57Ø HOME : INVERSE : HTAB 9: PRINT "ALTERNATIVKALKULATION":
    NORMAL
572 VTAB 4: PRINT R$: VTAB 6: PRINT "2 = MARKIERUNG": VTAB 8: GET
    X$: IF X$ < > "2" GOTO 168
574 U1$ = T$:T$ = "EINGABEZEILE ALS ALTERNATIVE MARKIEREN":F = 1:
    GOTO 226
576 HOME :V = Z:F = Ø:T$ = U1$: GOSUB 27Ø: IF T(7) = Ø OR T(8) =
    Ø THEN PRINT R7$: PRINT : PRINT R$;: GET X$: GOTO 168
578 PRINT S$(Z);:X = S(Z): GOSUB 138: PRINT MID$ (E$, LEN (X$) +
    1,1Ø)X$
58Ø VTAB 3: HTAB 1: PRINT "VERTIPPT J/N ";: GET X$: IF X$ = "J"
    GOTO 574
582 VTAB 5: HTAB 1: CALL - 958
584 INPUT "UNTERGRENZE:  ";X$:V1 = VAL (X$)
586 VTAB 7: HTAB 1: INPUT "OBERGRENZE:   ";X$:V2 = VAL (X$)
588 IF V1 < = Ø OR V2 < = Ø OR V1 > = V2 OR V1 > K6 OR V2 > K6
    GOTO 582
59Ø VTAB 9: HTAB 1: PRINT "VERTIPPT J/N ";: GET X$: IF X$ = "J"
    GOTO 582
592 HOME : PRINT CHR$ (7): HTAB 1: VTAB 1: INVERSE : PRINT R3$:
    NORMAL : VTAB 3: PRINT "1 BILDSCHIRMBERECHUNG": VTAB 5: PRINT
    "5 DRUCKERBERECHNUNG"
594 VTAB 7: GET X$: IF X$ < > "1" AND X$ < > "5" THEN 594
596 F = Ø: IF X$ = "1" THEN F = 1: HOME : PRINT T$: PRINT R1$:Y =
    7: GOTO 6Ø6
598 PRINT : PRINT CHR$ (4)"PR#1": PRINT
6ØØ PRINT T$: PRINT
6Ø2 REM *** Y = ANZAHL DER BERECHNUNGEN
6Ø4 Y = 25
6Ø6 Y = (V2 - V1) / (Y - 1) - .ØØØ5:U = S(V): IF F = 1 THEN U1$ =
    S$(V):U2$ = T$(7):U3$ = T$(8): GOTO 612
6Ø8 U = S(V):U1$ = RIGHT$ (S$(V),23)
61Ø U2$ = RIGHT$ (T$(7),19):U3$ = RIGHT$ (T$(8),19)
612 FOR Z = V1 TO V2 STEP Y
614 S(V) = Z: GOSUB 27Ø
616 PRINT U1$;:X = Z: GOSUB 138: PRINT MID$ (E$, LEN (X$) +
    1,1Ø)X$;: IF F = 1 THEN PRINT : GOTO 62Ø
618 PRINT "          ";
62Ø PRINT U2$;:X = T(7): GOSUB 138: PRINT MID$ (E$, LEN (X$) -
    3,1Ø)X$: IF F = 1 THEN 624
622 PRINT RØ$" ";
624 PRINT U3$;:X = T(8): GOSUB 138: PRINT MID$ (E$, LEN (X$) -
    3,1Ø)X$
626 NEXT :S(V) = U: IF F = 1 THEN VTAB 24: HTAB 1: PRINT R$;: GET
    X$: GOTO 63Ø
628 PRINT CHR$ (12): PRINT : PRINT CHR$ (4)"PR#Ø"
63Ø F = Ø: GOTO 168
-----------------------------------------------------(k)
632 FOR X = 24576 TO 25575: READ Y: POKE X,Y: NEXT : RETURN:
    REM *** HIRES-SHAPES FUER GROSSBUCHSTABEN UND SONDERZEICHEN
634 DATA 63,Ø,128,Ø,133,Ø,143,Ø,156,Ø
636 DATA 175,Ø,192,Ø,2Ø8,Ø,224,Ø,235,Ø
638 DATA 248,Ø,5,1,23,1,35,1,41,1
64Ø DATA 5Ø,1,55,1,65,1,84,1,96,1
642 DATA 111,1,128,1,142,1,158,1,176,1
644 DATA 19Ø,1,2Ø8,1,222,1,228,1,236,1
646 DATA 249,1,4,2,17,2,3Ø,2,47,2
648 DATA 63,2,79,2,95,2,1Ø9,2,123,2
65Ø DATA 136,2,153,2,166,2,178,2,189,2
```

```
652 DATA 2Ø4,2,214,2,228,2,243,2,1,3
654 DATA 15,3,32,3,48,3,65,3,77,3
656 DATA 91,3,1Ø5,3,12Ø,3,136,3,147,3
658 DATA 162,3,179,3,197,3,214,3,9,9
66Ø DATA 9,1,Ø,73,4,32,36,52,18,18
662 DATA 1Ø,73,Ø,72,8,24,32,52,1Ø,33
664 DATA 52,18,18,1Ø,9,Ø,9,36,36,36
666 DATA 214,45,45,39,52,54,54,38,12,63
668 DATA 63,149,73,9,Ø,8,45,45,32,59
67Ø DATA 63,32,41,45,31,48,54,54,38,74
672 DATA 9,Ø,8,12,12,12,12,22Ø,59,46
674 DATA 148,146,9,53,39,74,1,Ø,8,1ØØ
676 DATA 12,36,59,5Ø,14,1Ø,14,12,22,31
678 DATA 47,9,9,Ø,72,8,8,24,36,22
68Ø DATA 18,18,9,9,Ø,73,28,28,36,12
682 DATA 12,2Ø,18,18,18,9,9,Ø,73,12
684 DATA 12,36,28,28,2Ø,18,18,18,9,9
686 DATA Ø,8,12,4,224,76,54,54,54,12
688 DATA 33,35,32,33,18,18,18,9,Ø,73
69Ø DATA 32,36,52,26,43,45,61,18,1Ø,9
692 DATA Ø,9,12,52,74,9,Ø,72,24,4Ø
694 DATA 45,61,18,1Ø,9,Ø,73,2Ø,9,9
696 DATA Ø,8,12,12,12,12,148,18,18,9
698 DATA Ø,8,36,36,12,45,21,54,54,3Ø
7ØØ DATA 63,71,32,33,33,18,18,1Ø,9,Ø
7Ø2 DATA 9,45,39,36,36,244,21,18,18,9
7Ø4 DATA 9,Ø,33,12,12,1Ø1,36,59,63,5Ø
7Ø6 DATA 18,18,45,45,15,9,Ø,9,45,5
7Ø8 DATA 32,28,47,32,33,63,63,157,146,5Ø
71Ø DATA 9,9,9,Ø,73,33,36,36,52,51
712 DATA 51,51,45,45,23,1Ø,9,Ø,8,14
714 DATA 45,5,32,36,59,63,36,45,45,151
716 DATA 146,1Ø,9,Ø,8,36,45,173,54,59
718 DATA 63,32,36,12,12,45,23,18,18,1Ø
72Ø DATA 9,Ø,9,36,12,12,12,6Ø,63,47
722 DATA 146,18,1Ø,9,9,Ø,8,36,32,12
724 DATA 45,21,54,5Ø,3Ø,63,7,32,12,45
726 DATA 21,18,9,Ø,41,45,32,33,36,28
728 DATA 63,23,54,41,45,18,74,Ø,64,64
73Ø DATA 177,86,73,Ø,9,12,196,8,22,86
732 DATA 73,Ø,9,9,28,28,28,12,12,12
734 DATA 148,146,82,9,Ø,64,45,45,199,216
736 DATA 45,45,151,82,9,Ø,9,12,12,12
738 DATA 28,28,28,148,146,82,73,1,Ø,73
74Ø DATA 4,32,12,12,28,219,98,45,149,146
742 DATA 74,Ø,73,9,63,63,32,36,1ØØ,45
744 DATA 21,54,55,39,44,146,74,1,Ø,33
746 DATA 36,52,8,48,64,14,14,54,54,196
748 DATA 219,45,173,74,Ø,41,45,223,36,44
75Ø DATA 45,223,36,44,45,79,5Ø,22,54,9
752 DATA Ø,9,7,48,32,36,36,121,45,21
754 DATA 148,146,3Ø,27,45,77,Ø,73,57,63
756 DATA 36,36,36,45,125,17,54,54,148,9
758 DATA Ø,33,36,36,44,45,189,18,63,151
76Ø DATA 42,45,125,9,Ø,33,36,36,44,45
762 DATA 189,18,63,151,74,73,1,Ø,8,6
764 DATA 32,36,1ØØ,15,45,189,146,15,53,62
766 DATA 63,79,73,1,Ø,33,36,36,18Ø,42
768 DATA 45,37,36,15Ø,54,38,74,Ø,9,37
77Ø DATA 36,36,6Ø,13,151,146,1Ø,79,9,Ø
772 DATA 8,14,45,79,32,36,36,15Ø,18,78
774 DATA Ø,33,36,36,116,72,49,51,51,51
776 DATA 161,1Ø,148,161,9,Ø,33,36,36,18Ø
778 DATA 146,42,45,61,73,Ø,33,36,36,116
78Ø DATA 14,1Ø2,48,64,54,54,54,84,1,Ø
782 DATA 33,36,36,18Ø,161,1Ø,148,97,36,52
784 DATA 18,54,166,9,Ø,8,36,36,12,45
786 DATA 21,54,54,3Ø,63,79,73,1,Ø,33
788 DATA 36,36,44,45,79,162,246,63,151,74
79Ø DATA 73,1,Ø,8,36,36,12,45,21,54
792 DATA 166,18,28,28,148,58,79,73,1,Ø
794 DATA 33,36,36,44,45,79,162,246,63,79
796 DATA 162,1Ø,148,161,9,Ø,8,14,45,5
798 DATA 32,28,63,7,48,32,12,45,21,148
8ØØ DATA 146,78,Ø,73,36,36,36,63,13,45
8Ø2 DATA 151,146,74,1,Ø,8,6,32,36,36
8Ø4 DATA 7Ø,73,54,54,246,63,77,9,Ø,64
8Ø6 DATA 36,36,15Ø,82,148,97,12,36,36,15Ø
8Ø8 DATA 146,9,Ø,33,36,36,18Ø,146,97,18Ø
81Ø DATA 97,36,36,15Ø,5Ø,166,9,Ø,8,1Ø2
812 DATA 48,64,12,68,23Ø,219,166,1Ø,148,113
814 DATA 14,166,9,Ø,73,36,36,227,18Ø,73
816 DATA 12,18Ø,146,74,Ø,8,12,12,12,12
818 DATA 6Ø,63,239,146,18,46,45,125,9,Ø
82Ø DATA 33,36,164,82,84,84,84,18,36,36
822 DATA 4,24Ø,28,3Ø,28,6,Ø,1,32,36
824 DATA 4,112,14Ø,14Ø,12,15Ø,18,36,164,146
826 DATA 26,212,212,4,Ø,1,32,36,1ØØ,14
828 DATA 8,142,146,36,36,148,146,26,212,212
83Ø DATA 4,Ø,73,36,36,36,3Ø,84,161,1Ø
832 DATA 212,219,4,Ø,Ø,Ø,Ø,Ø,Ø,Ø
```

EINZEL.GROSS
(Neben EINZEL.KLEIN als Textfile auf Sammeldisk)

```
------------------------------------------------------------(1)

A1    MEHRWERTSTEUERSATZ.....
A2    VERTRIEBSKOSTENSATZ/NP.
A3    WERBEKOSTENSATZ/NP.....
A4    GEMEINKOSTENSATZ/NP....
A5    GEWINNSATZ/NP..........
ØØ    LADENPREIS.............
Ø1    VERKAUFSAUFLAGE........
Ø2    FREIEXEMPLARE..........
Ø3    RABATTSATZ.............
Ø4    VERTRIEBSSTUECKKOSTEN..
Ø5    FESTER WERBEETAT.......
Ø6    FIXE HERSTELLKOSTEN....
Ø7    VAR.HERSTELLSTUECKKOST.
Ø8    HONORARSATZ/LP.........
Ø9    HONORARSATZ/NLP........
1Ø    HONORARSATZ/NP.........
11    HONORARSTUECKKOSTEN....
12    HONORARPAUSCHALE.......
13    HONORARSATZ/LP (ANR.)..
14    HONORARSATZ/NLP (ANR.).
15    HONORARSATZ/NP (ANR.)..
16    HONORARSTUECKKO. (ANR.)
17    HONORARGARANTIE (ANR.).
18    SONSTIGE KOSTEN........
19    SONSTIGE ERLOESE.......
ØØ    UMSATZ.................
Ø1  - VERTRIEBSKOSTEN.....
Ø2  - WERBEKOSTEN.........
Ø3  - HERSTELLUNGSKOSTEN..
Ø4  - HONORARKOSTEN.......
Ø5  - SONSTIGE KOSTEN.....
Ø6  + SONSTIGE ERLOESE....
Ø7  = DECKUNGSBEITRAG.....
Ø8    * DECKUNGSSPANNE.%..
Ø9  - GEMEINKOSTEN........
1Ø  = GEWINN..............
11    * GEWINNSPANNE.%.....
12    INVESTITIONSKOSTEN...
13    ABSATZHON. (ANR.)....
14    DS-LADENPREIS........
15    DS-VERKAUFSAUFLAGE...
16    MINDEST-LP  I........
17    MINDEST-LP II........
18    DECKUNGSAUFLAGE  I...
19    DECKUNGSAUFLAGE II...
```

EINZEL.KONST
(als Textfile auf Sammeldisk)

```
------------------------------------------------------------(m)

7
1Ø
1Ø
2Ø
5
5
```

Funktionenprogramm für Apple II+

von Stefan Hubertus Weil

1.1. Berechnung der Funktionswerte für beliebige Funktionen

Das Hauptproblem stellt zunächst die Eingabe beliebiger Funktionen und die Berechnung der zugehörigen Funktionswerte dar. Hier bieten sich auf den ersten Blick zwei grundverschiedene Möglichkeiten an:

1. Eingabe der kompletten Zeile für die Funktionswertberechnung über ein Textfile
2. Direkte Manipulation der betreffenden Programmzeile durch POKE - Anweisungen

Ich habe mich für die zweite Methode einschieden, da sie mir eleganter erschien.

Für die Funktionswertberechnung habe ich die DEF FN - Funktion gewählt, da sie eine einfache Möglichkeit darstellt, Funktionswerte ohne Sprung in eine andere Programmzeile zu berechnen. Dies ist vor Allem beim Zeichnen der Ableitungsfunktion sehr vorteilhaft.

1.2. Absicherung für nicht definierte Stellen der Funktionen

Die Absicherung für nicht definierte Stellen der Funktionen ist nicht sehr schwierig. Am einfachsten läßt sich dieses Problem mit der ONERR GOTO Funktion lösen. Bei Auftreten eines Fehlers (z.B. ILLEGAL QUANTITY ERROR) wird einfach der nächste Funktionswert berechnet.

1.3. Darstellung der Funktionen in beliebigen Intervallen

Auf den ersten Blick stellt dies kein sehr großes Problem dar. Um eine Funktion zeichnen zu können, müssen vorher folgende Variablen berechnet werden:

1. Streckfaktor in X - Richtung (Punkte je Einheit)
2. dto. für Y - Richtung
3. Koordinaten für X - und Y - Achse

Der Programmteil "Funktionsgraph zeichnen" hat wesentlich mehr Zeit in Anspruch genommen als vorgesehen. Dies liegt vor allem an dem Programmteil zum Zeichnen des Koordinatensystems, da es hier viele Möglichkeiten für den Ausschnitt aus dem Koordinatensystem gibt.

1.4. Speicherbelegung durch Programm, Variablen und Graphik

Das Problem der Überschneidung von Programm bzw. Variablen und Graphik ergab sich erst beim Programmieren. Erste Schwierigkeiten ergaben sich, als die Variablen in den HGR 2 Bereich hineinragten. Dies konnte allerdings durch Verschieben von LOMEM verhindert werden. Durch weitere Ergänzungen reichen die Programmzeilen nun bis knapp unter den HGR 2 Bereich. Eine Programmerweiterung ist deshalb mit der normalen Speicheraufteilung nicht mehr möglich.

Das Programm müßte in den Bereich oberhalb von HGR (ab $4000) geladen werden.

2.1. Variablenliste

a) Feldvariablen

GL	Feld für den codierten Funktionsterm
A$,B$	Felder für Menütexte

b) Textvariablen

BF$,WS$	
K$,A3$	häufig vorkommende Texte
A4$	
D$	CTRL-D für DOS Anweisungen
W$	für ja/nein Abfragen und Menüauswahl
A$	allgemeine Verwendung
F$	Funktionsterm nach der Codierung und überprüfung
Y$	Eingegebener Funktionsterm
U$	allgemeine Verwendung
N$	Filename
NF$	neuer Filename

c) Gleitkommavariablen

I	Zähler in FOR - NEXT Schleifen
P	entspricht Pi (3.14159265)
E	entspricht e (2.71828138)
EP	Differenz in X - Richtung zur Berechnung der Ableitung
EQ	2 * EP
EX	minimales X oder Y Intervall
EE	minimales eingeschränktes X Intervall
W	Abfragen , Menü
A	allgemeine Verwendung
LE	Länge von Y$
GL	Zeiger auf nächste Position im Feld GL
Z	FOR NEXT bei Codierung , Variable für Einheitsberechnung
FZ	Flag für erkannte Funktion / Zeichen
FF	Flag für eingegebene Funktionsgleichung
FX	Flag für geändertes X Intervall
FY	dto. für Y
XF	Flag für eingegebenes X Intervall
YF	dto. für Y
F1	Flag für gezeichneten Funktionsgraphen
FG	Flag für Punkte (0) und Linien (1)
FR	Error-Flag beim Zeichnen
IN	inverse-Flag für Druckroutine
XC,XD	eingegebene Begrenzung des X Intervalls
YC,YD	dto. für Y
X1,X2	altes X Intervall
Y1,Y2	dto. für Y
X3,X4	X Intervall zum Zeichnen (eventuell eingeschränkt)
MX	Bildschirmkoordinate für X-Achse
MY	dto. für Y-Achse
SX	Streckfaktor in X-Richtung (Punkte/Einheit)
SY	dto. für Y
XE	X-Einheit
YE	Y-Einheit
XM	X-Bildschirmkoordinate des Punktes
YM	dto. für Y
X5,X6	X Intervall für Einheitenberechnung
Y5,Y6	dto.für Y
XA,XB	X Intervall für Wertetabelle
S	Schrittweite für Zeichnen und Wertetabelle
R	= 1/S (zum Runden der X-Werte für Wertetabelle)

2.2. Liste der wichtigen Unterprogramme

Zeile	Unterprogramm
4010	viermal Piepston ausgeben
4020	Warteschleife
4030	Ausdruck:"Exit : <RETURN>"
4200	Berechnung der X-Einheit
4320	dto. für Y
4910	Bildschirmorganisation NORMAL/INVERSE
4940	dto. für Punkte/Linien
4970	Abfrage: "Drucker bereit"

2.3. Grober Programmaufbau

Der Aufbau des Programms ist relativ einfach. Am Anfang des Programms befindet sich die DEF FN Zeile für die Funktionswertberechnung. Diese Zeile steht deshalb am Anfang, weil eine Programmänderung vor dieser Zeile eine Verschiebung der Adressen für die POKE - Anweisungen zur Folge hätte. Dies ware bei der Programmentwicklung sehr hinderlich. Dann folgen einige Vorbereitungen wie z.B. das Verschieben von LOMEM und das Laden des Maschinenunterprogramms.

Liste der Hauptprogrammteile:

Zeilen	Programmteil
220 - 380	Hauptmenü
410 - 490	Eingabe der Funktionsgleichung
510 - 560	entfernen der Leerzeichen aus dem eingegebenen Funktionsterm
560 - 720	Codierung der Funktionsgleichung
770 - 820	Einbindung des Funktionsterms in den BASIC - Text und Aufruf der DEF FN - Funktion
830 - 880	Test der Funktionsgleichung auf SYNTAX ERROR
890 - 1040	Eingabe des X Intervalls
1050 - 1190	dto. für Y
1200 - 1230	Fehlerbehandlung für 890 - 1190
1240 - 1470	Menü "Funktionsgraph zeichnen"
1480 - 1530	überprüfung, ob X und Y Intervall und Funktionsgleichung eingegeben wurden
1540 - 1580	überprüfung, ob ggf. in das alte Koordinatensystem gezeichnet werden kann
1590 - 1660	Eingabe des eingeschränkten Intervalls
1670 - 1750	Menü Funktionsgraph bzw. Ableitung zeichnen
1760 - 1810	Vorbereitungen zum Zeichnen
1820 - 2150	Koordinatensystem zeichnen
2160 - 2300	Y zeichnen
2310 - 2330	Aufruf der Maschinenunterprogramme
2340 - 2450	Y' zeichnen
2460 - 2470	Funktionsgraph anzeigen
2480 - 2630	Größen anzeigen
2640 - 3010	Wertetabelle
3020 - 3120	Menü "Diskettenoperationen"
3130 - 3290	Teil der Diskettenroutinen
3310 - 3380	Funktionsgleichung laden
3390 - 3500	Funktionsgleichung speichern
3510 - 3570	Funktionsgraph laden
3580 - 3710	Funktionsgraph speichern
3720 - 3770	File löschen
3780 - 3950	Filenamen ändern
3979 - 3990	DATA Liste für Funktionen, Zeichen und deren Code
4000 - 4180	Unterprogramme und Programmteile für Fehlerbehandlung
4190 - 4410	Einheitsberechnung
4420 - 4890	Druckroutinen

2.4. Speicherbelegung

BASIC - Text : $801 - $3F07
Graphik : $4000 - $5FFF und $D000 - $EFFF (16 K Karte)
Variablen : $6000 - ? und ? - $9600
Maschienenprogramme : $300 - $39B

2.5. Eingabe und Codierung der Funktionsgleichung

Der eingegebene Funktionsterm ist unter Y$ gespeichert. Zunächst werden alle Leerzeichen, die der Funktionsterm enthalten könnte, entfernt (Z. 510 - 560).

```
510  IF  RIGHT$ (Y$,1) = " " THEN Y$ =  LEFT$ (Y$, LEN (Y$) - 1): GOTO 51
     0
520  IF  LEFT$ (Y$,1) = " " THEN Y$ =  RIGHT$ (Y$, LEN (Y$) - 1): GOTO 52
     0
530  FOR I = 2 TO  LEN (Y$) - 1
540  IF  MID$ (Y$,I,1) <  > " " THEN 560
550 Y$ =  LEFT$ (Y$,I - 1) +  RIGHT$ (Y$, LEN (Y$) - I):I = I - 1
560  NEXT I
570 GL = 0:LE =  LEN (Y$)
580  FOR I = 1 TO LE
590 GL = GL + 1:FZ = 0: RESTORE
600 B$ =  MID$ (Y$,I,3)
610  FOR Z = 1 TO 11
620  READ A$,A: IF A$ <  > B$ THEN 640
630 GL(GL) = A:FZ = 1:I = I + 2:Z = 11
640  NEXT Z
650  IF FZ = 1 THEN 720
660 B$ =  MID$ (Y$,I,1)
670  FOR Z = 1 TO 21
680  READ A$,A: IF A$ <  > B$ THEN 700
690 GL(GL) = A:FZ = 1:Z = 21
700  NEXT Z
710  IF FZ = 0 THEN  RESTORE : GOTO 750
720  NEXT I
730  IF GL > 90 THEN  GOSUB 4010: HOME : VTAB 11: PRINT "Die Funktion ist
     "GL - 90: PRINT : PRINT "Zeichen / Funktionen zu lang!": GOSUB 4020:
     RESTORE : HOME : GOTO 410
740  RESTORE : GOTO 780
750  HOME : GOSUB 4010: VTAB 10
760  PRINT "Verwenden Sie nur die Zeichen und": PRINT : PRINT "Funktionen
    ,die Sie auf der unteren": PRINT : PRINT "Hälfte des Bildschirms sehe
    n !": GOSUB 4020: HOME : GOTO 410
```

Die Codierung der Funktionsgleichung geht folgendermaßen vor sich:

Zunächst wird GL, der Zeiger für den nächsten freien Platz im Feld GL, auf 0 gesetzt. I zeigt immer auf die augenblickliche Position in Y$, von wo an der nächste Teil der Funktionsgleichung codiert werden soll. In A$ ist immer die Funktion, bzw. das Zeichen gespeichert, das mit dem Teil der Funktionsgleichung (B$) verglichen wird. FZ gibt an ,ob nach einem Durchlauf eine Funktion bzw. ein Zeichen erkannt wurde (FZ=1), oder nicht (FZ=0).

Zuerst werden die ersten drei Zeichen des Funktionsterms mit allen Funktionen der Data - Liste verglichen. Stimmt B$ mit einer Funktion überein, dann wird der zugehörige Code, der in A gespeichert ist, im Feld GL an der Position GL abgelegt. Außerdem wird I um 2 erhöht, damit nach dem NEXT I I wieder auf die

erste noch nicht codierte Stelle in Y$ zeigt. FZ wird auf 1 und Z auf 11 gesetzt. Dadurch ist die innere FOR NEXT - Schleife beendet und da FZ=1 ist, erfolgt sofort ein Sprung auf das NEXT I in Zeile 720.

Wenn keine Funktion der Data - Liste mit B$ übereinstimmt, wird B$ mit allen Zeichen aus der Data - Liste verglichen. Dies geschieht genau wie bei den Funktionen, außer, daß I nicht erhöht werden muß.

Ist nach dem Durchlauf der beiden inneren FOR NEXT - Schleifen keine Funktion und kein Zeichen erkannt worden (FZ=0), so wird eine entsprechende Fehlermeldung ausgegeben.

Wurden alle Zeichen und Funktionen des Funktionsterms erkannt und der zugehörige Code jeweils im Feld GL abgelegt, dann werden die Werte ab Speicherplatz 2093 in den Speicher übertragen. In den verbleibenden Speicherplätzen bis 2183 wird der Code 58 für die Doppelpunkte, die als Platzhalter dienen, übertragen (siehe Zeile 30).

Das Feld GL wird von GL(1) bis GL(GL) in den Bereich nach dem Gleichheitszeichen in Zeile 30 übertragen. Anschließend wird die Zeile 30 durchlaufen, um die neue Funktion mit FN F(X) aufrufen zu können. Dann wird die Funktion noch auf einen Syntaxfehler überprüft.

Auf der nächsten Seite finden Sie einen Speicherauszug, der die Speicherung von BASIC Programmen verdeutlicht.

Speicherauszug (Zeile 30):

```
822- B8         Sprungadresse für nächste Zeile (LSB)
823- 08         dto. MSB
824- 1E         Zeilennummer (LSB)
825- 00         dto. MSB
826- B8         DEF
827- C2         FN
828- 46         F
829- 28         (
82A- 58         X
82B- 29         )
82C- D0         =
82D- DF         SIN     Anfangsadresse für POKE - Anweisungen
82E- 28         (
82F- 58         X
830- 29         )
831- 3A         Doppelpunkt (Platzhalter)
832- 3A         dto.
   .
   .
   .
886- 3A         dto.
887- 00         Ende der Zeile
```

2.6. Verwendung der ONERR GOTO Funktion

Die ONERR GOTO Funktion wird zu folgenden Dingen benötigt:

a) Überprüfung der Funktionsgleichung auf Syntaxfehler

Nachdem der Funktionsterm durch POKE Anweisungen in das Programm eingefügt wurde, und die DEF FN Funktion aufgerufen wurde, wird zur Probe FN F(1) berechnet. Tritt dabei der Fehler SYNTAX ERROR auf (PEEK(222)=16), dann wird eine entsprechende Fehlermeldung ausgegeben. Ein Test auf OVERFLOW ERROR (z.B. Y=1E40) ist nicht möglich, da dies für alle X geschehen müßte.

b) Absicherung für nicht definierte Stellen einer Funktion

Wenn bei einer Funktionswertberechnung ein Fehler (z.B. ILLEGAL QUANTITY oder DIVISION BY ZERO ERROR) auftritt, wird dieser Funktionswert übergangen und der nächste Funktionswert berechnet.

c) Absicherung für Punkte, die außerhalb des Y Intervalls liegen

Liegt ein Punkt der Funktion außerhalb des Y Intervalls (YM>191 oder YM<0), so tritt beim Plotten der Fehler ILLEGAL QUANTITY ERROR auf. In diesem Fall wird wie unter b) einfach der nächste Funktionswert berechnet. Außerdem wird das ERROR Flag (FR) auf 1 gesetzt, damit beim nächsten Punkt innerhalb des Y Intervalls keine falsche Linie gezogen wird.

d) Unterbrechung des Zeichnens und der Wertetabelle

Durch Eingabe von CTRL-C wird ein Fehler mit dem Code 255 erzeugt. Tritt dieser Fehler beim Zeichnen oder bei der Funktionswertberechnung im Programmteil "Wertetabelle" auf, erfolgt ein Sprung zum Hauptmenü.

e) Erkennung falscher Eingaben im Programmteil "Diskettenoperationen"

Durch Erkennung der Fehlermeldungen anhand der Fehlercodes ist es möglich, eine genaue Fehlermeldung für den Anwender auszugeben. Dies kann z.B. die Fehlermeldung "Es existiert kein File mit dem Namen ...." sein.

3.1. Menüstruktur

Menü "Druckroutinen"

↕

Hauptmenü <-> Menü "Funktionsgraph zeichnen" <-> Menü "Y/Y' zeichnen"

↕

Menü "Diskettenoperationen"

3.2. Einsatzmöglichkeiten des Programms

Das Programm ermöglicht folgende Funktionen:

a) Zeichnen einer beliebigen Funktion in einem beliebigen Intervall

b) Zeichnen der 1. Ableitung der Funktion

c) Zeichnen mehrerer Funktionen und/oder deren Ableitung in ein Koordinatensystem

d) Zeichnen abschnittsweise definierter Funktionen durch Einschränken des X Intervalls vor dem Zeichnen

e) Zwischenspeicherung des Funktionsgraphen in der 16K Karte

f) Erstellen einer Wertetabelle zu der Funktion

g) Speicherung des Funktionsgraphen und der Funktionsgleichung auf Diskette

h) Ausdrucken des Funktionsgraphen und des X/Y Intervalls

i) Vertauschen des Funktionsgraphen in der 16K Karte mit dem im HGR 2 Bereich durch Drücken der Taste 'C' im Modus "Funktionsgraph anzeigen"

Anmerkung: Es können natürlich nur die Funktionen gezeichnet werden, die sich in Applesoft - BASIC formulieren lassen und deren Länge 90 Zeichen bzw. Funktionen nicht überschreitet

3.3. Grenzen des Programms

Dem Programm sind Grenzen durch die Genauigkeit des Applesoft - BASIC gesetzt. Deshalb können Funktionen nicht in beliebig kleinen Intervallen betrachtet werden. Ebenso können Funktionen nur im Bereich von -1E38 bis +1E38 betrachtet werden. Dies ist für normale Anwendungen jedoch ausreichend. Die Auflösung des Monitors ist auf 280 * 192 Punkte beschränkt. Dies hat zur Folge, daß sehr eng schwingende Funktionen (z.B. SIN(1/X)) nicht richtig gezeichnet werden können.

Beim zeichnen kommt es in der Regel zu Abweichungen von einem Punkt.

3.4. Behandlung falscher Eingaben

Das Programm ist gegen folgende falschen Eingaben abgesichert:

- Eingabe von fehlerhaften Funktionsgleichungen
- Eingabe eines zu kleinen oder zu großen X oder Y Intervalls
- Eingabe einer zu großen oder zu kleinen Schrittweite im Programmteil "Wertetabelle"
- Eingabe falscher Zeichen bei der Menüabfrage
- Zeichnen des Funktionsgraphen, wenn das X oder Y Intervall oder die Funktionsgleichung fehlt
- Zeichnen des Funktionsgraphen in das alte Koordinatensystem, wenn die Achsenabschnitte geändert wurden
- Veränderung (Löschen, überschreiben, ...) geschützter Files
- Laden nicht vorhandener Files
- Speicherung von Funktionsgraph und Funktionsgleichung auf voller Diskette
- Eingabe eines zu langen Filenamens

3.5. Ungeklärte Probleme

Gelegentlich tritt beim Programmablauf der Fehler OUT OF MEMORY ERROR auf. Der Grund dafür könnte z.B. eine nicht abgeschlossene FOR NEXT Schleife sein, deren Rücksprungadresse, Step u.s.w. nicht gelöscht werden.

Tritt dieser Fehler auf, ist das Programm mit CTRL-RESET und GOTO 230 wieder zu starten. Die Variablen bleiben dabei erhalten.

3.6. Verbesserungsmöglichkeiten

Aufgrund der vielen Eingabekombinationsmöglichkeiten ist es fast unmöglich, das Programm gegen alle falschen Eingaben abzusichern. Ich habe mich bemüht, das Programm an den wichtigsten Stellen gegen Fehler abzusichern.

Das Programm könnte an einigen Stellen noch verbessert werden. Zum Beispiel könnte der INPUT Befehl bei der Eingabe der Funktionsgleichung durch einen Eingabeteil mit GET Anweisungen ersetzt werden. Außerdem könnten die Diskettenroutinen noch benutzerfreundlicher gestaltet werden. Dies gilt vor allem bei der Fehlermeldung "Es existiert bereits ein File mit dem Namen... ". Hier sollte ein überschreiben des Files mit einem Tastendruck möglich sein.

Grenzen setzt hier leider der zur Verfügung stehende Speicherplatz.

Applesoft-Programm FUNKTIONEN

```
10  REM  (C) BY STEFAN WEIL
20  GOTO 50
30  DEF  FN F(X) =  SIN (X):::::::::::::::::::::::::::::::::::::::::::::::::
    :::::::::::::::::::::::::::::::::::::::::::
40  RETURN
50  LOMEM: 24580
60  HOME
70  HGR2
80  TEXT
90  DIM GL(150)
100 P = 3.14159265
110 E = 2.71828183
120 EP = .00001:EQ = 2 * EP:EE = .000001:EX = .00001
130 FG = 1
140 BF$ = "Der Bereich ist zu klein!":WS$ = "Was möchten Sie ? "
150 K$ = "Koordinatensystem"
160 A3$ = "Funktionsgraph":A4$ = "Funktionsgleichung"
170 A$(1) = A4$ + " laden":A$(2) = A4$ + " speichern":A$(3) = A3$ + " lad
    en":A$(4) = A3$ + " speichern":A$(5) = "File löschen":A$(6) = "Filena
    men ändern":A$(7) = "File schützen (LOCK)":A$(8) = "CATALOG"
180 B$(1) = A4$ + " eingeben":B$(2) = "X-Achsenabschnitt wählen":B$(3) =
    "Y-Achsenabschnitt wählen":B$(4) = A3$ + " zeichnen":B$(5) = A3$ + "
    (-en) anzeigen":B$(6) = "Eingabegrößen auflisten":B$(7) = "Wertetabel
    le erstellen"
190 B$(8) = "Druckroutinen":B$(9) = "Diskettenoperationen"
200 D$ =  CHR$ (4)
210  PRINT D$"BLOADFUNKTIONEN EXC,A$300"
220  REM ---------- Hauptmenü ----------
230  POKE 34,0: POKE 216,0
240  POKE 35,24: HOME
250  INVERSE : PRINT "              FUNKTIONEN                ": NORMAL
260  POKE 34,2
270  VTAB 5: PRINT "Ihnen stehen folgende Operationen       zur Verfügung
    :"
280  PRINT : PRINT
290  FOR I = 1 TO 9
300  PRINT "   <";: INVERSE : PRINT I;: NORMAL : PRINT "> "B$(I)
310  NEXT I
320  VTAB 24: PRINT "Ende:<";: INVERSE : PRINT "CTRL-E";: NORMAL : PRINT
    ")";: VTAB 19
330  PRINT : PRINT "   "WS$;
340  GET W$
350  IF  ASC (W$) = 5 THEN  POKE 34,0: VTAB 24: SPEED= 190: FOR I = 1 TO
    24: PRINT : NEXT I: SPEED= 130: PRINT "(C) BY STEFAN WEIL";: VTAB 1: SPEED=
    255: END
360 W =  VAL (W$): IF W < 1 THEN 340
370  HOME
380  ON W GOTO 410,900,1050,1240,390,2510,2650,4430,3030
390  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0: GET W$: IF  ASC (
    W$) = 67 THEN  CALL 768: GOTO 390
400  TEXT : GOTO 230
410  REM ---------- Eingabe Funktionsgleichung ----------
420  VTAB 23: GOSUB 4030
430  FOR I = 11 TO 21: VTAB I: READ A$,A: HTAB 6: PRINT A$;: NEXT I
440  FOR I = 11 TO 21: VTAB I: READ A$,A: HTAB 19: PRINT A$;: NEXT I
450  FOR I = 11 TO 20: VTAB I: READ A$,A: HTAB 30: PRINT A$: NEXT I
460  VTAB 18: HTAB 21: PRINT "(=Pi)": HTAB 21: PRINT "(=e)": RESTORE
470  VTAB 4: PRINT "Geben Sie die Funktionsgleichung ein:"
480  VTAB 7: PRINT "Y = "F$"
490  VTAB 7: HTAB 5: INPUT "";Y$: IF Y$ = "" THEN 230
500  REM ---------- codieren ----------
510  IF  RIGHT$ (Y$,1) = " " THEN Y$ =  LEFT$ (Y$, LEN (Y$) - 1): GOTO 51
    0
520  IF  LEFT$ (Y$,1) = " " THEN Y$ =  RIGHT$ (Y$, LEN (Y$) - 1): GOTO 52
    0
530  FOR I = 2 TO  LEN (Y$) - 1
540  IF  MID$ (Y$,I,1) <  > " " THEN 560
550 Y$ =  LEFT$ (Y$,I - 1) +  RIGHT$ (Y$, LEN (Y$) - I):I = I - 1
560  NEXT I
570 GL = 0:LE =  LEN (Y$)
580  FOR I = 1 TO LE
590 GL = GL + 1:FZ = 0: RESTORE
600 B$ =  MID$ (Y$,I,3)
610  FOR Z = 1 TO 11
620  READ A$,A: IF A$ <  > B$ THEN 640
630 GL(GL) = A:FZ = 1:I = I + 2:Z = 11
640  NEXT Z
650  IF FZ = 1 THEN 720
660 B$ =  MID$ (Y$,I,1)
670  FOR Z = 1 TO 21
680  READ A$,A: IF A$ <  > B$ THEN 700
690 GL(GL) = A:FZ = 1:Z = 21
700  NEXT Z
710  IF FZ = 0 THEN  RESTORE : GOTO 750
720  NEXT I
730  IF GL > 90 THEN  GOSUB 4010: HOME : VTAB 11: PRINT "Die Funktion ist
     "GL - 90: PRINT : PRINT "Zeichen / Funktionen zu lang!": GOSUB 4020:
     RESTORE : HOME : GOTO 410
740  RESTORE : GOTO 780
750  HOME : GOSUB 4010: VTAB 10
760  PRINT "Verwenden Sie nur die Zeichen und": PRINT : PRINT "Funktionen
    ,die Sie auf der unteren": PRINT : PRINT "Hälfte des Bildschirms sehe
    n !": GOSUB 4020: HOME : GOTO 410
770  REM ---------- Funktionsterm in Programm einbinden ----------
780  FOR I = 1 TO GL
790  POKE I + 2092,GL(I)
800  NEXT I
810  FOR Z = I TO 90: POKE Z + 2092,58: NEXT Z
820  GOSUB 30:F$ = Y$
830  REM ---------- Test auf SYNTAX ERROR ----------
840  ONERR  GOTO 860
850 A =  FN F(1): POKE 216,0:FF = 1: GOTO 230
860  IF  PEEK (222) <  > 16 THEN  POKE 216,0:FF = 1: GOTO 230
870 F$ = ""
880  HOME : GOSUB 4010: VTAB 12: PRINT "Die Funktion ist fehlerhaft defin
    iert !": GOSUB 4020: POKE 216,0: HOME :F(0) = 0: GOTO 410
890  REM ---------- X- u. Y-Intervall eingeben ----------
900  HTAB 8: PRINT B$(2)
910  VTAB 6: PRINT "Geben Sie die Begrenzungen für X an!"
920  ONERR  GOTO 1200
930  VTAB 10: HTAB 8: INPUT "X min = ";XC
940  ONERR  GOTO 1210
950  VTAB 14: HTAB 8: INPUT "X max = ";XD
960  POKE 216,0
970  IF XC >  = XD THEN 1010
980  IF  - XC + XD < EX THEN  VTAB 18: HTAB 6: GOSUB 4010: PRINT BF$: GOSUB
    4020: HOME : GOTO 900
990 X5 = XC:X6 = XD: GOSUB 4200
1000 FX = 1:XF = 1: GOTO 230
1010  GOSUB 4010
1020  PRINT : PRINT : PRINT : PRINT "X min muß kleiner als X max sein!"
1030  GOSUB 4020
1040  HOME : GOTO 900
1050  HTAB 8: PRINT B$(3)
1060  VTAB 6: PRINT "Geben Sie die Begrenzungen für Y an!"
1070  ONERR  GOTO 1220
1080  VTAB 10: HTAB 8: INPUT "Y min = ";YC
1090  ONERR  GOTO 1230
1100  VTAB 14: HTAB 8: INPUT "Y max = ";YD
1110  POKE 216,0
1120  IF YC >  = YD THEN 1160
1130  IF  - YC + YD < EX THEN  GOSUB 4010: VTAB 18: HTAB 8: PRINT BF$: GOSUB
    4020: HOME : GOTO 1050
1140 Y5 = YC:Y6 = YD: GOSUB 4320
1150 FY = 1:YF = 1: GOTO 230
1160  GOSUB 4010
1170  PRINT : PRINT : PRINT : PRINT "Y min muß kleiner als Y max sein!"
1180  GOSUB 4020
1190  HOME : GOTO 1050
1200  VTAB 10: CALL  - 958: GOTO 930
1210  VTAB 14: CALL  - 958: GOTO 950
1220  VTAB 10: CALL  - 958: GOTO 1080
1230  VTAB 14: CALL  - 958: GOTO 1100
1240  HOME : POKE 34,2
1250  REM ---------- Funktion zeichnen ----------
1260  VTAB 1: INVERSE : PRINT "          FUNKTIONSGRAPH ZEICHNEN          "
    NORMAL
1270  VTAB 7
1280  PRINT "     <1> in altes "K$
1290  PRINT "     <2> in neues "K$
1300  PRINT "     <3> Bereich 1 --> Bereich 2"
1310  PRINT "     <4> Bereich 2 --> Bereich 1"
1320  PRINT "     <5> Bereich 1 <-> Bereich 2"
1330  PRINT "     <6> "B$(5)
1340  PRINT "     <7> "B$(6)
1350  INVERSE
1360  FOR I = 7 TO 13
1370  VTAB I: HTAB 6: PRINT I - 6
1380  NEXT I
1390  NORMAL
1400  VTAB 23: GOSUB 4030: VTAB 17
1410  PRINT "     Was möchten Sie ? ";
1420 AB = 0
1430  GET W$:W =  VAL (W$)
1440  IF  ASC (W$) = 13 THEN 230
1450  IF W < 1 OR W > 7 THEN 1430
1460  HOME
1470  ON W GOTO 1480,1480,2310,2320,2330,2460,2480
1480  IF FF <  > 0 AND XF <  > 0 AND YF <  > 0 THEN 1540
1490  GOSUB 4010: PRINT : PRINT "Als Voraussetzung fehlt:": PRINT : PRINT
    : PRINT
1500  IF FF = 0 THEN  HTAB 14: PRINT "Die Funktion Y=f(X)": PRINT : PRINT

1510  IF XF = 0 THEN  HTAB 14: PRINT "Der X-Achsenabschnitt": PRINT : PRINT

1520  IF YF = 0 THEN  HTAB 14: PRINT "Der Y-Achsenabschnitt"
1530  GOSUB 4020: GOTO 1240
1540  IF F1 = 0 THEN 1570
1550  IF FX = 0 AND FY = 0 THEN 1570
1560  IF W = 1 THEN 4080
1570 X1 = XC:X2 = XD:Y1 = YC:Y2 = YD:X3 = XC:X4 = XD
1580  HOME : GOTO 1680
1590  HOME : VTAB 5: HTAB 6: PRINT "X min = "X1: VTAB 8: HTAB 6: PRINT "X
     max = ";X2
1600  VTAB 23: GOSUB 4030: VTAB 11: HTAB 8: PRINT "Geben Sie X1 und X2 ei
    n:"
```

```
1610  VTAB 14: HTAB 6: INPUT "X1 = ";U$: IF U$ = "" THEN 1670
1620 X3 =  VAL (U$): VTAB 16: HTAB 6: INPUT "X2 = ";U$: IF U$ = "" THEN X
    3 = XC: GOTO 1670
1630  PRINT : PRINT
1640 X4 =  VAL (U$): IF X3 >  = X4 THEN  GOSUB 4010: PRINT "X1 muß kleine
    r als X2 sein!": GOSUB 4020: GOTO 1590
1650  IF X3 < X1 OR X4 > X2 THEN  GOSUB 4010: PRINT "Das Intervall muß in
    nerhalb von X min   und X max liegen!": GOSUB 4020: GOTO 1590
1660  IF  - X3 + X4 < EE THEN  GOSUB 4010: PRINT BF$: GOSUB 4020: GOTO 15
    90
1670  HOME : VTAB 1: INVERSE : PRINT "         FUNKTIONSGRAPH ZEICHNEN
        ": NORMAL : POKE 34,2
1680  VTAB 7: HTAB 11: PRINT "<1> Y  zeichnen": VTAB 9: HTAB 11: PRINT "<
    2> Y' zeichnen": VTAB 11: HTAB 11: PRINT "<3> X-Intervall einschränke
    n": VTAB 13: HTAB 11: PRINT "<4> Punkte <-> Linien"
1690  VTAB 18: HTAB 11: PRINT WS$
1700  VTAB 23: GOSUB 4030: GOSUB 4940
1710  GET A$:A =  VAL (A$): IF A$ =  CHR$ (13) THEN 1240
1720  IF A = 4 THEN FG =  NOT FG: GOSUB 4940: GOTO 1710
1730  IF A = 3 THEN 1590
1740  IF A = 2 AND  - X1 + X2 < .001 THEN  HOME : VTAB 10: GOSUB 4010: PRINT
    "Der Bereich ist zu klein um die": PRINT : PRINT "Ableitung zu zeichn
    en!": GOSUB 4020: GOTO 1670
1750  IF A > 3 OR A < 1 THEN 1710
1760  POKE 230,64
1770 MX = SX *  - X1:MY = SY * Y2
1780  IF W <  > 1 OR F1 = 0 THEN  CALL  - 3086
1790 S = 1 / SX:R = SX
1800  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0: HCOLOR= 3
1810  IF W = 1 AND F1 = 1 THEN 2170
1820  REM ---------- Koordinatensystem zeichnen ----------
1830  IF MX < 0 OR MX > 278 THEN 1850
1840  HPLOT MX,0 TO MX,191
1850  IF MX > 274 OR MY < 5 OR MY > 185 THEN 1870
1860  HPLOT 278,MY + 1: HPLOT 277,MY + 2: HPLOT 276,MY + 3: HPLOT 278,MY -
    1: HPLOT 277,MY - 2: HPLOT 276,MY - 3
1870  IF MY < 2 OR MY > 188 THEN 2010
1880  IF MX > 278 THEN 1960
1890 Z = MX
1900  IF Z < 0 THEN Z = ( INT ( - (Z / (XE * SX))) + 1) * (XE * SX) + Z
1910  HPLOT Z,MY - 2 TO Z,MY + 2
1920 Z = Z + XE * SX
1930  IF Z > 272 THEN 1950
1940  GOTO 1910
1950  IF MX < 0 THEN 2010
1960 Z = MX
1970  IF Z > 297 THEN Z = ( INT ( - (Z / (YE * SY))) + 1) * (YE * SY) + Z

1980  HPLOT Z,MY - 2 TO Z,MY + 2
1990 Z = Z - XE * SX: IF Z < 0 THEN 2010
2000  GOTO 1980
2010  IF MY < 0 OR MY > 191 THEN 2030
2020  HPLOT 0,MY TO 279,MY
2030  IF MY < 5 OR MX < 5 OR MX > 274 THEN 2050
2040  HPLOT MX - 1,1: HPLOT MX - 2,2: HPLOT MX - 3,3: HPLOT MX + 1,1: HPLOT
    MX + 2,2: HPLOT MX + 3,3
2050  IF MX < 2 OR MX > 276 THEN 2170
2060  IF MY < 5 THEN 2130
2070 Z = MY
2080  IF Z > 191 THEN Z = Z - SY * YE: GOTO 2080
2090  HPLOT MX - 2,Z TO MX + 2,Z
2100 Z = Z - SY * YE: IF Z < 5 THEN 2120
2110  GOTO 2090
2120  IF MY > 185 THEN 2170
2130 Z = MY
2140 Z = Z + SY * YE: IF Z > 190 THEN 2170
2150  HPLOT MX - 2,Z TO MX + 2,Z: GOTO 2140
2160  REM ---------- Y zeichnen ----------
2170  ONERR  GOTO 2270
2180 FR = 1:X = X3
2190  IF A = 2 THEN 2350
2200 Y =  FN F(X)
2210 XM = SX * X + .5 + MX:YM = MY - SY * Y
2220  IF FG = 0 OR FR = 1 THEN FR = 0: HPLOT XM,YM: GOTO 2240
2230  HPLOT  TO XM,YM
2240 X = X + S
2250  IF X > X4 THEN  TEXT :F1 = 1:FX = 0:FY = 0: POKE 216,0:W = 1: GOTO
    1670
2260  GOTO 2200
2270 FR = 1:X = X + S
2280  IF  PEEK (222) = 255 THEN X = X4 + 1
2290  IF X > X4 THEN  TEXT :F1 = 1:FX = 0:FY = 0: POKE 216,0:W = 1: GOTO
    1670
2300  GOTO 2200
2310  CALL 820: GOTO 1240
2320  CALL 872: GOTO 1240
2330  CALL 768: GOTO 1240
2340  REM ---------- Y' zeichnen ----------
2350  ONERR  GOTO 2420
2360 Y = ( FN F(X + EP) -  FN F(X - EP)) / EQ
2370 XM = SX * X + .5 + MX:YM = MY - SY * Y
2380  IF FG = 0 OR FR = 1 THEN FR = 0: HPLOT XM,YM: GOTO 2400
2390  HPLOT  TO XM,YM
2400 X = X + S: IF X > X4 THEN  TEXT :F1 = 1:FX = 0:FY = 0: POKE 216,0:W =
    1: GOTO 1670
2410  GOTO 2360
2420 FR = 1:X = X + S
2430  IF  PEEK (222) = 255 THEN X = X4 + 1
2440  IF X > X4 THEN  TEXT :F1 = 1:FX = 0:FY = 0: POKE 216,0:W = 1: GOTO
    1670
2450  GOTO 2360
2460  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0: GET W$: IF  ASC
    (W$) = 67 THEN  CALL 768: GOTO 2460
2470  TEXT : GOTO 1240
2480 F9 = 1: GOTO 2510
2490  PRINT "       Drücken Sie eine Taste!";: GET A$: GOTO 1240
2500  REM ---------- Größen anzeigen ----------
2510  INVERSE : VTAB 1: PRINT "             EINGABEGRöSSEN             ":
    NORMAL
2520  GOSUB 2560
2530  VTAB 22: PRINT "       Drücken Sie eine Taste!";: GET W$
2540  IF F9 = 1 THEN F9 = 0: GOTO 1240
2550  GOTO 230
2560  PRINT : IF F$ = "" THEN  PRINT : GOTO 2590
2570  IF  LEN (F$) > 35 THEN  PRINT "Y = "F$: GOTO 2590
2580  HTAB (37 -  LEN (F$)) / 2: PRINT "Y = "F$
2590  PRINT : PRINT : PRINT "  X min = "XC;: HTAB 23: PRINT "X max = "XD
2600  PRINT : PRINT : PRINT "  Y min = "YC;: HTAB 23: PRINT "Y max = "YD
2610  PRINT : PRINT : PRINT : PRINT "        X-Einheit : "XE
2620  PRINT : PRINT : PRINT "        Y-Einheit : "YE
2630  RETURN
2640  REM ---------- Wertetabelle ----------
2650  VTAB 1: INVERSE : PRINT "             WERTETABELLE              ":
    NORMAL
2660  IF FF = 0 THEN  VTAB 5: PRINT "Als Voraussetzung fehlt:": PRINT : PRINT
    : HTAB 15: PRINT "Die Funktion Y=f(X)": GOSUB 4010: GOSUB 4020: GOTO
    230
2670  PRINT : PRINT : HTAB 5: PRINT "Geben Sie die untere und obere
      Grenze sowie die Schrittweite          für die X-Werte an!"
2680  ONERR  GOTO 3000
2690  VTAB 12: INPUT "      Untergrenze = ";XA
2700  ONERR  GOTO 3010
2710  VTAB 15: INPUT "      Obergrenze = ";XB
2720  POKE 216,0
2730  VTAB 18
2740  IF XA >  = XB THEN  GOSUB 4010: PRINT "    Die Untergrenze muß klei
    ner": PRINT : PRINT "    als die Obergrenze sein !": GOSUB 4020: HOME
    : GOTO 2650
2750  IF  - XA + XB < 1E - 7 THEN  GOSUB 4010: HTAB 5: PRINT BF$: GOSUB 4
    020: HOME : GOTO 2670
2760  INPUT "     Schrittweite : ";S$:S =  VAL (S$): IF S <  = 0 THEN  VTAB
    18: GOTO 2760
2770  IF XB - XA < S THEN  VTAB 22: GOSUB 4010: PRINT "    Die Schrittwei
    te ist zu groß!": GOSUB 4020: VTAB 18: CALL  - 958: GOTO 2760
2780  IF S < 1E - 8 THEN  VTAB 22: GOSUB 4010: PRINT "     Die Schrittwei
    te ist zu klein!": GOSUB 4020: VTAB 18: CALL  - 958: GOTO 2760
2790  HOME
2800  PRINT : HTAB 7: PRINT "X";: HTAB 19: PRINT "I";: HTAB 33: PRINT "Y"

2810  PRINT "======================================="
2820 R = 1 / S
2830  ONERR  GOTO 2930
2840  POKE 34,5
2850  VTAB 24: PRINT "Exit : <";: INVERSE : PRINT "CTRL-C";: NORMAL : PRINT
    ">";: VTAB 7: PRINT : VTAB 6
2860  POKE 35,22
2870 Y =  FN F(XA)
2880  PRINT "   "XA;: HTAB 19: PRINT "I";: HTAB 25: PRINT Y
2890 XA =  INT (R * (XA + S + S / 1.5)) / R: IF XA > XB + S / 1.5 THEN  PRINT
    : PRINT "       Drücken Sie eine Taste!";: GET A$: GOTO 230
2900  IF  PEEK (37) > 19 THEN  GET A$: GOTO 2970
2910  HTAB 19: PRINT "I"
2920  GOTO 2870
2930  IF  PEEK (222) = 255 THEN 230
2940 XA =  INT (R * (XA + S + S / 1.5)) / R
2950  IF XA > XB + S / 1.5 THEN  PRINT : PRINT "     Drücken Sie eine Tas
    te!": GET A$: GOTO 230
2960  RESUME
2970  IF  ASC (A$) = 3 THEN 230
2980  IF A$ = " " THEN 2910
2990  HOME : GOTO 2870
3000  VTAB 12: CALL  - 958: GOTO 2690
3010  VTAB 15: CALL  - 958: GOTO 2710
3020  REM ---------- Diskettenoperationen ----------
3030  HOME : VTAB 1: INVERSE : PRINT "         DISKETTENOPERATIONEN
        ": NORMAL : VTAB 4
3040  VTAB 7
3050  FOR I = 1 TO 8
3060  PRINT "   <";: INVERSE : PRINT I;: NORMAL : PRINT ") "A$(I)
3070  NEXT I
3080  VTAB 23: GOSUB 4030: VTAB 18
3090  PRINT "   "WS$;
3100  GET W$: PRINT W$:W =  VAL (W$)
3110  IF  ASC (W$) = 13 THEN 230
3120  IF W < 1 OR W > 8 THEN  VTAB 18: GOTO 3090
3130  HOME
3140  IF W <  > 8 THEN 3180
3150  PRINT D$"CATALOG"
3160  PRINT : PRINT "Drücken Sie eine Taste";: GET A$
3170  HOME : GOTO 3030
3180  HOME : HTAB (40 -  LEN (A$(W))) / 2: PRINT A$(W)
3190  VTAB 23: GOSUB 4030
3200  IF W = 2 AND FF = 0 THEN  VTAB 10: GOSUB 4010: PRINT "  Es ist Kein
    e Funktion im Speicher!": GOSUB 4020: HOME : GOTO 3030
3210  IF W = 2 THEN  VTAB 18: PRINT "Y="F$
3220  VTAB 6: PRINT "      Geben Sie den Filenamen an!"
3230  PRINT : PRINT : INPUT "Filename : ";N$
3240  IF N$ = "" THEN 3030
3250  IF  LEN (N$) > 30 THEN  GOSUB 4010: PRINT : PRINT : PRINT : HTAB 8:
     PRINT "Der Filename ist zu lang!": GOSUB 4020: GOTO 3180
3260  IF W = 2 OR W = 4 THEN 3300
3270  ONERR  GOTO 4040
3280  PRINT D$"VERIFY"N$
3290  POKE 216,0
3300  ON W GOTO 3310,3390,3510,3580,3720,3780,3950
3310  ONERR  GOTO 3370
3320  PRINT D$"OPEN"N$
3330  PRINT D$"READ"N$
3340  INPUT Y$
3350  PRINT D$"CLOSE"
3360  POKE 216,0: PRINT : PRINT : PRINT "Y="Y$: GOTO 570
3370 F =  PEEK (222): POKE 216,0
3380  IF F = 13 THEN  GOTO 4040
3390  ONERR  GOTO 3420
3400  PRINT D$"VERIFY"N$
3410  GOTO 4140
3420 F =  PEEK (222): POKE 216,0
3430  IF F = 6 THEN 3460
3440  IF F = 9 THEN  VTAB 16: PRINT "         Die Diskette ist voll!": GOSUB
    4020: HOME : PRINT D$"DELETE"N$: GOTO 3030
3450  GOTO 3180
3460  PRINT D$"OPEN"N$
3470  PRINT D$"WRITE"N$
3480  PRINT F$
3490  PRINT D$"CLOSE"
3500  HOME : GOTO 3030
3510  ONERR  GOTO 3550
3520  VTAB 15: HTAB 14: PRINT "Bitte warten!"
3530  PRINT D$"BLOAD"N$",A$4000"
3540  POKE 216,0: GOTO 230
3550 F =  PEEK (222): POKE 216,0
3560  IF F = 13 THEN  PRINT : PRINT "Es existiert Kein Binärfile": PRINT
    : PRINT "mit dem Namen "N$: GOSUB 4020: HOME : GOTO 3180
3570  GOTO 3180
3580  ONERR  GOTO 3610
3590  PRINT D$"VERIFY"N$
3600  POKE 216,0: GOTO 4140
3610 F =  PEEK (222): POKE 216,0
3620  IF F <  > 6 THEN 3180
3630  VTAB 15: HTAB 14: PRINT "Bitte warten!"
3640  ONERR  GOTO 3670
3650  PRINT D$"BSAVE"N$",A$4000,L$1FFF"
3660  POKE 216,0: HOME : GOTO 3030
3670 F =  PEEK (222): POKE 216,0
3680  IF F <  > 9 THEN 3180
3690  PRINT : PRINT : HTAB 10: PRINT "Die Diskette ist voll!"
3700  PRINT D$"DELETE"N$
3710  HOME : GOTO 3030
3720  ONERR  GOTO 3750
3730  PRINT D$"DELETE"N$
3740  POKE 216,0: GOTO 3030
3750 F =  PEEK (222): POKE 216,0
3760  IF F = 10 THEN  PRINT : PRINT : PRINT "Das File ist geschützt!": GOSUB
    4020: HOME : GOTO 3180
```

```
3770  HOME : GOTO 3180
3780  VTAB 12: INPUT "neuer Filename : ";NF$
3790  CALL  - 958
3800  IF  LEN (NF$) > 30 THEN  PRINT : PRINT "Der neue Name ist zu lang!"
    : GOSUB 4020: GOTO 3780
3810  ONERR  GOTO 3880
3820  PRINT D$"VERIFY"NF$
3830  GOSUB 4010
3840  VTAB 14: HTAB 6: PRINT "Es existiert bereits ein File": VTAB 16:A$ =
    "mit dem Namen " + NF$
3850  IF  LEN (A$) > 38 THEN  PRINT A$: GOTO 3870
3860  HTAB (40 -  LEN (A$)) / 2: PRINT A$
3870  GOSUB 4020: VTAB 14: CALL  - 958: GOTO 3780
3880 F =  PEEK (222): POKE 216,0
3890  IF F <  > 6 THEN  HOME : GOTO 3180
3900  ONERR  GOTO 3930
3910  PRINT D$"RENAME"N$","NF$
3920  GOTO 3030
3930 F =  PEEK (222): IF F <  > 10 THEN 3030
3940  GOTO 3760
3950  PRINT D$"LOCK"N$
3960  GOTO 3030
3970  REM  ---------- Data für Funktionsterm ----------
3980  DATA   INT,211,ABS,212,SQR,218,LOG,220,EXP,221,SIN,223,COS,222,TAN,
    224,ATN,225,SGN,210,RND,219
3990  DATA    +,200,-,201,*,202,/,203,^,204,(,40,),41,P,80,E,69,X,88,.,46
    ,0,48,1,49,2,50,3,51,4,52,5,53,6,54,7,55,8,56,9,57
4000  REM ---------- Unterroutinen für Fehlerbehandlung ----------
4010  FOR I = 1 TO 4: CALL  - 198: NEXT I: RETURN
4020  FOR I = 1 TO 2600: NEXT I: RETURN
4030  PRINT "Exit : <";: INVERSE : PRINT "RETURN";: NORMAL : PRINT ">": RETURN

4040  GOSUB 4010: VTAB 14: HTAB 9: PRINT "Es existiert kein File": VTAB 1
    6:A$ = "mit dem Namen " + N$
4050  IF  LEN (A$) > 38 THEN  PRINT A$: GOTO 4070
4060  HTAB (40 -  LEN (A$)) / 2: PRINT A$
4070  GOSUB 4020: GOTO 3180
4080  VTAB 6: PRINT "Es ist nicht möglich,den Funktions-": PRINT : PRINT
    "graphen in das alte Koordinatensystem": PRINT : PRINT "zu zeichnen,d
    a Sie die Achsenabschnitte": PRINT : PRINT "geändert haben!"
4090  PRINT : PRINT "Sollen die alten Achsenabschnitte": PRINT : PRINT "w
    ieder eingesetzt werden? ";
4100  GET W$
4110  IF W$ = "J" OR W$ = "j" THEN XC = X1:XD = X2:YC = Y1:YD = Y2:X5 = X
    1:X6 = X2:Y5 = Y1:Y6 = Y2: GOSUB 4200: GOSUB 4320:FX = 0:FY = 0: GOTO
     1670
4120  IF W$ = "N" OR W$ = "n" THEN 1240
4130  GOTO 4100
4140  GOSUB 4010: VTAB 14: HTAB 6: PRINT "Es existiert bereits ein File"
4150  VTAB 16:A$ = "mit dem Namen " + N$
4160  IF  LEN (A$) > 38 THEN  PRINT A$: GOTO 4180
4170  HTAB (40 -  LEN (A$)) / 2: PRINT A$
4180  GOSUB 4020: GOTO 3180
4190  REM ---------- Einheitenberechnung ----------
4200 Z =  - X5 + X6
4210  IF Z < 1 THEN 4250
4220 U = 0
4230 U = U + 1:Z = Z / 10: IF Z > 1 THEN 4230
4240 XE = 10 ^ (U - 1): GOTO 4280
4250 U = 0
4260 U = U + 1:Z = Z * 10: IF Z < 1 THEN 4260
4270 XE = 10 ^  - U
4280  IF  - X5 = X6 THEN SX = 278 / (2 * X6): GOTO 4300
4290 SX = 278 / ( - X5 + X6)
4300  IF SX * XE > 60 THEN XE = XE / 2: GOTO 4300
4310  RETURN
4320 Z =  - Y5 + Y6
4330 U = 0: IF Z < 1 THEN 4360
4340 U = U + 1:Z = Z / 10: IF Z > 1 THEN 4340
4350 YE = 10 ^ (U - 1): GOTO 4380
4360 U = U + 1:Z = Z * 10: IF Z < 1 THEN 4360
4370 YE = 10 ^  - U
4380  IF  - Y5 = Y6 THEN SY = 190 / (2 * Y6): GOTO 4400
4390 SY = 190 / ( - Y5 + Y6)
4400  IF YE * SY > 50 THEN YE = YE / 2: GOTO 4400
4410  RETURN
4420  REM ---------- Druckroutinen ----------
4430  TEXT : HOME : VTAB 1: INVERSE : PRINT "         DRUCKROUTINEN (FX-8
    0)          ": NORMAL
4440  POKE 34,2
4450  HOME
4460  VTAB 6: HTAB 6: PRINT "<1> "A3$" drucken"
4470  VTAB 8: HTAB 6: PRINT "<2> dto. mit X/Y - Intervall"
4480  VTAB 10: HTAB 6: PRINT "<3> dto. mit "A4$
4490  VTAB 12: HTAB 6: PRINT "<4> NORMAL <--> INVERSE"
4500  VTAB 14: HTAB 6: PRINT "<5> Seitenvorschub"
4510  GOSUB 4910
4520  VTAB 23: GOSUB 4030: VTAB 18: HTAB 6: PRINT WS$;
4530  GET W$:W =  VAL (W$): IF W$ =  CHR$ (13) THEN  HOME : GOTO 230
4540  IF W = 4 THEN IN =  NOT IN: GOSUB 4910: VTAB 18: HTAB 23: GOTO 4530

4550  IF W = 5 THEN  PR# 1: PRINT  CHR$ (12);: PR# 0: GOTO 4530
4560  IF W <  > 1 AND W <  > 2 AND W <  > 3 THEN 4530
4570  HOME : VTAB 6: PRINT "Wie soll der "A3$" gedruckt": PRINT : PRINT "
    werden?"
4580  VTAB 11: HTAB 11: PRINT "<1> linksbündig": VTAB 13: HTAB 11: PRINT
    "<2> rechtsbündig": IF W = 1 THEN  VTAB 15: HTAB 11: PRINT "<3> in de
    r Mitte"
4590  VTAB 23: GOSUB 4030: VTAB 19: HTAB 11: PRINT WS$;
4600  GET A$:A =  VAL (A$): IF A$ =  CHR$ (13) THEN 4450
4610  IF A = 3 AND W = 2 THEN 4600
4620  IF A = 0 OR A > 3 THEN 4600
4630  GOSUB 4970
4640  IF A$ = "J" OR A$ = "j" THEN 4670
4650  IF A$ = "N" OR A$ = "n" THEN 4450
4660  GOTO 6080
4670  POKE  - 16304,0: POKE  - 16299,0: POKE  - 16297,0
4680  PR# 1
4690  PRINT  CHR$ (27)"P";
4700  PRINT  CHR$ (27)"1" CHR$ (0);
4710  IF W <  > 3 THEN 4740
4720  PRINT  CHR$ (27) CHR$ (15)"Funktion: Y = "F$ CHR$ (18) CHR$ (13) CHR$
    (10);
4730  PRINT  CHR$ (18);
4740  IF IN = 0 THEN  POKE 1913,2
4750  IF IN = 1 THEN  POKE 1913,34
4760  IF A = 3 THEN  PRINT  CHR$ (27)"1" CHR$ (16);
4770  IF A = 2 THEN  PRINT  CHR$ (27)"1" CHR$ (33);
4780  PRINT  CHR$ (17);
4790  IF W = 1 THEN 4890
4800  IF A = 1 THEN  PRINT  CHR$ (27)"1" CHR$ (51);
4810  IF A = 2 THEN  PRINT  CHR$ (27)"1" CHR$ (0);
4820  FOR I = 1 TO 4: PRINT  CHR$ (27)"j" CHR$ (255);: NEXT I
4830  PRINT  CHR$ (27) CHR$ (15);
4840  PRINT "X-Interval:" CHR$ (18)" X" CHR$ (27)"S" CHR$ (1)"min" CHR$
    (27)"T"" = "X1; CHR$ (13) CHR$ (10);
4850  PRINT  SPC( 8)"X" CHR$ (27)"S" CHR$ (1)"max" CHR$ (27)"T"" = "X2; CHR$
    (13) CHR$ (10) CHR$ (10);
4860  PRINT  CHR$ (15)"Y-Intervall:" CHR$ (18)" Y" CHR$ (27)"S" CHR$ (1)"
    min" CHR$ (27)"T"" = "Y1 CHR$ (13) CHR$ (10);
4870  PRINT  SPC( 8)"Y" CHR$ (27)"S" CHR$ (1)"max" CHR$ (27)"T"" = "Y2 CHR$
    (13) CHR$ (10) CHR$ (10);
4880  PRINT "X-Einheit: "XE CHR$ (13) CHR$ (10)"Y-Einheit: "YE
4890  FOR I = 1 TO 5: PRINT  CHR$ (10);: NEXT I: PR# 0: GOTO 4430
4900  REM ---------- Unterroutinen ----------
4910  VTAB 23: HTAB 28: INVERSE : IF IN = 1 THEN  PRINT "INVERSE";
4920  IF IN = 0 THEN  PRINT "NORMAL";
4930  NORMAL : PRINT " ": RETURN
4940  VTAB 23: HTAB 30: INVERSE : IF FG = 1 THEN  PRINT "LINIEN";
4950  IF FG = 0 THEN  PRINT "PUNKTE";
4960  NORMAL : VTAB 18: HTAB 28: RETURN
4970  HOME : VTAB 12: HTAB 13: PRINT "Drucker bereit?";
4980  GET A$
4990  RETURN
5000  REM   V 1.1
```

Maschinenprogramm FUNKTIONEN EXC

```
0300-  AD 83 C0   LDA  $C083
0303-  AD 83 C0   LDA  $C083
0306-  A9 00      LDA  #$00
0308-  85 00      STA  $00
030A-  85 02      STA  $02
030C-  A9 D0      LDA  #$D0
030E-  85 03      STA  $03
0310-  A9 40      LDA  #$40
0312-  85 01      STA  $01
0314-  A0 00      LDY  #$00
0316-  B1 00      LDA  ($00),Y
0318-  48         PHA
0319-  B1 02      LDA  ($02),Y
031B-  91 00      STA  ($00),Y
031D-  68         PLA
031E-  91 02      STA  ($02),Y
0320-  C8         INY
0321-  D0 F3      BNE  $0316
0323-  E6 01      INC  $01
0325-  E6 03      INC  $03
0327-  A5 01      LDA  $01
0329-  C9 60      CMP  #$60
032B-  D0 E7      BNE  $0314
032D-  AD 81 C0   LDA  $C081
0330-  AD 81 C0   LDA  $C081
0333-  60         RTS
0334-  AD 83 C0   LDA  $C083
0337-  AD 83 C0   LDA  $C083
033A-  A9 00      LDA  #$00
033C-  85 00      STA  $00
033E-  85 02      STA  $02
0340-  A9 D0      LDA  #$D0
0342-  85 01      STA  $01
0344-  A9 40      LDA  #$40
0346-  85 03      STA  $03
0348-  A0 00      LDY  #$00
034A-  B1 00      LDA  ($00),Y
034C-  EA         NOP
034D-  B1 02      LDA  ($02),Y
034F-  91 00      STA  ($00),Y
0351-  EA         NOP
0352-  EA         NOP
0353-  EA         NOP
0354-  C8         INY
0355-  D0 F3      BNE  $034A
0357-  E6 01      INC  $01
0359-  E6 03      INC  $03
035B-  A5 01      LDA  $01
035D-  C9 F0      CMP  #$F0
035F-  D0 E7      BNE  $0348
0361-  AD 81 C0   LDA  $C081
0364-  AD 81 C0   LDA  $C081
0367-  60         RTS
0368-  AD 83 C0   LDA  $C083
036B-  AD 83 C0   LDA  $C083
036E-  A9 00      LDA  #$00
0370-  85 00      STA  $00
0372-  85 02      STA  $02
0374-  A9 D0      LDA  #$D0
0376-  85 03      STA  $03
0378-  A9 40      LDA  #$40
037A-  85 01      STA  $01
037C-  A0 00      LDY  #$00
037E-  B1 00      LDA  ($00),Y
0380-  EA         NOP
0381-  B1 02      LDA  ($02),Y
0383-  91 00      STA  ($00),Y
0385-  EA         NOP
0386-  EA         NOP
0387-  EA         NOP
0388-  C8         INY
0389-  D0 F3      BNE  $037E
038B-  E6 01      INC  $01
038D-  E6 03      INC  $03
038F-  A5 01      LDA  $01
0391-  C9 60      CMP  #$60
0393-  D0 E7      BNE  $037C
0395-  AD 81 C0   LDA  $C081
0398-  AD 81 C0   LDA  $C081
039B-  60         RTS
```

Erklärung des Autors

Der Autor, Stefan Hubertus Weil aus Wettenberg bei Gießen, erklärt, daß das obige Werk keine Zitate, auch keine Zitate ohne Quellenangabe, aus fremden Werken enthält.

Kyan-Pascal Programming Toolkits

System Utilities, MouseText und TurtleGraphics

getestet von Matthias Meyer

Kyan, Inc. bietet zu der Programmiersprache Kyan-Pascal fünf Software Toolkits mit Assembler-Include-Files an. Mit diesen vorgefertigten Programm-Modulen erhält der Kyan-Pascal-Programmierer sofort einsetzbare Standardprozeduren und -funktionen zur Einbindung in eigene Anwendungen.
Von den Kyan-Toolkits sind bereits erhältlich: System Utilities, MouseText, TurtleGraphics und Advanced Graphics. (Letztere werden in einem der nächsten Hefte besprochen.) MouseGraphics ist zwar bereits angekündigt, jedoch noch nicht erschienen.

1. System Utilities Toolkit

1.1. Einsatzgebiet

Mit Kyan-Pascal bekommt man eine Menge Software fürs Geld. Wenn man jedoch ein komplettes Anwendungsprogramm schreiben möchte, muß man erst einmal zahlreiche Standardprozeduren und -funktionen selbst entwikkeln, die im normalen Sprachumfang von Pascal nicht enthalten sind. Um nun diese Software-Entwicklung abzukürzen und damit die gesamte Software-Erstellung zu vereinfachen, wird dem Pascalprogrammierer mit den System Utilities eine Unterprogrammsammlung mit allgemein einsetzbaren Software-Modulen zur Verfügung gestellt.

1.2. Leistungumfang

Die System Utilities bestehen aus Assemblerprozeduren und -funktionen, die sich durch den Include-Befehl direkt in Pascalprogramme einbinden lassen. Sie umfassen die folgenden vier Teilgebiete:

– *ProDOS-Module:* Append, BLoad, BSave, Copy, Delete, Filesize, Filetype, Find, FindClock, Format, GetClock, GetDate, GetDir, GetPrefix, GetTime, Lock, MakeDir, PrintFile, PrtMLIerror, RemDir, Rename, ScanFile, SetClock, SetDate, SetPrefix, SetTime, Unlock
– *Funktionen zur Auswertung von Eingabegeräten* wie Maus, Joystick, Trackball und Paddle
– *Bildschirm-Management-Routinen:* ClrEOLN, ClrEOP, ClrLine, CLS, Col80, CursorX, CursorY, GetChar, GotoXY, IDMachine, Inverse, Normal, ON40, ON80, ScrnBottom, ScrnFull, ScrnTop, ScrollDown, ScrollUp, TAB, die größtenteils nur für Apple IIc oder Apple IIe mit 80-Zeichen-Karte geeignet sind
– *Standardfunktionen:* Zufallszahlengenerator, externes Sortieren/Mischen, Routine zum Einlesen des Eingabepuffers in einen String, Umwandlungsroutinen REAL nach STRING und umgekehrt sowie INTEGER nach STRING und umgekehrt.

Auf der Rückseite der Toolkit-Diskette befinden sich Demonstrationsprogramme, die einige dieser Utilities verwenden und somit als Anwendungsbeispiele dienen. Die Demo-Software wurde offensichtlich nicht besonders sorgfältig geschrieben, so daß z.B. bei einem CATALOG-Programm vor Anzeige eines Filenamens jeweils das ganze Directory eingelesen wird, was nicht gerade geschwindigkeitssteigernd wirkt. Ein ebenfalls auf der Diskette vorhandenes, in Pascal geschriebenes Programm mit dem Namen ESort demonstriert recht eindrucksvoll, wie lange externes Sortieren auf Diskette dauern kann.

Bei den ProDOS-Utilities funktionieren die Routinen GetClock und SetClock nur mit einer original Thunderclock-Karte. Eine ProDOS-kompatible Timemaster-II-Uhrenkarte z.B. wird zwar von FindClock gefunden, jedoch führt ein Aufruf von GetClock zum Systemabsturz. Außer bei der Bildschirmsteuerung (s.o.) treten sonst aber keine weiteren Kompatibilitätsprobleme mit anderen Apple-II-Konfigurationen auf.

Bewertung: Abgesehen von den nicht ganz fertig entwickelten Demoprogrammen, die im praktischen Gebrauch jedoch keine Bedeutung haben, kann man die System Utilities aufgrund der großen Anzahl nützlicher Erweiterungen sowohl einem reinen Pascal-, als auch einem Assemblerprogrammierer als Softwarebibliothek für Kyan-Pascal weiterempfehlen.

1.3. Dokumentation

Toolkit I von Kyan, Inc. besteht aus einer doppelseitigen Diskette und einer Loseblattsammlung von 31 Blättern zum Einheften in einen Ordner, der das gleiche Format wie der des Kyan Pascal User Manuals hat. Ein von Kyan, Inc. für diesen Zweck vertriebener Ordner kostet US$ 9.95 incl. Porto. Man kann sich aber statt dessen auch einen gewöhnlichen DIN-A5-Ordner kaufen und alle Blätter nochmals lochen.

Bewertung: Die englischsprachige Dokumentation ist wie die des Kyan Pascal User Manuals recht ausführlich und beschreibt im wesentlichen die Übergabe der Parameter an die verschiedenen Utilities.

Ein-Blick

Name:	System Utilities Toolkit
Einsatz:	Assemblerprozeduren und Assemblerfunktionen für Kyan-Pascal
Gesamtwertung:	****
Dokumentation	***** (55 Seiten)
Zweckdienlichkeit:	*****
Demo-Software:	**
Verpackung	****
Preis/Leistung:	****
Preis:	US$ 49,95
Hersteller:	Kyan Software, Inc., San Francisco, USA

(Die beste Bewertung entspricht 5 Sternen)

2. MouseText Toolkit

2.1. Einsatzgebiet

MouseText ist in erster Linie für diejenigen Pascal- oder Assemblerprogrammierer entwickelt worden, die das Maus-Design der Macintosh-Software auch auf dem Apple II in Ihren Programmen verwenden möchten. Es handelt sich hierbei nicht um ein Grafikprogramm, sondern um eine Benutzerschnittstelle mit Fenstertechnik, Menüleisten, Pull-Down-Menüs und mauskontrollierten Programmabläufen. MouseText läuft nur auf dem Apple IIc oder dem neuen bzw. umgerüsteten Apple IIe mit Maus-Zeichensatz.

2.2. Leistungsumfang

MouseText besteht aus einem von Apple, Inc. entwickelten, 12K langen Runtime-Modul, welches im Adreßbereich $6100 bis $8FFF liegt. Zur Anpassung an den Pascal-Compiler hat Kyan, Inc. die folgenden Assemblerprozeduren und -funktionen entwickelt:

– *Cursor-Befehle:* SetCursor, ShowCursor, HideCursor, ObscureCursor
– *Interrupts:* CheckEvents, FlushEvent, GetEvent, PeekEvent, PostEvent, SetKeyEvent
– *Menü-Befehle:* CheckItem, DisableItem, DisableMenu, HighLightMenu, InitMenu, MenuKey, MenuSelect, SetMark, SetMenu
– *Kontroll-Befehle:* ActivateControl, FindControl, SetControlMax, TrackThumb, UpdateThumb.
– *Fenster-Befehle:* CloseAll, CloseWindow, DragWindow, FindWindow, FrontWindow, GrowWindow, InitWindowMargin, OpenWindow, ScreentoWindow, SelectWindow, WindowBlock, WindowChar, WindowOp, WindowString, WindowtoScreen, WindowText.

Für Assemblerprogrammierer wird vom Runtime-Modul ein sog. Machine-Language-Interface zur Verfügung gestellt, das dem ProDOS-MLI nachempfunden wurde.
Die Software macht insgesamt einen sehr ausgereiften und professionellen Eindruck. Mitgeliefert werden zwei Demonstrationsprogramme. Das eine ist in Pascal geschrieben, das andere in Assembler. MouseText kann zwar auch ohne Maus mit der Tastatur bedient werden, doch ist dies recht umständlich. Leider bleiben bei Verwendung von MouseText in Pascal- oder Assemblerprogrammen nur noch 8,25K Programmspeicher übrig, so daß Programmteile öfters von Diskette nachgeladen werden müssen. Um diesem Mißstand abzuhelfen, gibt es zwei Wege:
Entweder eine RAM-Disk verwenden oder bei Kyan, Inc. den Quellcode der Pascal-Library und des MouseText-Moduls bestellen. Mit diesem Quellcode ist laut Handbuch eine bedingte Assemblierung möglich, d.h. es werden jeweils nur diejenigen Teile assembliert, die im Pascalprogramm tatsächlich verwendet werden. Das soll bis zu 50% Speicherplatz einsparen.

Bewertung: Mausorientierte Software ist Geschmackssache. Der eine schwört darauf, der andere hält die Maus für eine Spielerei. MouseText ist ein Hilfsmittel, um die Programmierung von mausorientierter Software zu vereinfachen. Nicht zu übersehende Nachteile aller mausorientierten Programmen sind großer Speicherbedarf und aufwendige Programmierung. Wer jedoch ein Programmierwerkzeug mit „Macintosh-Look und -Feeling" benötigt, ist mit MouseText bestens bedient.

2.3. Dokumentation

Toolkit II von Kyan, Inc. besteht aus einer doppelseitigen Diskette und einer Loseblattsammlung von 94 Blättern zum Einheften. Das MouseText User Manual beschreibt in einer Einführung die Anwendung der Maus- und Fenstertechnik beim Macintosh. Dann wird der Aufruf von MouseText-Befehlen aus Pascal- und Assemblerprogrammen heraus erklärt und die Speicherorganisation behandelt. Es folgen fünf Kapitel mit Beschreibungen einzelner Befehlsgruppen. Auf den nachfolgenden 90 Seiten werden alle MouseText-Befehle, deren Übergabeparameter und Funktionen aufgelistet. Das letzte Kapitel behandelt die in MouseText verwendeten Datentypen und -strukturen. Im Anhang werden das Apple Mouse-Interface und das Mouse-Firmware-Interface, MouseText-Fehlermeldungen, die Diskettenorganisation und die Anpassung anderer Eingabegeräte an MouseText behandelt.

Bewertung: Insgesamt kann man die Dokumentation als sehr ausführlich und daher komplett bezeichnen. Erfreulich ist auch der Anhang über die Hardware und Firmware der Apple-Maus.

Ein-Blick	
Name:	MouseText Toolkit
Einsatz:	Verwendung von Fenstertechnik und Apple-Maus in Kyan-Pascal-Programmen
Gesamtwertung:	*****
Dokumentation:	***** (184 Seiten)
Zweckdienlichkeit:	*****
Demo-Software:	*****
Verpackung:	****
Preis/Leistung:	*****
Preis:	US$ 49,95
Hersteller:	Kyan Software, Inc., San Francisco, USA

(Die beste Bewertung entspricht 5 Sternen)

3. TurtleGraphics Toolkit

3.1. Einsatzgebiet

TurtleGraphics unterstützt eine Auflösung von 280x192 Pixeln und funktioniert deshalb mit jedem Apple-II-Typ. Mit Hilfe dieses Programms kann man unter Kyan-Pascal einfache Hires-Bilder laden und abspeichern, Punkte und Linien zeichnen und Flächen ausfüllen. Die Diskette enthält außerdem Prozeduren zur Erzeugung von Tönen und Geräuscheffekten.

3.2. Leistungsumfang

Die TurtleGraphics-Diskette enthält Include-Files mit folgenden Prozeduren und Funktionen:

– *Grafik:* InitTurtle, PenColor, GrafMode, TextMode, Turn, TurnTo, Move, MoveTo, TurtleX, TurtleY, TurtleAng, ViewPort, FullPort, FillPort, SaveHires, LoadHires
– *Toneffekte:* Beep (wie Control-G), Note (Ton mit bestimmter Höhe und Dauer), Click, Phaser
– *Diagramme:* BarChart (Balkendiagramm), PieChart (Tortengrafik), PlotXY (Kurvendiagramm).

Mitgeliefert werden drei in Pascal geschriebene Demonstrationsprogramme: TURTLEDEMO, SOUND.DEMO und CHART.DEMO. Das erste zeichnet ein paar einfache Grafiken auf den Bildschirm und demonstriert damit die Anwendung fast aller Befehle aus der TurtleGraphics Library. Das zweite Demo-Programm erläutert die Programmierung verschiedener Sound-Effekte mit Hilfe der Befehle aus der Sound Effects Library, so z.B. ein klingelndes Telefon, Phaser-Geräusche und ein Musikstück. CHART.DEMO ist dagegen schon ein kleines Anwendungsprogramm. Es erfordert die Eingabe einer Anzahl von Zahlenwerten und präsentiert dann auf Wunsch ein einfaches Balkendiagramm, eine Tortengrafik oder ein Kurvendiagramm.

Bewertung: Für einfache Grafik-Programmierung in Pascal ist TurtleGraphics ausreichend. Wird jedoch mehr Komfort und doppelte Auflösung gefordert, sollte man sich besser Advanced Graphics zulegen. Die zusätzlich mitgelieferten Sound-Effekte von TurtleGraphics sind wohl mehr als kostenlose Beigabe gedacht.

3.3. Dokumentation

Toolkit IV von Kyan, Inc. besteht aus einer einseitig beschriebenen Diskette und einer Loseblattsammlung von 17 Blättern zum Einheften. Das TurtleGraphics User Manual beschreibt die Philosophie der „Schildkröten"-Grafik und daran anschließend im wesentlichen die Übergabe der Parameter an die verschiedenen Grafik- und Sound-Befehle.

Bewertung: Die Dokumentation ist wie die der anderen Kyan-Toolkits ausführlich genug und somit kein Anlaß zur Kritik.

Ein-Blick	
Name:	TurtleGraphics Toolkit
Einsatz:	Grafik- und Soundbefehle für Kyan-Pascal
Gesamtwertung:	****
Dokumentation:	***** (28 Seiten)
Zweckdienlichkeit:	****
Demo-Software:	***
Verpackung:	****
Preis/Leistung:	****
Preis:	US$ 29,95
Hersteller:	Kyan Software, Inc., San Francisco, USA

(Die beste Bewertung entspricht 5 Sternen)

Apple-II-Produkte

ProDOS-Uhr für Apple IIc

Daten- und Programmfiles können auch auf einem Apple IIc unter ProDOS automatisch mit einem Eintrag von Zeit und Datum in das Diskettendirectory versehen werden. Ermöglicht wird dies durch die batteriegepufferte Kalenderuhr M2000, welche an eine serielle V.24/RS232-Schnittstelle des IIc angeschlossen wird. Für eine möglichst umfassende Kompatibilität mit allen ProDOS-Programmen wurde die offizielle ProDOS-Uhr, wie sie vom Apple IIe bekannt ist, auf dem IIc mit der M2000 emuliert. Erfolgreiche Tests wurden mit Appleworks, BASIC.SYSTEM, EDASM.SYSTEM und MERLIN.SYSTEM durchgeführt. Zeit und Datum können von eigenen Programmen abgefragt und verarbeitet werden. Weitere Eigenschaften der Uhr: paralleler 8-Bit-Port für Einzelbitein- und -ausgabe mit dem IIc, Stand-alone Betrieb als Schaltuhr (Programmierung mit IIc).

Der Preis für die M2000 beträgt ca. DM 387,–.
Bezugsquelle: Pandasoft, Berlin oder IDW, Neufahrn

Apple-Assembler

Dipl.-Math. M. Grözinger befaßt sich in seinem Buch „Der Gläserne Apfel" und dem dazugehörenden Programm „ASTOR" mit der 6502-Assemblerprogrammierung des Apple II und seiner internen Software-Struktur. Die Anleitung besteht aus zwei Komponenten:
– dem Assembler-Tutor (ASTOR), einem leistungsfähigen Tracer und Debugger, mit dem alle internen Abläufe auf Maschinenebene „in Zeitlupe" beobachtet werden können
– dem 150 Seiten starken Handbuch, das problemorientiert aufgebaut ist und mit systemspezifischen Internas vertraut macht.
Dem Assemblerneuling soll ASTOR durch direktes Beobachten des Programmierablaufes und Kontrollieren der Resultate den Einstieg in die Maschinensprache erleichtern. Der Fortgeschrittene findet mit ASTOR schneller Fehler in selbstgeschriebenen Programmen und findet interessante Betriebssystem-Routinen, die er in seine Programme einbauen kann.
Im Buch wird anhand kleiner Assemblerprogramme die Handhabung von ASTOR erläutert. Der zweite Teil befaßt sich mit dem Monitor und dem Basic-Interpreter. Hauptthema ist der interne Aufbau von Basic-Programmen und deren Abarbeitung auf Maschinenebene. Eine Zusammenstellung der wichtigsten Routinen rundet das Kapitel ab. Im dritten Teil werden 5 größere Assemblerprogramme mit ca. 5-8 Seiten Quelltext erläutert.

ASTOR und alle im Handbuch entwickelten Programme werden auf Diskette mitgeliefert und laufen auf allen Apple-Rechnern mit den Prozessoren 6502 und 65C02. Gesamtpreis: ca. DM 79,-.
Bezugsquelle: Stimmler-Elektronik, Tübingen

Fußball-Bundesliga

„Bundesliga" ist ein Programm zur Verwaltung und Verarbeitung der Fußball-Bundesliga-Ergebnisse für Mikro-Computer mit den Betriebssystemen CP/M (z.B. Apple mit Z80-Karte) und PC-DOS (z.B. IBM-PC). Zusammen mit dem Programm wird eine Datendatei geliefert, die alle Spielpaarungen und Ergebnisse seit Bestehen der 1. (1963) und der einteiligen 2. (1981) Bundesliga enthält. Bundesliga wertet Spielergebnisse aus, gibt Listen aller deutschen Meister und alle Plazierungen gewünschter Mannschaften aus etc., kurzum, es erfaßt die gesamte deutsche Fußball-Bundesliga.
Bundesliga kostet für CP/M oder PC-DOS je ca. DM 400,-, eine Demo-Version kann für DM 20,– bezogen werden.
Bezugsquelle: Bekra-Software, Mülheim-Kärlich

Aktienanalyse mit Stockmaster

Computerbesitzern bietet sich jetzt eine gute Möglichkeit, an der Börse zusätzliches Geld zu verdienen: Das Programm Stockmaster II ermittelt durch technische Analyse die aussichtsreichsten Aktien und gibt klare Hinweise auf deren zu erwartende Weiterentwicklung. Dabei können bis zu 150 Aktien gleichzeitig beobachtet werden. Die Computer-Auswertung zeigt auf einen Blick, welche Aktien gekauft, gehalten oder veräußert werden sollten. Eine grafische Darstellung des Wertverlaufes in logarithmischer Einteilung ist möglich. Das Programm ist z.Zt. für Computer von Apple, Commodore und Schneider erhältlich.
Der Preis liegt bei ca. DM 485,–.
Bezugsquelle: Töngi Computer-Praxis, Wiesbaden; in der Schweiz: Denton Consultants AG, Forch/Zürich

Juki-Typenraddrucker

Juki 6500: Der 60 Zeichen/Sekunde schnelle Typenraddrucker Juki 6500 ist eine weiterentwickelte, hochwertige Version der bereits bekannten Drucker Juki 6200 und 6300. Hauptmerkmale sind das duale Interface (per Schalter kann zwischen Centronics-Parallel- und V.24-Schnittstelle gewählt werden), ein Diablo-96-Zeichen-Typenrad, eine Diablo-Hytype-II-Farbbandkassette, Einzelblattzuführung sowie eine internationale Stromversorgungseinheit.
Juki 6300T: Der Juki 6300T ist ein vielseitig verwendbarer Typenraddrucker mit einer Leistung von 40 Zeichen pro Sekunde, der als direkter Ersatz für die IBM-Modelle 5256, 5225, 5219 und 4214 Modell 2 konzipiert wurde. Bei Verwendung des Twin-Ax-Adapters ist der Drucker ebenfalls voll kompatibel mit den IBM-Systemen 34, 36 und 38, IBM 5251 Modell 12 und der IBM 5294 Remote-Control-Unit. Hauptmerkmale sind ein Standard-Diablo-Typenrad und -Farbband und eine 16"-Walze.

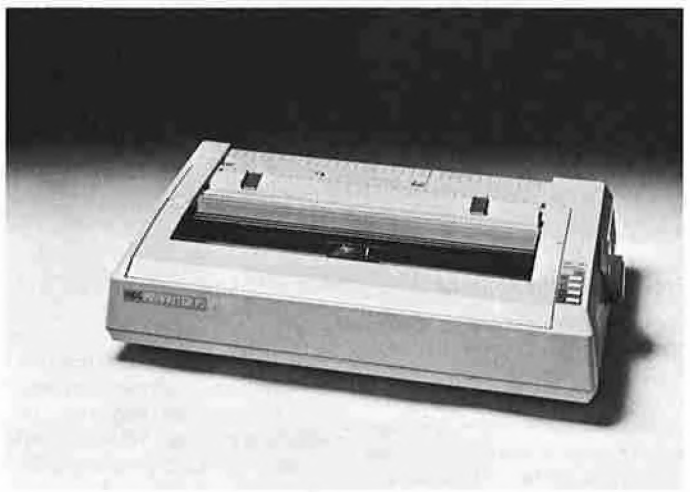
Bezugsquelle: Juki Europe GmbH, Hamburg

Microline-Matrixdrucker

Die Punktmatrixdrucker **Microline 292/293** zeichnen sich durch hohe Druckgeschwindigkeiten und ein gestochen scharfes Schriftbild aus. Der Druckkopf ist mit 18 Nadeln bestückt (versetzte Nadelanordnung) und ist für eine Standfestigkeit von 200 Mio Zeichen ausgelegt. Im Datenverarbeitungsmodus werden bis zu 200 Zeichen/Sekunde gedruckt, im Schönschreibmodus setzen die Microline 292/293 mit 100 Zeichen/Sekunde neue Maßstäbe für die Geschäftskorrespondenz. Kombinationen von verschiedenen Schriftarten (Hoch- und Tiefstellungen, Kursivschrift, horizontaler und vertikaler Fettdruck, Unterstreichen, Proportionalschrift etc.) und Zeichendichten, 14 nationale Zeichensätze und IBM-Satz erlauben die Gestaltung von zahlreichen Schriftvarianten. Durch Farbbandwechsel ist die Umstellung in einem Farbdrucker mit 6 (über Menü) bzw. 13 Farben (über Software) möglich. Die Auflösung bei Bit-Image-Grafiken beträgt 288 x 144 Punkte/Zoll, der Druckpuffer ist 15K groß. Microline 292 und 293 verfügen über eine halbautomatische Einzelblattverarbeitung. Auf Wunsch ist ein vollautomatischer Einzelblatteinzug lieferbar.

Der Microline 292 kostet ca. DM 1900,- zzgl. DM 500,- für das Personality-Modul. Der Microline 293 (Papierbreite bis 16 Zoll) kostet ca. DM 2500,- zzgl. DM 500,- für das Personality-Modul.
Bezugsquelle: Okidata GmbH, Düsseldorf

Der neue **Microline 294** von Okidata ist ein Hochleistungsmatrixdrucker mit einer Druckgeschwindigkeit mit bis zu 400 Zeichen/Sekunde im Datenverarbeitungsmodus. Im Schönschreibmodus druckt er 100 Zeichen/Sekunde. Wie die Microline 292 und 293 ist er mit 18 Nadeln (parallele Nadelansteuerung) und halbautomatischem Einzelblatteinzug ausgestattet und kann bis zu 46 Schriftvarianten ausdrucken. Als Farbdrucker erzeugt er bis zu 13 Farben. Der Microline 294 arbeitet mit Friktions- und Traktoreinzug (maximale Papierbreite 406,4mm). Zur Standardausstattung gehören ein Traktor für Endlosformulare und eine Schallschutzabdeckung. Folgende Personality-Module sind für den ML 294 erhältlich: ML-parallel, ML-seriell (RS232C oder RS422), IBM-seriell und IBM-parallel. Der ML 294 läßt sich leicht an einen IBM-PC oder andere gängige Computersysteme anschließen.
Der Microline 294 kostet ca. DM 3350,- zzgl. DM 500,- für das Personality-Modul.
Bezugsquelle: Okidata GmbH, Düsseldorf



NEC-Matrixdrucker P6/P7

Die neuen Punktmatrixdrucker Pinwriter P6 und P7 von NEC, die sich nur in der Schreibbreite unterscheiden (P6: 80 Zeichen, P7: 136 Zeichen, jeweils bei 10 Zeichen/Zoll), besitzen einen 24-Nadel-Präzisionsdruckkopf. 24 Nadeln (2 x 12) sorgen für ein Schriftbild in echter Korrespondenzqualität. Dank der Auflösung von 360 x 360 Punkten/Zoll werden auch Grafiken detailliert wiedergegeben. Die optionale Colorausstattung ermöglicht brillanten Farbdruck. Die Drucker bieten Centronics-Standardschnittstellen (8 Bit parallel) und RS232C/20-MA-Current-Loop-Schnittstellen, der Druckpufferbereich kann von 8K auf max. 40K erweitert werden. Die Druckgeschwindigkeit wird mit 216 Zeichen/Sekunde bei Schnellschrift und 72 Zeichen/Sekunde bei Korrespondenzschrift angegeben. 12 nationale Zeichensätze sind implementiert. Die Zeichensatzwahl erfolgt manuell oder über Software und wird digital angezeigt.
Bezugsquelle: NEC Business Systems GmbH, Düsseldorf

CP/M Plus für Apple IIe

Das Cirtech-CP/M-Plus-System verändert die gewohnten Apple-IIe-Funktionen nicht, erweitert aber die Einsatzfähigkeit des IIe um ein Vielfaches. Die gesamte Apple-CP/M-Software läuft damit auch auf dem Apple IIe. Zum System gehören der Cirtech-CP/M-Modul mit CPU Z80H (8MHz) und das Original-Betriebssystem CP/M-Plus von Digital Research. Technische Daten: Interne Verarbeitungsgeschwindigkeit 4-6 MHz, Drucker-Spooler für 12000 Zeichen, Ein- und Ausgabespooler für 4000 Zeichen, 128K-RAM-Programmspeicher (Z80-Karte mit 64K RAM ist notwendig).
Massenspeicher: Apple Disk II mit 140K, UniDisk mit 800K, ProFile und Flipper-Karte mit 1M RAM oder andere RAM-Karten (Apple Standard).
Auf dem System läuft die Standard-CP/M-Software, z.B. MBASIC, GBASIC, Pascal, Forth, Turbo Pascal 3.0, Wordstar, Multiplan, dBase, Superquick etc.
Das CP/M-Modul ist voll CP/M-2.23-kompatibel, das CP/M-Plus-Betriebssystem ist kompatibel zu CP/M 2.0.
Preise: IIe CP/M-Plus-Karte mit Betriebssystem CP/M 3.0 ca. DM 550,-
IIe CP/M-Plus-Karte mit Betriebssystem CP/M 3.0 und Wordstar/Mailmerge-Software ca. DM 930.-.
Bezugsquelle: Semjan Computer Systeme, Frankfurt

Sonderangebot für Peeker-Leser

Festplattenkomplettlösung für jedermann

Mit der Firma Frank & Britting GmbH, die auf Festplatten spezialisiert ist, konnten wir ein extrem günstiges Sonderangebot aushandeln, das eine Festplattenkomplettlösung selbst für Apple-Besitzer mit kleinerem Geldbeutel erschwinglich macht. Sie können unter zwei Varianten wählen:

Luxus-Lösung: 20-Megabyte-Festplatte MDB20 (MDB = Mobile Datenbox) + Megaboard-Controller + Handbuch + 3 Konfigurationsdisketten + Anschlußkabel + DB-Meister-Dateiverwaltungsprogramm + Handbuch + 2 Programmdisketten zum Gesamtpreis von nur DM 3199,- inkl. MwSt.

Standard-Lösung: 10-Megabyte-Festplatte MDB10 + Megaboard-Controller + Handbuch + 3 Konfigurationsdisketten + Anschlußkabel + DB-Meister-Dateiverwaltungsprogramm + Handbuch + 2 Programmdisketten zum Gesamtpreis von nur DM 2799,- inkl. MwSt.
(jeweils 6 Monate Garantie.)

Wozu eine Festplatte?

Für eine Festplatte sprechen zwei Argumente, nämlich erstens der bedeutend größere Datenspeicher und zweitens die sehr hohe Zugriffszeit.

● 80- und 160-Spur-Laufwerke haben zwar einen größeren Datenspeicher als die normalen 35-Spur-Laufwerke, doch ist die Zugriffszeit hier wie dort bescheiden.

● RAM-Karten haben zwar eine hohe Zugriffszeit, doch ist der Datenspeicher flüchtig: Strom aus – Daten weg!

Für unsere Festplatten-Standard-Lösung (MDB10) gilt demgegenüber:

– Die Nettospeicherkapazität beträgt hier ca. 9.650K = ca. 9.880.000 Zeichen = ca. 15 Disketten mit 160 Spuren = ca. 69 Disketten mit 35 Spuren.

– Die Datenübertragungsrate beträgt auf unterster Ebene ca. 44K/s = ca. 45.000 Zeichen/s. Zum Vergleich beläuft sich die Übertragungsrate bei normalen Diskettenlaufwerken auf ca. 7,5K/s und bei RAM-Karten auf ca. 44K/s, d.h. RAM-Karten sind also keineswegs schneller, sondern meist sogar langsamer als die MDB-Festplatte.

Sie erhalten deshalb mit unserer Festplattenlösung einen großen externen Massenspeicher bei sehr hoher Datenübertragungsrate zu einem äußerst niedrigen Preis.

Als Betriebssysteme können Sie im einzelnen verwenden:
– DOS 3.3,
– Diversi-DOS 2C und 4C,
– Apple-Pascal 1.1 und 1.2,
– ProDOS 1.0.1, 1.0.2, 1.1.1,
– CP/M 2.2 56K (z.Zt nur diese CP/M-Version).

DB-Meister

Speziell für die zukünftigen MDB-Besitzer wurde das Adreß- und Schemabriefprogramm DB-Meister, das in der normalen Diskettenversion DM 290,- kostet, zum Festplattenbetrieb umgeschrieben. Technische Daten:
– Datensatzlänge bis zu 230 Zeichen, aufteilbar in bis zu 25 Felder; maximal 1000 Datensätze pro Volume
– 2 variable Indexfelder und 1 zusätzliches, konstantes Suchfeld.
– Maskengenerator-Modul für die freie Definition von Bildschirmeingabe- und Ausdruck-Masken.
– Dateipflege-Modul zum Neueingeben, Ändern, Löschen, Ansehen, Bildschirm-Ausdruck usw.
– Sortier- und Filter-Module zum Sortieren und Untersortieren nach Indexfeldern und zum selektiven oder kumulierenden Filtern nach beliebigen Feldern.
– Druck-Modul zum Ausdrucken von Selbstklebe-Etiketten, tabellarischen Listen und Schemabriefen mit automatischem Einschub von Adressen und Anreden sowie weiteren Feldeinschüben im Briefkörper; Einzelblatt und Endlospapier verwendbar; Druckertreiber für Typenraddrucker mit anderer Typenradbelegung können eingebunden werden.
– Brief-Modul zum Schreiben der Schemabriefe; bei Bedarf separat verwendbar.
– Schnelles Backup-Programm (nur 10 Minuten für 2,5 Megabytes bzw. 18 DB-Meister-Dateien).

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 die MDB10 bzw. MDB20, der Megaboard-Controller, das Handbuch und die Konfigurierungsdisketten mit 6 Monaten Garantie geliefert werden. Gleichzeitig erhalten Sie vom Hüthig-Software-Service das DB-Meister-Programm (2 Disketten und 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 normalen 35-Spur-Laufwerken einsetzen.

Zur Bestellung können Sie eine der im Peeker eingehefteten Bestellkarten verwenden. Stichwort:
1 x MDB10-Sonderangebot für DM 2799,-
oder
1 x MDB20-Sonderangebot für DM 3199,-

Warum wollen Sie ein teures Textverarbeitungsprogramm kaufen, wenn es ein billiges und besseres gibt?

Fast-Writer

von Harald Grumser
Programmdiskette und Handbuch
Gerätevoraussetzung: Apple IIe oder IIc (nicht II+)
DOS-3.3-Version. Auslieferung ab 1.6.86
Normalpreis DM 128,– (ISBN 3-7785-1419-9)
Sonderpreis für Peeker-Abonnenten DM 98,–

ProDOS-Version. Auslieferung ab 1.9.86
Normalpreis DM 128,– (ISBN 3-7785-1421-0)
Sonderpreis für Peeker-Abonnenten DM 98,–
Kombinationspreis für Bezieher der DOS-3.3-Version DM 28,–

Der Fast-Writer von Harald Grumser ist in zahlreichen Funktionen wie Scrollen, Suchen und Ersetzen mit Abstand das schnellste und damit angenehmste Textverarbeitungsprogramm für den Apple IIe oder IIc.

Flexibilität

Viele Textverarbeitungsprogramme sind geschützt und laufen deshalb nur in Verbindung mit normalen Disk-II-Laufwerken. Nicht so der Fast-Writer!

– Der Fast-Writer modifiziert weder DOS 3.3 (oder Diversi-DOS) noch ProDOS und kann deshalb mit BRUN FAST.WRITER gestartet werden. Unter Diversi-DOS ist der Fast-Writer dann in 3 Sekunden im Speicher. Vergleichen Sie einmal, wie lange es dauert, bis andere Textprogramme im Speicher sind!

– Da der DOS-3.3-Fast-Writer in den oberen 16K (= Language Card) liegt, kann man ihn vorübergehend verlassen und mit einem einfachen Befehl wieder starten. Mit anderen Worten: Der Fast-Writer ist permanent verfügbar, auch wenn Sie zwischendurch beispielsweise mit FID Dateien kopiert haben.

– Der Fast-Writer läuft mit allen externen Datenspeichern, die für DOS 3.3 oder ProDOS gedacht sind, z.B. mit dem Erphi-160-Spur-Subsystem, mit der Megaboard-MDB-Festplatte, mit RAM-Karten usw. Spezielle Anpassungen sind nicht erforderlich. Suchen Sie einmal ein Textprogramm, das mit diesen Datenspeichern auf Anhieb funktioniert!

– Der Fast-Writer kann mühelos über ein Menü für Ihre speziellen Aufgaben konfiguriert werden. Sie können z.B. per Knopfdruck die Zeilenbreite (normal 80 Zeichen) am Bildschirm einstellen, wobei ab einer Breite von weniger als 41 Zeichen automatisch auf die größere Bildschirmschrift umgestellt wird. Ferner können Sie die Größe des Arbeitsspeichers (insgesamt ca. 35 500 Zeichen) beliebig in Textspeicher und Hilfspuffer (für Löschen und Blockverschieben) aufteilen. Wenn Sie z.B. große Textblöcke im Speicher zu verschieben haben, so können Sie einen entsprechend großen Hilfspuffer von z.B. 10 000 Zeichen einrichten. Damit entfällt das zeitraubende Zwischenspeichern und Einlesen von Diskette.

Befehlsvorrat

Der Fast-Writer verfügt über eine große Zahl von Befehlen, von denen Sie in der Praxis jedoch nur wenige benötigen. Fünf Befehlsübersichten sind durch eingebaute Hilfsübersichten immer abrufbar, so daß Sie schon nach einer mehrstündigen Einarbeitung auf das Handbuch verzichten können. Eine Auswahl der wichtigsten Befehle:

– Freie Cursorbewegung in allen vier Richtungen mit eingebauter Schnell-Scroll-Routine.

– Diverse, per Knopfdruck ein- und ausschaltbare Optionen, z.B. Wortumbruch/kein Wortumbruch, Return sichtbar/unsichtbar, Kopfzeile (Statuszeile) mit Speicherbelegung, Cursorposition usw. eingeblendet/ausgeblendet, Bildschirm geteilt/ungeteilt, Tabulatorleiste sichtbar/unsichtbar, überschreibmodus (statt normalen Einfügmodus) ein/aus usw.

– Eingabe von Kontrollbuchstaben (einschließlich Ctrl-V!) möglich. Automatische Konvertierung in Groß- oder Kleinschreibung (unter Berücksichtigung der Umlaute und ß!)

– Extrem schnelles Suchen und Ersetzen von Zeichenketten (vorwärts und rückwärts).

– Makros frei definierbar und per Knopfdruck abrufbar. Makros können nicht nur stereotype Wortfolgen sein (z.B. „Sehr geehrte Herren"), sondern auch alle Befehlsfolgen, die man beim Fast-Writer sonst über die Tastatur eingeben würde. So läßt sich beispielsweise ein Text automatisch von Laufwerk 1 laden und auf Laufwerk 2 speichern.

– DOS-Kommandos wie Catalog, Delete, Rename usw. immer verfügbar (bei DOS 3.3 zusätzlich Init, bei ProDOS zusätzlich Online und Datum)

– Ausdruck auf Matrixdrucker (normal endlos), Schreibmaschine (normal mit Einzelblatt) und zu Kontrollzwekken auf Bildschirm; links- und rechtsbündig, zentriert und Blocksatz; einstellbarer linker, rechter und oberer Rand (im Text änderbar), bei Bedarf mit Kopfzeile und Paginierung usw. Der Ausdruck kann über eigene Druckertreiber umgelenkt werden, um z.B. Probleme mit Typenrädern, Steuerzeichen usw. zu beheben.

– Makros, Druckparameter, Druckertreiber und Tabulatoren können auf Diskette gespeichert werden.

Hüthig Software Service · Postfach 10 28 69 · 6900 Heidelberg 1