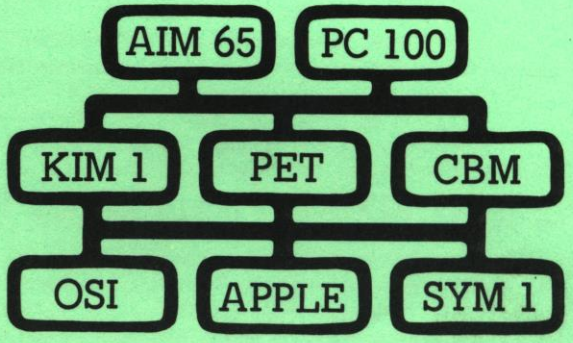
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 9,50 Nr. 29 Januar 1983



Inhaltsverzeichnis

DUO Plott Interface

für MX80 F/T und
ITOH 8510
und COMMODORE-Rechner 30/40/80

Paralleles IEEE 488 - Interface, genormter IEEE-Stecker und Kabel
kompletter Zeichensatz des CBM-Computers
zwei Geräteadressen für Groß-/Grafikmodus und Textmodus
alle Funktionen der Drucker bleiben erhalten, Floppy-Kompatibilität
Deutsche Umlaute, ß und Paragraph
Problemloser Einbau, inklusive deutsches EPSON-Handbuch
Komplettpreis einschl. Kabel, CBM-Grafiksatz und Handbuch incl. MWSt
für EPSON DM 398,--
für ITOH DM 398,--

Umrüstsatz MX80 F/T auf MX80 Typ 3

Aus Ihrem EPSON MX-80 F/T wird der MX-80 Typ 3
Einzelnadelansteuerung, erweiterter Befehlssatz, Elite-Schrift
Preis inkl. MWSt DM 98,--
Komplettpreis Interface, Kabel, Handbuch und Umrüstsatz inkl. MWSt DM 450,--

Für COMMODORE

Deutsche Tastatur mit Original-Tastensätzen (keine Aufkleber)
inkl. deutschem Zeichengenerator
für CBM 8032 DM 98,--
für CBM 8032 und 8096 DM 98,--
nur 8096, ohne Tasten DM 98,--

Speichererweiterung auf 96 K

LOS-kompatibel, inkl. MWSt DM 798,--

ISAM-Routinen
Datenhaltungsprogramm, Indexed Sequential Access Method
siehe Besprechung in Heft 23 des 65xx MICRO MAG, **DM 298,--**

Drucker:

MX-80 Typ 3 DM 998,-- + MWSt

ITOH 8510 DM 1495,-- + MWSt

Stellberg Computer-Systeme

COMMODORE EPSON C. ITOH SOFTWARE INTERFACE

Blindenaaf 36 5063 Overath Tel.: 022 06 - 66 44

Parser und Entscheider

Das ON GOTO-Prinzip

1. Übersicht

Such- und Entscheidungsvorgänge kommen in fast allen Programmen vor, und zwar sowohl in System- wie auch in Anwenderprogrammen. Es ist daher angebracht, sich zu einer zweckmäßigen Codierung Gedanken zu machen, und zwar mit besonderer Hinsicht auf Assemblerprogramme, und dabei das ON GOTO-Prinzip für eine strukturierte Programmierung zu empfehlen.

Entscheidungen: Element nicht gefunden

Suchvorgänge stehen typisch vor Entscheidungen für den besonderen Anwendungsfall. Ein Suchelement wird z.B. in einer zu vergleichenden Menge von Elementen nicht gefunden. Dann mag eine der folgenden Entscheidungen zu treffen sein:

a) Das Suchelement muß der Menge hinzugefügt werden, weil bisher unbekannt: Ein neuer Kunde ist in die Adreßdatei aufzunehmen oder ein neuer Krebs ist in das Verzeichnis der Meeresfauna aufzunehmen.

b) Es liegt ein Fall OTHERWISE vor, das Suchelement mag falsch beschrieben worden sein oder es betrifft die Menge nicht.

Element gefunden

Häufiger ist es jedoch, daß das Element in einer Menge gefunden wird und daß nun eine bestimmte Aktion auszuführen ist. Diese Aktion mag von den verschiedensten Eigenschaften/Attributen des Elementes abhängen. Z.B. gehört es zu den ersten 20 Elementen in einer Liste. Dann hat es ein Attribut der Stellung oder Zone. Oder es hat außer seiner Bezeichnung ein oder mehrere Attribute, die es Gruppen oder Teilmengen zuweisen, z.B. 'war schon Kunde', 'Umsatzklasse 10-50000 DM' usw..

In jedem Fall brauchen wir für das Suchen und Entscheiden eine Routine, die die Menge oder Liste zunächst absucht (Parser) und die die Attribute des gefundenen Elementes aufzuschlagen gestattet. Die Attribute bestimmen dann, an welcher Stelle das Programm fortzusetzen ist. Such- und Entscheidungsroutinen sollten dabei möglichst häufig wiederverwendet werden können, und zwar auch innerhalb eines einzelnen Programmes.

Zwei Fallgruppen

Elemente brauchen einen Bezeichner, einen Namen, nach dem gesucht werden kann. Er wird in eine Tabelle NAMTAB eingeschrieben, wenn das Element der Suchliste bekannt sein soll. Kann er dort nicht gefunden werden, so ist der Fall OTHERWISE auszuführen.

Bezeichner können a) aus einem oder b) aus mehreren Bytes bestehen. Fall a) ist dabei wesentlich einfacher und eigentlich fast immer aus dem Handgelenk zu lösen. Wir stellen ihn zur Vervollständigung dar, obwohl es dazu schon Beispiele gegeben hat.

2. Suchen und Entscheiden bei 1 Byte-Bezeichnern

In der Assemblerprogrammierung ist es üblich, das Suchkriterium im Akkumulator zu übergeben. Dort kann es mit dem vollen Befehlssatz mit wo immer befindlichen Tabellen verglichen werden. Die Qualität dieses Bezeichners im Akku kann nun sehr unterschiedlich sein, u.a.:

a) ASCII- oder Hexwert des Tastendruckes, z.B. bei der Ausführung von Befehlstasten.

b) Vom Interfacebaustein abgenommene Prozeßinformation, die in Aktionen umzusetzen ist.

c) Beginnbuchstabe eines Bezeichners, der auf weitere Tabellen hinweist. Für einen mit 'L' beginnenden Namen braucht man nicht mehr auf 'A-K' abzusuchen.

d) Erwarteter Folgebuchstabe nach einem String, z.B. SPACE oder '=' usw..

Für einen erkannten Bezeichner soll eine gezielte Aktion ausgelöst werden. Das setzt Information voraus, wo das Programm fortzusetzen ist. Es ist dabei üblich, aus der Stellung der Bezeichners in der NAMTAB einen Indexwert im X- oder Y-Register zu gewinnen, der zu einem Sprungziel im Programm weiterverarbeitet werden kann. Wir machen im Sinne einer ON GOTO-Programmierung davon Gebrauch.

Assembler-Programmierer überlegen ihren Code u.a. unter dem Gesichtspunkt der Zeitersparnis. Beim Gebrauch von Entscheidungstabellen ist es daher gut, wenn die am häufigsten vorkommenden Fälle zuallererst abgerastert werden. So ist in Nachnamen der Buchstabe 'S' sicher viel häufiger als 'X' oder 'Y'. Für die Programmlogik ergeben sich aus dem Gesichtspunkt der Ausführungszeit damit lediglich Anordnungsfragen.

2.1. Programmierstil

Wenn nur einige wenige Bezeichner abzuprüfen sind, kann man für die Bezeichnet 04, 06 und 08 z.B. programmieren:

```
0200 C934    0002          CMP #$34        ;EINE ASCII 'VIER' ?
0202 F008    0003          BEQ VIER        ;JA
0204 C936    0004          CMP #$36        ;EINE 'SECHS' ?
0206 F004    0005          BEQ SECHS       ;JA
0208 C938    0006          CMP #$38        ;EINE 'ACHT' ?
020A F000    0007          BEQ ACHT        ;JA
020C         0008 OTHER                     ;OTHERWISE, NICHTS DAVON
020C         0009 VIER                      ;BEFEHLE FUER 4
020C         0010 SECHS                     ;FUER 6
020C         0011 ACHT                      ;FUER 8
```

Dieses ist eine lineare Abfrage, deren Ergebnisse aber schon unübersichtlich werden, wenn gemischt mit 'BEQ' und dann einmal wieder mit 'BNE' programmiert wird. Beim 'BNE' muß man zum Verständnis des Programmes erst einmal in den Befehlen zurückgehen, um zu sehen, welche Fälle vorher schon ausgeschlossen wurden. Man wende also möglichst gleichmäßig 'BEQ' oder 'BNE' an.

In BASIC würde eine lineare Abfragefolge (z.B. nach einem Tastendruck) etwa wie folgt aussehen:

```
 100 GETA$:IF A$=""THEN100
 110 IFA$="4"THEN400
 120 IFA$="6"THEN600
 130 IFA$="8"THEN800
 400 REM TAETIGKEITEN
 600 REM
 800 REM
READY
```

Solche Abfragen können sehr schnell lang und unübersichtlich werden. Sie werden in den Veröffentlichungen zwar noch immer häufig angetroffen, sie sollten aber für BASIC nach den Vorschlägen in Heft 23 (BASIC mit Struktur) in ein ON GOTO aufgelöst werden.

Der empfehlenswerte Programmierstil in Assembler ist , wie auch an der genannten Stelle empfohlen, die Abfrage in einer Schleife, wenn es sich um mehrere Bezeichner handelt, die abgeprüft werden müssen. Ein zyklisches Abfrageprogramm, ein Parser, sieht dann etwa wie folgt aus, wenn das Eingabezeichen im Akku übergeben wurde:

```
0000         0000 JUMP   =$A47D          ;BEIM AIM 65
0000         0001        *=$200
0200         0002 ;VERGLEICHSZEICHEN IM AKKU UEBERGEBEN
0200 A202    0003        LDX #2          ;MENGE DER BEZEICHNER-1
```

```
0202 DD1F02   0004 PARSER CMP NAMTAB,X
0205 F003     0005        BEQ ACTION        ;ZEICHEN GEFUNDEN
0207 CA       0006        DEX
0208 10F8     0007        BPL PARSER
020A          0008 OTHER                    ;OTHERWISE-AKTION
020A          0009
020A 8A       0010 ACTION TXA               ;NUMMER DES ZEICHENS UEBERTR
020B 0A       0011        ASL A             ;MULTIPLIKATION MIT 2
020C AA       0012        TAX               ;ZURUECK Z. ADRESSIERUNG
020D          0013
020D BD2202   0014        LDA ACTVEC,X      ;VEKTOR ZURECHTLEGEN
0210 8D7EA4   0015        STA JUMP+1
0213 BD2302   0016        LDA ACTVEC+1,X    ;HI BYTE
0216 8D7FA4   0017        STA JUMP+2
0219 6C7EA4   0018        JMP (JUMP+1)      ;DIE AUSFUEHRUNG ANSPRINGEN
021C          0019
021C 00       0020 AH     BRK               ;ROUTINE FUER FALL A
021D 00       0021 BEH    BRK               ;FUER B
021E 00       0022 CEH    BRK               ;FUER C
021F          0023
021F 414243   0024 NAMTAB .BYT 'ABC'        ;VERGLEICHSTABELLE MIT A B C
0222          0025
0222 1C02     0026 ACTVEC .WOR AH,BEH,CEH   ;TABELLE DER VEKTOREN
0224 1D02     0026
0226 1E02     0026
```

In der vorstehenden Befehlsfolge wird der momentane Stand des Indexregisters X nach dem Ausprung bei BEQ ACTION benutzt, um nach Multiplikation von X mit 2 (TXA - ASL A - TAX) der Vektortabelle ACTVEC das Sprungziel zu entnehmen, das aus der Tabelle nach JUMP+1 und nach JUMP+2 übertragen und dann indirekt angesprungen wird. Die RAM-Zellen bei JUMP können dabei an beliebiger Stelle im RAM stehen. Das vorstehende Beispiel ist Programmierung ONGOTO.

Für den 6502 ist zu bemerken: Die Wortadresse des Sprungzieles darf nicht im letzten Byte einer Speicherseite beginnen (wrap around bei JMP indirekt). Zweitens: Der CMOS-Prozessor R65C02 hat einen erweiterten Befehlssatz, der mit X indiziert einen indirekten Sprung in eine Tabelle erlaubt. Man wird ihn ggfs. heranziehen.

Die Tabelle ACTVEC kann im WORD-Format bis zu 128 Sprungziele enthalten und erlaubt damit eine weite Auffächerung der Abfragen und Sprungziele. Und man weiß beim Ankommen an jeder symblischen Programmstelle, wie 'AH', 'BEH' usw. was vorher gelaufen ist und was jetzt auszuführen ist..

Es ist dem Programmierer natürlich unbenommen (außer Beachtung der Häufigkeiten), eine Gleichbehandlung für verschiedene Eingabebezeichner vorzusehen. Er wird ihnen dann in ACTVEC gleiche Sprungziele zuordnen und er kann dann noch immer am Stand des Registers X prüfen, ob später evtl. noch auf Unterfälle aufzufächern ist. – Damit sind die Möglichkeiten der Vektorierung (der Programmverzweigung ON GOTO) im wesentlichen erschöpft, es sei denn, man arbeitet mit weiteren Tabellen für Attribute des Bezeichners, die in Abhängigkeit vom Indexregister aufgeschlagen werden.

Dieser Abschnitt 2 für 1-Byte Bezeichner ist zusammenzufassen: Bei mehreren Abfragen ist das Schleifenprinzip zu empfehlen, das über das benutzte Indexregister auf eine Sprungtabelle und weiter auf die gezielte Ausführung führt. An den Ansprungspunkten weiß man, woher man gekommen ist und welche Aktion in der Folge ansteht.

Das ON GOTO-Prinzip kann sowohl in Assembler (wie gezeigt) als auch in höheren Programmiersprachen angewandt werden. In BASIC heißt es auch ON GOTO oder ON GOSUB, in PASCAL gibt es das CASE-Statement, das auch in FORTH nachzuvollziehen ist (in diesen Heft).

3. Suchen und Entscheiden bei Mehrbyte-Bezeichnern

Die nachfolgenden Ausführungen betreffen den Fall, daß die Bezeichner aus mehreren Bytes bestehen. Eine Eingabetextkette ist gegen eine kompliziertere Texttabelle zu vergleichen, wobei sowohl der Eingabestring wie auch die Elemente der Texttabelle jeweils unterschiedliche Stringlängen haben dürfen. Es soll weiterhin auf ON GOTO ausgeführt werden, wobei auch weitere 'properties' oder Eigenschaften der Tabellenelemente aufschlagbar sein sollen.

Geordnete Mengen

Für das Suchen gibt es verschiedenen Lösungswege: Wenn die Menge der Elemente in der Liste z.B. wie in einem Telefonbuch alphabetisch geordnet ist, dann kann man das binäre Suchen anwenden. Man schlägt in der Buchmitte auf und entscheidet, ob man in der vorderen oder hinteren Hälfte weitersuchen muß (Suchargument kleiner oder größer). Danach schlägt man wieder in der Mitte der gewählten Hälfte auf, bis man an das Ziel kommt oder keine Entsprechung findet. Dieser Algorithmus wird hier nicht dargestellt, weil hier speziell ungeordnete Listen behandelt werden sollen, die historisch gewachsen sind oder die unter dem Gesichtspunkt der Häufigkeit angelegt wurden.

Das Längenattribut

Wie schon erwähnt, dürfen die Bezeichner der Elemente eine unterschiedliche Länge haben, z.B. 'Meier' contra 'Koch'. In vielen Dateien und ihren Programmen kann man die Schwierigkeiten, die im Längenattribut liegen mögen, dadurch umgehen, daß die Feldlänge für alle Bezeichner in der Tabelle auf die maximal vorkommende Feldlänge ausrichtet (und ggfs. kürzere Bezeichner nach hinten mit Spaces auffüllt) oder eine willkürliche kleinere Feldlänge als 'signifikant' betrachtet. So finden wir in BASIC für Variablennamen eine Feldlänge von nur 2 Byte oder in vielen Assemblern von 6 Zeichen für die Symboltafel. In solchen standardisierten Fällen weiß der Programmierer stets, mit wievielen Schritten er für Suchzwecke zum Namen des nächsten Elementes in der Liste kommt.

Trennzeichen

Werden für die Bezeichner der Elemente unterschiedliche Längenattribute zugelassen, so sind Trennzeichen zwischen den Bezeichnern notwendig. Üblich sind folgende Trenner:

a) Bit 7 ist im letzten Namensbyte gesetzt.

b) Auf den Namen folgt des Trennbyte hex 00, das leicht zu erkennen ist.

c) Auf den Namen folgt ein anderes Trennbyte, meist hex FF.

Der bei a) genannte Trenner hat den Vorteil, daß er keine Bytes verschwendet und schnell ausführt. Die Bezeichner ADAM und BERTA würden in einer solchen Tabelle wie folgt kompakt codiert sein (Hexbytes):

41 44 41 CD 42 45 52 54 C1

Bei einer solchen kompakten Speicherung ergibt sich das Längenattribut der Bezeichner aus dem Abstand der Trenner (Bit 7). Die Benutzung dieses Bits beschränkt einen Parser auf den Vergleich von ASCII-Strings, denn alle z.B. grafischen Codes würden das Trennzeichen enthalten. Nun sind es ja ASCII-Strings, die am häufigsten abgesucht werden, und wir beschränken uns darauf. Bei anderen Strings müßte man 'reservierte Trenner' z.B. nach b) oder c) benutzen.

Weitere Attribute der Vergleichsliste und des Suchstrings

Zu den Attributen der Vergleichsliste gehört die Mächtigkeit der Menge, die Zahl ihrer Elemente, die abgesucht werden muß. Sie wird im folgenden Parserprogramm in die Variable Menge übertragen und laufend abgefragt.

Auch der von der Eingabe herkommende Suchstring (auf den die Menge abzusuchen ist), hat ein Längenattribut (STRLEN). Sein zweites Attribut ist die Startadresse im Speicher. Dabei sind folgende Fälle zu unterscheiden:

a) Der Suchstring beginnt immer an einer definierten Stelle, an die man ihn transportiert hat.

b) Er beginnt an beliebiger Speicherstelle, z.B. in einem Textspeicher.

c) Er steht in einem Eingabespeicher (BUF), dort aber an variabler Stelle.

Wir behandeln hier speziell den Fall c), weil er Beweglichkeit bietet und nur voraussetzt, daß man die Startadresse des Strings relativ zu BUF als Offset bestimmt und in die Variable BUFZEI gebracht hat. Die Suchmethode ist dann einfach für den Fall a) zu reduzieren. Hinsichtlich des Falles b) ist zu bemerken, daß er keine indizierte Adressierung mit BUFZEI,X, sondern eine Pointeradressierung mit indirekt (BUFZEI),Y nahelegt. Das Indexregister X wird dadurch frei, der Lösungsweg ist aber auch dann analog: Der Befehl CMP BUF,X ist dann zu ändern in den Befehl CMP (STRINGBEGINN),Y.

Die Bestimmung der Anfangsadresse relativ zum Eingabepuffer und die Bestimmung der Länge des Suchstrings (STRLEN) hängt von den Bedürfnissen des Anwenders ab. Bei Texten übergeht man führende Zwischenräume (SPACE) und bemißt die Stringlänge bis hin zum nächsten SPACE oder CR (Carriage Return). In Programm-Quelltexten mag es aber auch andere rechte Begrenzungen eines Schlüsselwortes geben, wie z.B. '=', ')' oder '%' usw.. Für das Suchen nach solchen Begrenzern kommt wiederum die im Abschnitt 2 genannte Methode in Betracht.

Der Parser

Ziele und Voraussetzungen sind genannt: Der Parser soll möglichst vielseitig verwendbar sein und eine ON GOTO-Programmierung ermöglichen. Er sucht in einer oder mehreren NAMTAB(s) nach der Entsprechung zum Eingabestring.

Parser werden in den Programmiersprachen häufig zunächst vom ROM in das RAM übertragen (speziell in die Zero Page) und dort ausgeführt, weil dort eine laufende Modifikation des Parser-Pointers möglich ist und weil Zero Page-Adressierung schneller ausführt. Das ist aber nicht nötig. Im gelisteten Programm wird der Parser dort belassen, wo der Programmier ihn ins RAM oder in ein ROM hingelegt hat. Seine Beweglichkeit erhält er durch Variable, die irgendwo im RAM liegen dürfen und speziell durch den Parser-Pointer PARSPT, der wegen der indirekten Adressierungsart in der Zero Page liegen muß. Auch der Eingabepuffer BUF mag an beliebiger Stelle im RAM stehen, also beim AIM 65 z.B. auch im MONRAM ab DIBUFF (A438).

Der Parser erhält seine Vorgaben durch 'parameter passing', um vielseitig einsetzbar zu sein. Die Anfangsadresse der Vergleichsliste NAMTAB u.ä. wird nach PARSPT gebracht, die Zahl der Elemente in dieser Liste nach MENGE (MENGE könnte mit kleinen Programmänderungen auch in 2 Bytes geführt werden, um mehr als 256 Tabellenelemente abrastern zu können. Um spätere Umrechnungen hinsichtlich der Zahl der überstrichenen Elemente zu ersparen, wird diese in COUNT von 0 an hochgezählt und später zur Indizierung auf den Sprungvektor mit 2 multipliziert.

Während der laufenden Vergleiche im Parser können nun folgende Fälle eintreten:

a) Die momentan verglichenen Zeichen des Eingabestrings und des aktuellen Tabellen-Elementes stimmen nicht überein. Dann wird bei STEP geprüft, ob das Zeichen aus der Tabelle Bit 7 gesetzt hatte (Trenner).

   a1) Wenn nein, dann leeres Weiterparsen. Das Carry-Bit ist der Umschalter.
   a2) Wenn ja, dann Erhöhung des Pointers PARSPT um die Länge des überstrichenen Bezeichers (in Y geführt) und Weiterparsen bei PAR1, es sei den die MENGE sei erschöpft (=0).

b) Die verglichenen Zeichen stimmen überein:

   b1) Der Eingabestring und das Tabellenelement sind nicht erschöpft, dann bei Label PAR5 zum Weiterparsen nach PAR2 verzweigen.
   b2) Das Tabellenelement ist am Label PAR4 erschöpft (negativ=Trenner). Es wird zu PAR7 verweigt. Nun gibt es 2 Möglichkeiten:

b21) Der Eingabestring ist auch erschöpft. Dann liegt volle Übereinstimmung der Strings nach Zeichenfolge und Länge vor und der Parser kann bei ENDPAR verlassen werden.

b22) Der Eingabestring ist noch nicht erschöpft. Er hat zwar bis zum Trenner des Tabellenelementes einen gleichen Anfang, aber er ist länger. Dann ist nach dem Prüfen auf Tabellen-Ende bei PAR6 die Suche ggfs. fortzusetzen.

Die Fallentscheidungen

Weil dieser Parser beide Indexregister benutzt, die damit im Gegensatz zu Abschnitt 2 nicht mehr als Zähler für die überstrichenen Tabellenelemente benutzt werden können, wird die Variable CONT für diesen Zweck benutzt und von 0 an hochgezählt. Das erspart spätere Umrechnungen mit den verbrauchten Elementen aus MENGE.

Eine weitere Variable, die den mehrfachen Einsatz des Parsers in einem Programm ermöglicht, ist FALL. Sie adressiert in eine Tabelle FALLGR mit Sprungvektoren für eine Vorentscheidung (Fallgruppe, die um jeweils +2 zu definieren ist). FALL kann benutzt werden, um am Zielpunkt zu wissen, was geparst wurde. – Bei einfacheren Such- und Entscheidungsaufgaben kann man natürlich auf die über FALL und FALLGR mögliche mehrstufige Auffächerung verzichten und geht nur einmal über ONGOTO.

Der Parser läßt sich natürlich auch als Unterprogramm implementieren. Hier wurde ein gleichmässiges Verbleiben auf der Hauptprogrammebene programmiert.

Aufruf des Parsers

Bei mehrstufigen Entscheidungen (häufigere Benutzung des Parsers in einem Programm) ist die Angabe des FALL notwendig (jeweils +2 je Gruppe). In diesem Fall ist auch jeweils die Basisadresse der abzurasternden Liste (NAMTABx) mitzuteilen. Zur Parameterübergabe gehört auch jeweils die Zahl der in der Vergleichsliste (MENGE) enthaltenen Elemente. – Der Parser kann bei Übergabe dieser Parameter von jeder Programmstelle aus angesprungen werden (JMP), man gelangt immer auf der gleichen Programmebene an vorentschiedene Ansprungspunkte (ON GOTO), an denen man aus der Aufgabenstellung weiß, was nun für den Fall (CASE) nachfolgend zu programmieren ist.

4. Zusammenfassung

Der hier aufgezeichnete Parser wurde entworfen, um eine klar strukturierte Assembler-Programmierung zu ermöglichen. Er kann von überall aus angesprungen werden, wenn man ihm die aktuellen Parameter mitteilt. Und er erlaubt es, Suchtabellen in einer klar gegliederten Form und veränderungsfreundlich anzulegen und zu benutzen. Er erlaubt 128x128=16384 vorentschiedene Aufsetzpunkte zur vorentschiedenen Programmfortsetzung.

Für den Programmierkomfort scheint wesentlich, daß PARSER und ONGOTO zusammen eine klar strukturierte Programmierung ermöglichen. Der Programmierstil des 'Durchhangelns' kann weitgehend verlassen werden.

Definierte Sprungziele haben auch für die Programmerprobung Vorteile. In der Anfangsphase eines Programmes kann man dort 'Programmstummel' anlegen, in Form eines BRK-Befehles, und prüfen, ob man dort wie erwartet ankommt. Im Beispielprogramm ist das an den Stellen ALPHA, BETA, GAMMA und bei NIETE0 so ausgeführt.

Das Prinzip des Parsers und der gezielten Verzweigungen läßt sich ohne große Schwierigkeiten auch auf die Befehlssätze anderer CPU's anpassen. Für den 6502 ist hinzuzufügen, daß der erweiterte Befehlssatz der CMOS-CPU hier und da anders definierte Befehlsformulierungen und Abfragen zuläßt.

```
                        ;-------------------------------------------------------
                        ;PARSER UND ENTSCHEIDER
                        ;BY ROLAND LOEHR
                        ;COPYRIGHT 1983
                        ;REC. PARSER0, 22.1.83
                        ;-------------------------------------------------------
                        ;ARBEITSSPEICHER IN DER ZERO PAGE
                                * =0
0000                    PARSPT  *=*+2           ;PARSER POINTER
0002                    BUFZEI  *=*+1           ;ZEIGER IN DEN EINGABEPUFFER
0003                    STRLEN  *=*+1           ;LAENGE DES EINGABESTRINGS
0004                    MENGE   *=*+1           ;ZAHL DER ELEMENTE IN LISTE
0005                    COUNT   *=*+1           ;ZAHL DER ABRASTERUNGEN
0006                    FALL    *=*+1           ;BEZEICHNER, WAS GEPARST WIRD
0007
0007                            *=*
0007 4C0000             JUMP    JMP 0000        ;SPRUNGVERTEILER IM RAM
000A
                        ;-------------------------------------------------------
                        ;EINGABEPUFFER IRGENDWO IM RAM
000A                            *=$200
0200                    BUF     *=*+80
0250
                        ;-------------------------------------------------------
                        ;HAUPTPROGRAMM
0250                            *=$300
0300                    PARMPS                  ;PARAMETER PASSING TO PARSER
0300 A900               PAR0    LDA #0          ;PARSEN FUER FALL 0
0302 8506                       STA FALL
0304
0304 A96C                       LDA #<NAMTAB    ;BASISADRESSE DER TABELLE
0306 A203                       LDX #>NAMTAB
0308 A003                       LDY #3          ;ZAHL DER ELEMENTE IN TABELLE
030A                    ;JMP PARSER SPRUNG, WENN PARSER ENTFERNT STEHT
                        ;-------------------------------------------------------
                        ;PARSER-PROGRAMM
030A
030A                    PARSER                  ;PARAMETER-UEBERTRAGUNG
030A 8500                       STA PARSPT      ;TABELLENBASIS
030C 8601                       STX PARSPT+1
030E 8404                       STY MENGE       ;ZAHL DER ELEMENTE
0310 A900                       LDA #0          ;COUNT=0
0312 8505                       STA COUNT
                        ;-------------------------------------------------------
                        ;-------------------------------------------------------
0314
0314 A602               PAR1    LDX BUFZEI      ;ZEIGER AUF STRINGBEGINN
0316 CA                         DEX             ;NEUTRALISIERUNG SPAETEREN IN
0317 A000                       LDY #0          ;ZEICHENZAEHLER
0319
0319 38                 PAR2    SEC             ;VERGLEICHSMODUS SETZEN
031A
031A B100               PAR3    LDA (PARSPT),Y  ;ZEICHEN AUS TABELLE
031C 48                         PHA             ;RETTEN WEGEN VORZEICHEN
031D E8                         INX             ;NAECHSTES ZEICHEN ADRESSIEREI
031E C8                         INY             ;DITO
031F 900F               STEP    BCC STEP        ;GGFS. LEERES PARSING
```

```
0321            0054
0321 297F       0055            AND #$7F            ;MACHE IMMER ZU ASCII
0323 DD0002     0056            CMP BUF,X           ;VERGLEICHE M. EINGABESTRING
0326 D008       0057            BNE STEP            ;UNGLEICH, GGFS. LEERES PARSING
0328            0058
0328 68         0059            PLA                 ;ZEICHEN MIT VORZEICHEN ZURUECK
0329 301D       0060 PAR4       BMI PAR7            ;TRENNER GEFUNDEN, ENDE?
032B C403       0061            CPY STRLEN          ;EINGABESTRING ERSCHOEPFT?
032D D0EA       0062 PAR5       BNE PAR2            ;NEIN, WEITER PARSEN
032F 48         0063            PHA                 ;STACKAUSGLEICH
0330            0064
0330 68         0065 STEP       PLA                 ;PRUEFE AUF ENDE TABELLENELEMENT
0331 18         0066            CLC                 ;SETZE MODUS F. LEERES PARSING
0332 10E6       0067            BPL PAR3            ;LEERES PARSING, ZEICHEN POSITIV
0334            0068
0334 E605       0069 STEP1      INC COUNT           ;TRENNER ANGETROFFEN
0336 C604       0070            DEC MENGE           ;VORGABE-1
0338 F012       0071            BEQ ENDPAR          ;TABELLE ERSCHOEPFT, CARRY=0
033A            0072
033A 98         0073 PAR6       TYA                 ;UEBERSTRICHENE ZEICHEN
033B 6500       0074            ADC PARSPT          ;POINTER ERHOEHEN, CARRY=0
033D 8500       0075            STA PARSPT
033F A900       0076            LDA #0
0341 6501       0077            ADC PARSPT+1        ;HI-BYTE
0343 8501       0078            STA PARSPT+1
0345 4C1403     0079            JMP PAR1            ;VERLEICHE NAECHSTES ELEMENT
0348            0080
0348 C403       0081 PAR7       CPY STRLEN          ;EINGABESTRING ERSCHOEPFT?
034A D0E8       0082            BNE STEP1           ;TRENNER WAR SCHON GEFUNDEN
034C            0083
034C            0084 ENDPAR                         ;CARRY=1, WENN GEFUNDEN
034C            0085
-------------------------------------------------------------------------------
SPRUNGVERTEILER ON GOTO
034C            0087
034C A606       0088            LDX FALL            ;ADRESSIERE FALLGRUPPE
034E BD7A03     0089            LDA FALLGR,X        ;LO BYTE
0351 BC7B03     0090            LDY FALLGR+1,X      ;HI BYTE
0354            0091
0354 8508       0092 ONGOTO     STA JUMP+1          ;VEKTOR ABLEGEN
0356 8409       0093            STY JUMP+2
0358 6C0800     0094            JMP (JUMP+1)        ;MOEGLICH AUCH JMP JUMP,
035B            0095 ;WENN DORT OPCODE $4C
035B            0096
-------------------------------------------------------------------------------

-------------------------------------------------------------------------------
PROGRAMMFORTSEZUNG AN DEN SPRUNGZIELEN
035B            0099 FALLO                          ;VORAUFFAECHERUNG
035B 0605       0100            ASL COUNT           ;ELEMENTZEIGER *2
035D A605       0101            LDX COUNT           ;FUER VEKTORIERUNG
035F BD7E03     0102            LDA ACTVEC,X        ;LOW BYTE
0362 BC7F03     0103            LDY ACTVEC+1,X      ;HI BYTE
0365 4C5403     0104            JMP ONGOTO          ;2. AUFFAECHERUNG
-------------------------------------------------------------------------------

0368            0106 FALL2                          ;ENTSPRECHEND AUFFAECHERN
0368            0107
```

```
AUSFUEHRENDE ROUTINEN FALLGRUPPE 0
0368 00       0109 ALPHA  BRK              ;PROGRAMMSTUEMPFE
0369 00       0110 BETA   BRK
036A 00       0111 GAMMA  BRK
036B 00       0112 NIETEO BRK              ;ELEMENT NICHT GEFUNDEN, FALLG

ENSTSPR. AUSFUEHRENDE ROUTINEN GRUPPE 2,4, USW.

LISTE DER VERGLEICHSELEMENTE
036C          0115 NAMTAB
036C 414C     0116         .BYT 'ALPH',$C1 ;ALPHA
0370 C1       0116
0371 424554   0117         .BYT 'BET',$C1  ;BETA
0374 C1       0117
0375 4741     0118         .BYT 'GAMM',$C1 ;GAMMA
0379 C1       0118
037A          0119

VOR-AUFFAECHERUNG FUER DIE FALLGRUPPEN
037A 5B03     0121 FALLGR .WOR FALL0,FALL2
037C 6B03     0121

SPRUNGVERTEILER IN FALLGRUPPE 0
037E          0123 ACTVEC                  ;AUFFAECHERUNG DER AKTIONEN
037E 6B03     0124         .WOR ALPHA,BETA,GAMMA,NIETEO
0380 6903     0124
0382 6A03     0124
0384 6B03     0124

SPRUNGVERTEILER FUER WEITERE FALLGRUPPEN
0386          0126
0386          0127         .END

 ERRORS= 0000                                              R.L. #
```

Dr. A. Schnell, 5100 Aachen

Disk-Interface für FIG-FORTH

und CBM-Computer

Die Programmiersprache FORTH ist in der Grundausstattung im 'Public Domain', d.h. die Sprache kann von jedermann implementiert werden, ohne daß Copyrightbestimmungen berücksichtigt werden müssen. Dies gilt insbesondere für die von der FORTH Interest Group verbreitete FIG-FORTH Version. Source Code und Implementierungshinweise sind gegen Zahlung einer Unkostengebühr von der FORTH Interest Group für alle gängigen Prozessortypen erhältlich, so auch für den in den Commodore-Rechnern verwendeten 6502. Lediglich systemspezifische Teile müssen selbst implementiert werden, dies ist insbesondere das Softwareinterface zu einem Diskettenlaufwerk.

Es gibt zwar einige Anbieter von FORTH-Systemen für CBM-Rechner komplett mit Diskinterface, diese Anbieter lassen sich jedoch die Systemanpassung in der Regel teuer bezahlen. Deswegen ist nachfolgend ein Diskinterface für Commodore-Rechner der BASIC-Versionen 3 und 4 und der FIG-FORTH Grundversion wiedergegeben. Das Interface arbeitet mit den Diskettenlaufwerken 3040, 4040 oder 4031. Hierbei ist das Interface so ausgelegt, daß nur die Standardsprungvektoren am Ende des Commodore-Betriebssystems verwendet werden. Außerdem sind noch eine Reihe von Wortdefinitionen enthalten. Dies sind zu einem fundamentale Stringfunktionen, wie das Definie-

ren von Stringvariablen, Abspeichern von Strings sowie die Stringaddition (diese Definitionen basieren weitgehend auf /1/ und /2/). Dann sind die nötigen Grunddefinitionen eines Editors vorhanden, wobei die wesentliche Editierarbeit dem eingebauten Bildschirmeditor innerhalb des Commodore-Betriebssystems überlassen bleibt. Mit dem FORTH-Wort L wird der zuletzt bearbeitete Screen so gelistet, daß nach jeder Zeilennummer jeweils das Editierwort P erscheint. Damit kann in der vom Commodore-BASIC her gewohnten Weise ein kompletter FORTH-Screen auf dem Bildschirm editiert werden. Neben der Funktion JSR, welche Unterprogrammsprünge in Systemroutinen von der High Level FORTH-Ebene einfach ermöglicht /3/, sind noch alle zum Betrieb des IEEE-Busses der CBM-Rechner nötigen Definitionen vorhanden. Besonders ist zu bemerken, daß das vorliegende Diskinterface in einem ROM oder EPROM abgelegt werden kann, nötige Variablen sind in den 2. Kassettenbuffer von 826 bis 1023 ausgelagert.

Ausgetestet wurde das Interface mit dem AIM 65 FORTH, welches dem FIG-FORTH entspricht, aber den Vorteil hat, in einer ROM-Version für den Prozessor 6502 vorhanden zu sein. Dieses FORTH wurde so modifiziert, daß es auf den CBM-Rechnern der 3000 Serie anstelle der BASIC-ROMs eingesetzt werden kann (nach dem Motto: BASIC raus, FORTH rein -- nach dem Einschalten meldet sich der Rechner mit CBM B3 FORTH V2.0). Damit hat man ein knapp 10K großes FORTH-System. Das Interface sollte aber auch mit jeder anderen Version des FIG-FORTH arbeiten. — Doch jetzt zur Erklärung der einzelnen FORTH-Worte in Form eines 'Glossary', wie es für FORTH üblich ist:

/1/ Rockwell: AIM 65 FORTH User's Manual, 1981, Anhang I
/2/ R. Löhr: Strings für FORTH, 65xx MICRO MAG, Nr. 25, 1982
/3/ A. Schnell: Einfache Anbindung von Systemroutinen in High Level FORTH-Worte, H. 29, 65xx MICRO MAG

FORTH Interest Group, PO Box 1105, San Carlos, CA 94070, USA

Disk Interface Glossary

```
Scr.# 18 und 20: Utilities

ACCU , XREG und YREG  (--- adr.)
geben die Speicherplatzadresse auf den Stack, die zur tempo-
rären Zwischenspeicherung der entsprechenden internen Pro-
zessoregister verwendet werden.

JSR ( adr. ---)
ist eine Hilfsfunktion, die es erlaubt, von der FORTH-Ebene
direkt in Systemroutinen zu springen. Siehe auch /3?.

LIST (Scr.# ---)
übliche Definition zum Listen eines Screens auf dem
Bildschirm.

P ( ---)
Ermöglicht das Editieren einer einzelnen Zeile eines Screens
Die vor P stehende Zeilennummer des current screen wird durc
die hinter P stehende FORTH-Zeile ersetzt.

L ( ---)
Listet den current screen in der zum Editieren mit dem CBM
günstigsten Form. Hinter jeder Zeilennummer wird automa-
tisch P eingefügt. Das Editieren eines Screens wird wie vom
BASIC her gewohnt ('Insert', 'Delete', Cursorsteuerung)
durchgeführt. Jedes 'Return' korrigiert dann die entspr.
Zeile des FORTH-Screens.
```

2DUP (n1,n2 --- n1,n2,n1,n2)
verdoppelt Wert mit doppelter Genauigkeit auf dem Stack; ist in einigen FORTH-Versionen nicht direkt verfügbar.

DSB (--- adr.)
gibt die Startadresse des 18 Zeichen umfassenden Disk-String-Buffers auf den Stack. Die Adresse liegt im 2. Kassettenbuffer.

Scr.# 21 bis Scr.# 23: Elementare Stringoperationen

LEN (adr. --- len)
Ermittelt die gegenwärtige Länge eines Strigs der Startadresse adr..

$VARIABLE (len ---)
Definiert Speicherplatz für einen neuen Datentyp Stringvariable der Länge len. Beispiel: 10 $VARIABLE S1$.

IB (--- adr.)
gibt die Startadresse des Intermediate Buffer auf den Stack. Die maximale Länge dieses für Stringfunktionen benötigten Buffers beträgt 172 Zeichen. Dieser Buffer liegt im 2. Kassettenbuffer.

(")
Hilfsdefinition für Compiletime und " .

" (--- adr. len)
beginnt und beendet einen String (z.B. " STRING"). Während des Compilierens wird der String im FORTH-Speicherbereich ab HERE abgelegt, während im Kommandomodus der String in den IB Pufferbereich abgelegt wird. " ist eine IMMEDIATE-Definition, d.h. sie wird beim Compilieren nicht als Adresse zur Verfügung gestellt, sondern ausgeführt.

STR$ (dlow, dhigh --- adr. len)
wandelt eine Zahl doppelter Genauigkeit (32 Bit) in einen String um, dessen Adresse und Länge auf den Stack gegeben wird.

$CONSTANT (adr. len ---)
definiert einen neuen Datentyp Stringkonstante und kennzeichnet diese Stringkonstante, so daß sie nicht verändert werden kann. Beispiel: " STR.CONSTANTE" $CONSTANT C1$.

2DROP (n1,n2,n3,n4 --- n1,n2)
benötigt 32 Bit Zahl vom Stack. Ist in einigen FORTH-Versionen nicht direkt verfügbar.

$! (adr1. len1 adr2. len2 ---)
speichert Inhalt des String 1 in der Stringvariablen 2 ab. Beispiel: " STRING" S1$ $! oder S1$ S2$ $!.
Prüft, ob Stringvariable 2 eine Stringkonstante ist und gibt gegebenenfalls eine Fehlermeldung aus.

$+ (adr1. len1 adr2. len2 ---)
speichert den Inhalt des String 2 zum Inhalt des des String an die Adresse von String 1. Prüft auf Stringkonstante.
Beispiel: " ABC" S1$ $! " CDE" S2$ S! S1$ S2$ $+
ergibt den String ABCCDE in S1$ (vorausgesetzt, die maximale Länge von S1$ reichte!).

Scr.# 24 bis Scr.# 28: Disk und IEEE-Interface

OPENSFN (log. DEV.# sec.adr. len ---)
öffnet einen File auf einem IEEE-Device; entpricht z.B. dem BASIC-Commando OPEN 1,4,12,"FILE".
Beispiel: 1 4 12 " FILE" OPENSFN .

CLOSE(log. ---)
schließt einen File mit der logischen Nummer log.
Beispiel: 1 CLOSE .

OPEN (log. Dev.# ---)
Einfache Version von OPENSFN ohne Sekundäradresse und Filename. Beispiel: 1 4 OPEN entspricht dem Befehl in BASIC OPEN 1,4.

CHECKOPEN (log. --- Flag)
prüft, ob File mit logischer Nummer log. bereits geöffnet ist und setzt Flag entsprechend.

OPENERR? (log. ---)
gibt Fehlermeldung " FILENOTOPENERROR" aus, wemm File mit logischer Nummer log. nicht geöffnet ist. Ist nötig, da die vom System erzeugten Fehlermeldungen in das BASIC-ROM zurückspringen.

CLRCH (---)
Ausgabe und Eingabe des CBM werden auf Bildschirm und Tastatur wiederhergestellt.

GET# (log. --- ASCII)
liest einzelnes Zeichen vom File vom File mit der logischen Nummer log. vom IEEE-Bus und gibt dieses Zeichen auf den Stack.

BLOCK>TS (Block# --- Sector Track)
rechnet eine Blocknummer zwischen 0 und 664 in eine Track und Sektornummer (für CBM-Laufwerke mit 35 Tracks wie z.B. 4040 oder 4031) um. Directorytrack 18 wird nicht belegt.

TS-APPEND (adr. len. Sector Track --- adr. len.)
u einem vorbereiteten Disk-Kommandostring werden Sector- und Tracknummer hinzugefügt.
Beispiel: " U2:5,0" 12 34 TS-APPEND
ergibt den String " U2:5,0,12.34" .

CMD (log. ---)
entspricht dem BASIC-Kommando CMD4. Der angesprochene File bleibt als Listener am Bus. Weitere Ausgaben werden statt zum Bildschirm zu diesem File umgeleitet.

PRINT# (adr. len. log. ---)
Ausgabe eines Strings zum logischen File log. Entspricht PRINT# log., "STRING" in BASIC.
Beispiel: S1$ 4 PRINT# .

DISCOM (adr. len. ---)
übermittelt einen Kommandostring zum Device# 8, normalerweise dem Diskettenlaufwerk, mit secondary address 15.
Beispiel: " I0" DISCOM initialisiert Laufwerk null.

RB (adr. ---)
Codedefinition zum Lesen eines Blocks (254 Byte) aus einem Puffer des Diskettenlaufwerkes und Abspeicherung ab Adresse adr. Das 255. Byte wird mit dem Hexcode für Space aufgefüllt.

READBLK (adr. Block# ---)
Der Block mit Nummer wird von der Diskette in den Rechner übertragen und ab adr. abgelegt.

WB (adr. ---)
Codedefinition zum Schreiben eines Blocks in einen Diskettenlaufwerks-Puffer ab der Adresse adr. des Rechners.

WRITEBLK (adr. Block# ---)
Ab Adresse adr. werden 254 Zeichen als Block im Direktzugriff auf die Diskette geschrieben.

DR/W (adr. Block# Flag ---)
Kompletter Block wird abhängig vom Flag von Diskette gelesen oder auf Diskette geschrieben.

INITD (---)
initialisiert Diskinterface und Screenbuffer mit 4 Screens. Einbindung der Codefiledaddress von DR/W in das jeweilige FORTH-system muß ggfs. angepaßt werden. In vorliegender Form gilt sie für das von mir benutzte spezielle AIM 65 FORTH.

DISKERR (---)
Fehlermeldung wird vom Diskettenlaufwerk abgefragt und auf den Bildschirm ausgegeben.

```
scr # 18
  0 hex
  1 : accu 203 ;
  2 : xreg 204 ;
  3 : yreg 205 ;
  4 code jsr ( adr. --- )
  5       xsave stx,
  6       4c # lda, n sta,
  7       bot lda, n 1+ sta,
  8       bot 1+ lda, n 2 + sta,
  9       accu lda, xreg ldx, yreg ldy,
 10       n jsr,
 11       accu sta, xreg stx, yreg sty,
 12       xsave ldx, pop jmp,
 13 ( end-code)
 14 14 load
 15
```

```
scr # 20
  0 decimal forth
  1 : list ( scr# --- )
  2      dup ." scr# " . scr ! 16 0 do cr
  3      i 2 .r space i scr @ .line
  4      loop ;
  5 : p scr @ (line) over swap blanks
  6      0 word here count 64 min rot
  7      swap cmove update ;
  8 : l ( --- )
  9      scr @ dup ." scr# " . scr ! 16 0 do cr
 10      i 2 .r ."  p " i scr @ .line
 11      loop ;
 12 : 2dup ( n1,n2 --- n1,n2,n1,n2 )
 13      over over ;
 14 : dsb 1001 ( --- adr. ) 18 1000 c! ;
 15 -->

scr # 21
  0 : len ( adr. --- len )
  1      dup begin dup c@ swap 1+ swap
  2      0= end swap - 1 - ;
  3 : $variable ( len --- )
  4      <builds abs 172 min 1 max dup
  5      c, 0 do 32 c, loop 0 c,
  6      does>  1+ dup len ;
  7 : ib 827 ( --- adr. ) 172 826 c! ;
  8 : (") ( fuer " --- adr.,len )
  9      r count dup 1+ r> + >r ;
 10 : " 34 state @ if compile (")
 11      word here c@ 1+ allot
 12      else word here count ib swap
 13      rot over ib swap 1+ cmove
 14      2dup + 0 swap c! then ;
 15 immediate -->


scr # 22
  0 : str$ ( dlow,dhigh --- adr.,len )
  1      swap over dabs <# dpl @ 0=
  2      0= if begin dol @ 0= 0=
  3      while # dpl @ 1 - dpl !
  4      repeat 46 hold then #s
  5      sign #> ;
  6 : $constant ( adr.,len --- )
  7      <builds 0 c, swap drop 0 do
  8      ib i + c@ c, loop 0 c,
  9      does> 1+ dup len ;
 10
 11
 12
 13
 14
 15 -->
```

```
scr # 23
  0 : 2drop drop drop ; hex
  1 : $! ( adr.,len,adr.,len --- )
  2     drop dup cfa 1 - @ 22 -
  3     ' $constant 0= if 2drop
  4     drop ." stringconst.!" else
  5     dup 1 - c@ rot min 0 max
  6     2dup + 0 swap c! cmove then ;
  7
  8 : $+ ( adr.,len,adr.,len --- )
  9     4 pick 1 - c@ dup 0= if 2drop
 10     2drop drop ." stringconst.!" else over - 4 pick
 11     - dup 0< if + else min then
 12     >r rot rot + r> 2dup + >r
 13     cmove 0 r> c! then ;
 14 hex
 15 -->


scr # 24
  0 : opensfn ( log.,dev.#,sec.,adr.,len ---)
  1     d1 c! da ! d3 c! d4 c! d2 c!
  2     ffc1 @ 3 + jsr ;
  3 : close   ( log. --- )
  4     d2 c! ffc4 @ 3 + jsr ;
  5 : open ( log.,dev.# --- )
  6     over close 0 ib 0 opensfn ;
  7 : checkopen ( log. --- flag )
  8     dup ae c@ -dup if 251 + 251 do dup i
  9     c@ = if 1+ then loop then swap - ;
 10 : openerr? ( log. ---)
 11     checkopen 0= if ." file not open error"
 12     cr abort then ;
 13 : clrch ( --- )
 14     ffcc jsr ;
 15 -->


scr # 25
  0 : get# ( log. --- ascii )
  1     dup openerr? xreg c! ffc6 jsr ffe4 jsr
  2     accu c@ clrch ;
  3 decimal
  4 : block>ts  ( block# --- sector,track )
  5     dup 357 < if 21 /mod 1+ else
  6     dup 471 < if 357 - 19 /mod 19 + else
  7     dup 579 < if 471 - 18 /mod 25 + else
  8     dup 664 < if 579 - 17 /mod 31 + else
  9     then then then then ;
 10 : ts-append ( adr.,len,sector,track --- adr.,len )
 11     4 pick >r 100 * + 0 2 dpl !
 12     str$ $+ r> dup len ; hex
 13 : cmd ( log. --- )
 14     dup openerr? xreg c! ffc9 jsr ;
 15 -->
```

```
scr # 26
  0 : print# ( adr.,len,log. --- )
  1     cmd type cr clrch ;
  2 : discom ( adr.,len --- )
  3     f print# ;
  4 code wb xsave stx, bot lda, n sta,
  5     bot 1+ lda, n 1+ sta,
  6     5 # ldx, ffc9 jsr, ff #  ldy,
  7     begin, dey, n )y lda,
  8     ffd2 jsr, 00 # cpy, 0= until,
  9     ffcc jsr, xsave ldx, pop jmp,
 10     ( end-code)
 11 : writeblk ( adr.,block --- )
 12     swap " b-p:5,1" f print# wb
 13     " u2:5,0," dsb f $! dsb dup len
 14     block>ts ts-append f print# ;
 15 -->

scr # 27
  0 code rb xsave stx, bot lda, n sta,
  1     bot 1+ lda, n 1+ sta, 5 # ldx,
  2     ffc6 jsr, ff # ldy,
  3     begin, dey, tya, pha,
  4     ffe4 jsr, tax, pla, tay, txa,
  5     n )y sta, 00 # cpy, 0= until,
  6     dey, 20 # lda, n )y sta,
  7     ffcc jsr, xsave ldx, pop jmp,
  8     ( end-code)
  9 : readblk ( adr.,block --- )
 10     " u1:5,0," dsb f $! dsb dup len rot block>ts
 11     ts-append f print# " b-p:5,1" f print# rb ;
 12 : dr/w ( adr.,block,flag --- )
 13     5 openerr? base @ 3ff c! a base !
 14     if readblk else writeblk then 3ff c@ base ! ;
 15 -->

scr # 28
  0 : initd ( --- )
  1     100 ub/buf ! 4 ub/scr !
  2     b003 94 ! ' dr/w cfa ur/w !
  3     f close f 8 f " i" opensfn
  4     5 close 5 8 5 " #" opensfn
  5     limit b/buf 4 + b/scr *
  6     4 * - ufirst !
  7     0 offset ! first use !
  8     first prev ! empty-buffers ;
  9 : diskerr ( --- )
 10     20 0 do f get# dup 0d = if
 11     leave then emit loop cr ;
 12 ;s
 13
 14
 15
```

#

Klaus Flesch, 7820 Titisee-Neustadt

FORTH-Editor

für bildschirmorientiertes Editieren

Es war das Ziel, einen bildschirmorientierten FORTH-Editor zum Bearbeiten der sog. Screens zu schreiben. Ein Screen ist in der FORTH-Terminologie ein Block von 16 Zeilen mit einer Länge von je 64 Zeichen. Dazu hat das FIG-FORTH schon einige Standardbefehle im Grundsystem. Sie führen praktisch eine virtuelle Speicherverwaltung eines zur Verfügung gestellten Bereiches durch. Maßgebend für den Einssatz dieses Editors ist also der Anschluß eines DOS an das FORTH. Dies kann z.B. auch durch die Speicherverwaltung aus MICRO MAG Nr. 26, Seite 21 von Herrn Asche realisiert werden. Ebenso kann man auch die Routine im AIM 65 Handbuch für den RM 65-Controller verwenden.

Der Screen-Editor entstand in mehreren Anläufen. Zuerst wurde ein Primitiv-Editor geschrieben, der mit einem über das Memory gekoppelten Interface das Editieren übernahm. Nachdem wir jedoch die Videoplatine wechseln mußten und dann auch noch optional der AIM 65 mit einem Terminal gesteuert werden mußte, wurde nachfolgender Editor erstellt. Er macht keinerlei Gebrauch davon, daß er die Zeichen aus dem Bildwiederholspeicher zurücklesen müßte. Um den Editor hierfür anzupassen, wurde ein Terminal-Vokabular geschaffen. Dort werden bildschirmabhjängige Befehlsworte abgespeichert.. Dieses Vokabular besteht zur Zeit aus drei Worten:

CLS	löscht den Bildschirm
ESC	gibt den ESCAPE-Code aus (dezimal 27)
GOTOXY	positioniert den Cursor auf die Position X,Y mit X=0...23 und Y=0...79. Bei größeren Werten werden die Grenzen angenommen.

Ebenfalls sollten die CASE-Compilerbefehle der FORTH Interest Group definiert sein. Diese Worte sind alle IMMEDIATE definiert, d.h. sie werden auch innerhalb einer ':'-Definition ausgeführt.

Der Editor beginnt auf Screen 116. Zu Listing ist zu sagen, daß der Klammeraffe als Paragraphenzeichen dargestellt wird. Aus Zeitgründen mußte die Kommentierung auf das nachfolgende Glossar beschränkt werden, das die Funktion der Befehlsworte erklärt und die Bildschirmbefehle listet.

Falls das Listing in den Editor des AIM 65 eingegeben wird, braucht man nur den Text hinter den Zeilennummern einzutippen. Die jeweiligen Worte 'next screen' (-->) sind dann auszulassen.

Kurzbeschreibung der Worte

C/LINE	die Konstante 64. Bei X-Forth für 6809 muß diese 63 sein
SCREEN	enthält das Abbild des gerade bearbeiteten Screens
CURSOR	enthält die Position des Cursors
SCREEN #	enthält die Nummer des zu editierenden Screens
UPDATED	Flag ob sich der Inhalt geändert hat
MODE	Flag ob eingefügt werden muß
S=COUNT	Hilfsvariable für S=
S=	(addr1 addr2 --- Flag) überprüft zwei Strings auf Gleichheit
MODIFY	Inhalt hat sich geändert!
CURSOR-CHK	überprüft CURSOR und setzt sie in den Bereich 0...1023

§(CURSOR)	(--- Char)
	holt das Zeichen auf dem Cursor
!(CURSOR)	speichert einen Wert ab
.CURSOR	positioniert den Bildschirmcursor auf den Wert
	von Cursor
+.CURSOR	addiert einen Wert zu CURSOR und positioniert ihn
.LINE	(n ---)
	zeigt die Zeile auf dem Bildschirm an
?LINE	(--- n)
	holt die aktuelle Bildschirmzeile
.LINE-TO-END	zeigt den Rest der aktuellen Zeile an
MOVE-LINE	(FROM TO ---)
	verschiebt eine Zeile
.SCREEN#	zeigt den aktuellen Screen an
.GOTO	positioniert den Cursor auf die Zeile 2 Spalte 55
.MODE	gibt den aktuellen Modus aus
.WRITING	gibt an, daß auf die Diskette geschrieben wird
.READING	gibt an, daß von der Diskette gelesen wird
.INSERTING	gibt an, daß der Text eingefügt wird
§(SCREEN#)	holt einen Block nach SCREEN
!(SCREEN#)	schreibt einen Block von SCREEN
BORDER	zeichnet eine Umrahmung
.SCREEN	gibt den gesamten Screen aus
NEW-SCREEN	(---) holt einen neuen Screen
.SCREEN-TO-END	zeigt den Rest des Screens ab der aktuellen Zeile
NEXT-SCREEN	holt den nächsten Screen
PREV-SCREEN	holt den vorhergehenden Screen
SPREAD-LINE	fügt eine Leerzeile an der aktuellen Position ein
	die Zeile 15 geht verloren
DELETE-LINE	löscht die aktuelle Zeile
	die Zeile 15 ist dann leer
BACKUP	Cursor links
TAB-RIGHT	Tabulator vorwärts
TAB-LEFT	Tabulator rückwärts
TAB-UP	Zeile hoch
TAB-DOWN	Zeile runter
ERASE-TO-END	löscht den Rest des Screens
ERASE-LINE	löscht die Zeile Rest wird jedoch nicht nachgeschoben

DELETE-CHAR	löscht aktuelles Cursorzeichen Rest der Zeile wird nach links geschoben
SPREAD-CHAR	schafft Platz für ein Zeichen das letzte Zeichen der Zeile geht verloren
INSERT-CHAR	fügt so lange Zeichen ein bis ein Controllzeichen eingegeben wird
(EDIT)-FLAG	Flag ob Editmodus verlassen werden soll
(EDIT)	Hauptcontrollschleife für die Eingabe
"IN	holt eine Zahl von der Tastatur keine Korrekturmöglichkeit
EDIT	damit wird der Editor aufgerufen es gibt nun zwei Möglichkeiten: E editieren eines Screens C kopieren von Screens

Kontrollbefehle (Tastendrücke) für die Steuerung des Editors:

CTRL. +	Bedeutung
A	Tabulator links
C	nächster Screen
D	Cursor rechts
E	Zeile hoch
F	Tabulator rechts
H	Cursor home
I	Tabulator rechts
K	speichert den Screen zurück und schreibt alle geänderten Blöcke auf die Diskette zurück
M	CR geht an den Anfang der nächsten Zeile (return)
N	fügt eine Leerzeile ein
O	beendet das Editieren
P	löscht die Zeile (d. h. Zeile wird mit Blanks überschrieben)
Q	holt den gerade aktuellen Screen zurück (d.h. alle Editfunktionen werden rückgängig gemacht)
R	holt den vorhergegangenen Screen
S	Cursor links
V	einfügen von Zeichen
Y	löscht die Zeile Rest wird nachgeschoben
Z	füge ein Zeichen ein

F2	löscht den Screen
F3	neuschreiben des Bildschirms
DEL	löschen eines Zeichens

Außer den vorgenannten werden alle anderen Zeichen in den Screen eingefügt.

In den nachfolgenden Screens sind die unbenutzten unteren Zeilen aus Platzgründen für diesen Abdruck fortgelassen worden.

```
SCR # 116
  0 ( EDITOR VON LAB     MODIFIED K.F.    1.12.82)
  1 VOCABULARY EDITOR IMMEDIATE EDITOR DEFINITIONS
  2
  3 64 CONSTANT C/LINE
  4 1024 CARRAY SCREEN
  5 0 VARIABLE CURSOR
  6 0 VARIABLE SCREEN#
  7 0 VARIABLE UPDATED
  8 0 VARIABLE MODE
  9 0 VARIABLE S=COUNT
 10
 11 : S= 0 S=COUNT ! 0 DO
 12                  2DUP C$ SWAP C$ - S=COUNT +! 1+ SWAP 1+ LOOP
 13          DROP DROP S=COUNT $ 0= ;
 14 -->
 15


SCR # 113
  0  ( TERMINAL VOCAB                )
  1 :   CARRAY <BUILDS 1+          0 DO 0 C, LOOP
  2             DOES>             + ;
  3 VOCABULARY TERMINAL IMMEDIATE TERMINAL DEFINITIONS
  4 : CLS  26 EMIT ;
  5 : ESC  27 EMIT ;
  6
  7
  8
  9
 10
 11 : GOTOXY (  Y X ---  ) SWAP
 12          ESC 61 EMIT 0 MAX 79 MIN 32 + EMIT 0 MAX 23 MIN
 13          32 + EMIT ;
 14 FORTH DEFINITIONS
 15

SCR # 114
  0 : CASE   ?COMP CSP $ !CSP 4 ; IMMEDIATE
  1 : OF   4 ?PAIRS COMPILE OVER COMPILE =
  2        COMPILE OBRANCH HERE 0 , COMPILE DROP 5 ; IMMEDIATE
  3 : ENDOF  5 ?PAIRS COMPILE BRANCH HERE 0 , SWAP 2
  4          ÄCOMPILEÜ ENDIF 4 ; IMMEDIATE
  5 : ENDCASE 4 ?PAIRS COMPILE DROP BEGIN SP$ CSP $ = 0=
  6           WHILE 2 ÄCOMPILEÜ ENDIF REPEAT
  7           CSP ! ; IMMEDIATE ;S
  8
```

```
SCR # 117
  0 ( FORTH SCREEN EDITOR )
  1 : MODIFY 1 UPDATED ! ;
  2 : CURSOR-CHK     CURSOR $ 0 MAX 1023 MIN CURSOR ! ;
  3 : $(CURSOR)    CURSOR $ SCREEN C$ ;
  4 : !(CURSOR) CURSOR $ SCREEN C! MODIFY ;
  5 : .CURSOR     CURSOR $ C/LINE /MOD SWAP
  6                6 + SWAP 3 + TERMINAL GOTOXY ;
  7 : +.CURSOR CURSOR +! CURSOR-CHK .CURSOR ;
  8 -->
  9

SCR # 118
  0 ( FORTH EDITOR     )
  1 : .LINE DUP 3 + 6 SWAP TERMINAL GOTOXY EDITOR
  2         C/LINE * SCREEN DUP C/LINE + SWAP
  3         DO I C$ EMIT LOOP ;
  4 : ?LINE    CURSOR $ C/LINE /MOD SWAP DROP ;
  5 : .LINE-TO-END  ?LINE 1+ C/LINE * SCREEN CURSOR $
  6                 SCREEN DO I C$ EMIT LOOP ;
  7 : MOVE-LINE     SWAP C/LINE * SCREEN SWAP C/LINE * SCREEN
  8                 C/LINE CMOVE MODIFY ;
  9 -->
 10

SCR # 119
  0 ( SCREEN EDITOR )
  1 : .SCREEN#    5 1 TERMINAL GOTOXY EDITOR
  2               ." SCREEN # " SCREEN# ? ;
  3 : .GOTO TERMINAL 55 1 GOTOXY ;
  4 : .MODE    .GOTO ." MODE = "
  5      MODE $ IF ." COMMAND" ELSE ." EDIT    " THEN ;
  6 : .WRITING .GOTO ." WRITING DISK     " ;
  7 : .READING .GOTO ." READING DISK     " ;
  8 : .INSERTING .GOTO ." INSERTING TEXT   " ;
  9 -->
 10


SCR # 120
  0 ( SCREEN EDITOR )
  1 : $(SCREEN#) 0 UPDATED ! .READING SCREEN# $ 8 * DUP 8 + SWAP DO
  2              I BLOCK I 7 AND 128 * SCREEN 128 CMOVE LOOP OFF ;
  3 : !(SCREEN#) UPDATED $ IF
  4              .WRITING SCREEN# $ 8 * DUP 8 + SWAP DO
  5      I BLOCK I 7 AND 128 * SCREEN SWAP 128 CMOVE UPDATE
  6      LOOP OFF  THEN ;
  7  -->
  8

SCR # 121
  0 ( SCREEN EDITOR    )
  1 : BORDER TERMINAL CLS 5 2 GOTOXY 66 0 DO 45 EMIT LOOP
  2   EDITOR 16 0 DO MODE $ 0= IF TERMINAL
  3    0 I 3 + GOTOXY I 3 .R THEN
  4    5 I 3 + GOTOXY 124 EMIT
  5    70 I 3 + GOTOXY 124 EMIT LOOP
  6    5 19 GOTOXY 66 0 DO 45 EMIT LOOP ;
  7  -->
  8
```

```
SCR # 122
  0  ( SCREEN EDITOR     )
  1 : .SCREEN     .SCREEN# .MODE 16 0 DO I .LINE LOOP
  2               .CURSOR ;
  3 : NEW-SCREEN $(SCREEN#) .SCREEN ;
  4 : .SCREEN-TO-END 16 ?LINE DO I .LINE LOOP .CURSOR ;
  5 -->
  6

SCR # 123
  0 (  SCREEN EDITOR    )
  1 : NEXT-SCREEN  !(SCREEN#) 1 SCREEN# +! NEW-SCREEN ;
  2 : PREV-SCREEN !(SCREEN#) SCREEN# $ 1 - 0 MAX
  3               SCREEN# ! NEW-SCREEN ;
  4 -->
  5

SCR # 124
  0  ( SCREEN EDITOR     )
  1 : SPREAD-LINE ?LINE  15 - IF ?LINE 1 - 14 DO
  2         I DUP 1+ MOVE-LINE -1 +LOOP THEN
  3         ?LINE C/LINE * SCREEN C/LINE BLANKS
  4         .SCREEN-TO-END ;
  5  -->
  6

SCR # 125
  0  ( SCREEN EDITOR     )
  1 : DELETE-LINE ?LINE 15 - IF
  2        ?LINE 1+ C/LINE * SCREEN DUP C/LINE - OVER
  3        1024 SCREEN SWAP - CMOVE THEN
  4        15 C/LINE * SCREEN C/LINE BLANKS .SCREEN-TO-END
  5        MODIFY ;
  6  -->
  7


SCR # 126
  0  ( SCREEN EDITOR    )
  1 : BACKUP -1 +.CURSOR ;
  2 : TAB-RIGHT CURSOR $ 8 / 1+ 8 * CURSOR ! CURSOR-CHK
  3             .CURSOR ;
  4 : TAB-LEFT CURSOR $ 1- 8 / 8 * CURSOR ! CURSOR-CHK
  5             .CURSOR ;
  6 : TAB-UP   C/LINE NEGATE +.CURSOR ;
  7 : TAB-DOWN C/LINE +.CURSOR ;
  8 -->
  9

SCR # 127
  0  ( SCREEN EDITOR    )
  1 : ERASE-TO-END CURSOR $ SCREEN 1024 SCREEN OVER - BLANKS
  2                .SCREEN MODIFY ;
  3 : ERASE-LINE ?LINE DUP C/LINE * SCREEN C/LINE BLANKS
  4              DUP .LINE C/LINE * CURSOR ! .CURSOR MODIFY ;
  5 -->
  6
```

```
SCR # 128
  0 ( SCREEN EDITOR      )
  1 : DELETE-CHAR  CURSOR @ SCREEN DUP  1+ SWAP ?LINE 1+
  2                C/LINE * 1- SCREEN DUP >R CURSOR @ SCREEN -
  3                CMOVE 32 R> C! .LINE-TO-END .CURSOR MODIFY ;
  4 : SPREAD-CHAR   CURSOR @ SCREEN ?LINE 1+ C/LINE * SCREEN 1 - -
  5         IF CURSOR @ SCREEN ?LINE 1+ C/LINE * SCREEN 1 -
  6         DO I 1 - C@ I C! -1 +LOOP THEN 32 !(CURSOR)
  7         .LINE-TO-END .CURSOR ;
  8 -->
  9


SCR # 129
  0 ( FORTH EDITOR    )
  1 : INSERT-CHAR .INSERTING .CURSOR BEGIN KEY DUP 32 <
  2   IF DROP 1 ELSE SPREAD-CHAR DUP EMIT !(CURSOR) 1 +.CURSOR
  3   0 THEN UNTIL .MODE .CURSOR ;
  4 -->
  5


SCR # 130
  0 ( FORTH EDITOR    )
  1 0 VARIABLE (EDIT)-FLAG
  2 : (EDIT) 0 (EDIT)-FLAG ! BORDER NEW-SCREEN BEGIN
  3    KEY CASE
  4    29   OF   0 CURSOR ! ERASE-TO-END ENDOF
  5    15   OF   !(SCREEN#) 1 (EDIT)-FLAG ! ENDOF
  6    16   OF   ERASE-LINE ENDOF
  7    05   OF   TAB-UP ENDOF
  8    24   OF   TAB-DOWN ENDOF
  9    04   OF   1 +.CURSOR  ENDOF
 10    19   OF   -1 +.CURSOR ENDOF
 11   -->
 12


SCR # 131
  0   ( FORTH EDITOR    )
  1    06   OF   TAB-RIGHT  ENDOF
  2    25   OF   DELETE-LINE ENDOF
  3    14   OF   SPREAD-LINE ENDOF
  4    09   OF   TAB-RIGHT   ENDOF
  5    1    OF   TAB-LEFT    ENDOF
  6    13   OF   ?LINE 1+ C/LINE * CURSOR ! CURSOR-CHK
  7              .CURSOR     ENDOF
  8    08   OF   0 CURSOR ! .CURSOR ENDOF
  9    18   OF   PREV-SCREEN ENDOF
 10     3   OF   NEXT-SCREEN ENDOF
 11    17   OF   NEW-SCREEN  ENDOF
 12    22   OF   INSERT-CHAR ENDOF
 13   -->
 14
```

```
SCR # 132
  0 ( FORTH EDITOR    )
  1     30 OF     BORDER 0 CURSOR ! .SCREEN ENDOF
  2     127 OF        DELETE-CHAR    ENDOF
  3     26  OF        SPREAD-CHAR    ENDOF
  4     95  OF        SPREAD-CHAR    ENDOF
  5     11  OF   !(SCREEN#) FLUSH .MODE .CURSOR ENDOF
  6     DUP DUP EMIT !(CURSOR) 1 +.CURSOR
  7     ENDCASE (EDIT)-FLAG $ UNTIL ;
  8     -->
  9

SCR # 133
  0 ( SCREEN  EDITOR )
  1 : #IN  0 BEGIN KEY DUP EMIT 10 DIGIT IF SWAP 10 * + 0
  2      ELSE 1 THEN    UNTIL ;
  3 -->
  4

SCR # 134
  0 ( SCREEN EDITOR     )
  1 FORTH DEFINITIONS DECIMAL
  2 : EDIT  EDITOR 0 SCREEN# ! BEGIN 1 MODE ! BORDER .MODE
  3     TERMINAL 20 6 GOTOXY  ." SCREEN EDITOR  BY K.F. "
  4     20 9  GOTOXY  ."  ENTER KOMMAND:  "
  5     KEY DUP 31 > IF DUP EMIT THEN CASE
  6     EDITOR 69 OF 0 MODE ! 20 11 TERMINAL GOTOXY EDITOR
  7     ." ENTER SCREEN NUMBER: " #IN 0 MAX 200 MIN SCREEN# !
  8     (EDIT) ENDOF
  9     15 OF .WRITING FLUSH TERMINAL CLS EDITOR
 10        ." ALL BLOCKS WRITTEN TO DISK." CR CR ." OK" QUIT ENDOF
 11     67 OF    TERMINAL 20 11 GOTOXY EDITOR
 12     ." ENTER SCREEN # TO COPY FROM:  " #IN SCREEN# ! $(SCREEN#)
 13     TERMINAL 20 13 GOTOXY EDITOR ." ENTER SCREEN # TO COPY TO:"
 14     MODIFY 4 SPACES #IN SCREEN# ! !(SCREEN#)  ENDOF
 15     ENDCASE  AGAIN ;  FORTH ;S
```

#

St.Dir. Peter Rix, 2350 Neumünster

FORTH mit Fließkommaarithmetik (2)

Im ersten Teil wurde dargestellt: Die Einrichtung eines besonderen (und vom üblichen Parameterstapel des FORTH unabhängigen) Fließkommastapels ist zweckmäßig, um die zu bearbeitenden Argumente mit ihren unterschiedlichen Formaten einfach und getrennt verwalten zu können. Dieser Fließkommastapel wurde 6 Register tief angelegt. Das oberste Register ist zugleich der Fließkomma-Akkumulator, in dem die Rechenergebnisse zur Verfügung gestellt werden.

Im Gegensatz zum Parameterstapel des FORTH werden Registerinhalte nur bei arithmetischen Operationen verbraucht, nicht jedoch bei Vergleichen, beim Anzeigen, Abspeichern oder Laden, so daß mit ihnen weitergerechnet werden kann.

Fließkommazahlen haben ein Speicherformat (in Konstanten, Variablen und Arrays) von 5 Byte. In den Registern werden sie in einem entpackten 6-Byte-Format gehalten. Die Befehlsworte zum Laden und Speichern übernehmen automatisch das Entpacken und Packen. In das MATHE-ROM des Autors konnten daneben alle notwendigen Umwandlungsroutinen für Float zu ASCII und vice versa und für Integer zu Float und umgekehrt implementiert werden sowie komfortable Möglichkeiten zur Ausgabeformatierung. Ferner sind die höheren mathematischen Funktionen wie in Microsoft BASIC enthalten. -- Die hinter den FORTH-Worten stehenden ausführenden Assembler-Routinen wurden so dokumentiert, daß alle Leistungen des MATHE-ROM auch für die Assembler-Programmierung zur Verfügung stehen (ROM für den Steckplatz hex D000).

Übersicht für die implementierten Befehlsgruppen

Die Befehlsworte sind soweit wie möglich mnemonisch ähnlich aus FIG-FORTH-Worten abgeleitet worden. Die 58 neuen Worte beginnen in der Mehrzahl mit einem F. Sie sind wie folgt zu gruppieren:

1. *Stapeloperationen. Für den Fließkommastapel sind alle vergleichbaren Manipulationen vorgesehen, wie Initialisieren, Duplizieren FDROP, FOVER, FSWAP, FROT (oberste 3 Register Rotieren), alle 6 Register Rollieren, Float zu Integer und umgekehrt mit dem Parameterstapel, .FS (Anzeige des gesamten Stapels).*
2. *Eingabe. FP kennzeichnet die nachfolgende Zahl als Fließkommazahl und befördert sie auf den Datenstack. INPUT unterbricht ein Programm und wartet auf die Eingabe einer solchen Zahl.*
3. *Ausgabe, Ausgabeformate. Systemvariable steuern die Länge des Ausgabefeldes (zwischen 2 und 255) und die Nachkomma-Stellenzahl (plus/minus 127 Stellen). FP. druckt den obersten Stapelwert in diesem Format aus.*
4. *Stapel-Arithmetik. Die vier Grundrechenarten stehen zur Verfügung, ferner Potenzierung, Reziprokwertbildung, Verdoppelung, Halbierung, Vorzeichenumkehr.*
5. *Defining-Worte bilden Fließkomma-Konstante, Variable und Arrays.*
6. *Speicherbezogene Befehle sind für das Laden und Abspeichern von GK-Zahlen aus dem und ins Memory sowie für das Addieren, Subtrahieren, Multiplizieren und Dividieren direkt in Variable hinein implementiert.*
7. *Vergleichsoperationen. 7 Vergleiche auf kleiner, gleich, größer (binäre Vergleiche) und kleiner gleich oder größer Null (unär) vergleichen Akku gegen Memory bzw. Inhalt des Akku. Ein weiterer Befehl läßt den Vergleich der beiden obersten Register zu. Alle Vergleiche können mit NOT auch komplementär ausgeführt werden.*
8. *Funktionen. Neben den von BASIC her bekannten mathematischen Funktionen kann auch die Berechnung von Polynomen ausgeführt werden (n-ten Grades und 2n+1).*
9. *Umwandelungen: STR$ wandelt einen Fließkommawert in ASCII um und legt den String im Memory ab. VAL wandelt einen ASCII-String in eine GK-Zahl und bringt diese auf den Stapel.*

Beispiele

Besser als die vorstehende Übersicht mögen die nachfolgenden Beispiele einen Eindruck geben, wie komfortabel es sich mit einem Fließkommastapel rechnen läßt.

```
<6>
 AIM 65 FORTH V1.3
SOURCE IN=M
( FLIESSKOMMA-BEISPIELE)
DECIMAL
FCL          ( CLEAR STACK)
.FS          ( ALLE REGISTER ANZEIGEN)

       0.000000000000
       0.000000000000
```

```
      0.000000000000
      0.000000000000
      0.000000000000
      0.000000000000
20 FLEN !   ( FELDLAENGE =20)
12 DPP !    ( 12 NACHKOMMASTELLEN FESTLEGEN)
FP 1.22     ( ZAHLENEINGABE)
FP 8.33E14  ( ZAHL MIT EXPONENT)
F*          ( MULTIPLIKATION)
FP.         ( ANZEIGE AKKU-REGISTER)
  1.016260000000E+15 FDUP        ( DUPLIZIEREN)
FROT        ( 3 OBERSTE WERTE ROTIEREN)
.FS         ( STAPELANZEIGE)

      0.000000000000
  1.016260000000E+15
  1.016414420000E+15
      0.000000000000
      0.000000000000
      0.000000000000
FDROP       ( STACK ABBAUEN)
FP.         ( ANZEIGE GK-AKKU)
  1.016414420000E+15 L
FP 20 FCON ALPHA   ( ANLAGE DER KONSTANTEN 20)
FP .001 FVAR BETA   ( ANLAGE DER VARIABLEN BETA MIT .001)
ALPHA BETA F$ F+   ( ADDITION VON APHA UND BETA)
FP.                ( ANZEIGE DER SUMME)
     20.001000000000 BETA F*!          ( BETA=BETA*AKKUINHALT)
BETA F$ FP.        ( ANZEIGE BETA)
      0.020001000000
PI COS FP.         ( COSINUS PI ANZEIGEN)
     -1.000000000000 FINIS

OK                                                              #
```

FORTH-Splitter (2)

In Heft 28 zeigten wir bereits, wie man aus FORTH auf die TOP-Zeile des AIM-Editors aufsetzen kann. Eine zweite Version ist:

```
HEX ASSEMBLER
CREATE T F6CF JMP, SMUDGE
```

Mit T springt man damit auf den Label REENTR des Editors. – Source-Texte aus dem Editor können mit dem folgenden Wort L während des Compilierens gelistet werden. L listet immer die Folgezeile, es sollte daher z.B. unsichtbar in Zwischenzeilen stehen:

```
CREATE L F727 JSR, NEXT JMP, SMUDGE IMMEDIATE
```

Beispiel: Im Editor steht als Quelltext

```
CR L
( KOMMENTAR)
L
FINIS
```

In FORTH gibt man dann SOURCE IN=M und erhält:

```
( KOMMENTAR)
FINIS
```

Das CASE-Statement ist in einer sehr strukturierten Form in diesem Heft benutzt worden, und zwar im FORTH-Editor von Klaus Flesch. Eine andere mehr hausgemachte Version des Herausgebers benutzt das EXECUTE des FORTH. Dieses Statement setzt voraus, daß sich eine Codefeldadresse auf dem Datenstack befindet, die exekutiert wird.

In einem mit NameX versehenen Array (hier heißt es FALL) vom Typ CASES (defining word) sind die Codefeldadressen für die Normalfälle sowie für den Fall OTHERWISE enthalten, der bei Bereichsüberschreitung in der Adressierung eines Falles angesprungen wird. Damit handelt es sich nicht um einen Aufzählungstyp des Arrays, sondern um einen Stellungstyp, der mit der Angabe eines Stellungsparameters benutzt werden muß. Das 'defining word' CASES bringt zur Ausführungszeit eine Codefeldadresse auf den Datenstack.

```
ON SOURCE IN=M
: 0CASE ." NULL" CR ;
: 1CASE ." EINS" CR ;
: OTHER ." OTHERWISE" CR ;

: CASES     ( DEFINING WORD)
<BUILDS ,   ( STORE # OF ELEMENTS WHEN COMPILING)
DOES>       ( AT EXECUTION TIME)
DUP @       ( LOOK AT MAX ITEMS)
ROT         ( ADDR MAX #)
DUP >R      ( SAVE ADDRESSED ITEM)
< IF R> DROP DUP @ 1+ ( POSITION OF OTHERWISE)
  ELSE R>   ( NORMAL CASE)
  THEN
1+ 2 * + @    ( GET CFA OF ADDRESSED ITEM)
EXECUTE
;
2 CASES FALL    ( AUFBAU EINES CASE-ARRAY)
' 0CASE CFA ,
' 1CASE CFA ,
' OTHER CFA ,

( ANWENDUNG VON FALL)
0 FALL
NULL
1 FALL
EINS
2 FALL
OTHERWISE
FINIS
```

In den strukturierten Sprachen FORTH, PASCAL u.a.m. ist es üblich, die logische Ebene der Konstrukte IF, ELSE, THEN usw. z.T. sehr tief in die Zeilen einzurücken, um die Übersicht zu behalten. Das verbraucht je nach Editor viel Platz und führt auch zu unnötigem Aufwand beim Rangieren in die richtige Spalte. Beim Herausgeber hat es sich bewährt, die logische Ebene als Kommentar (in Klammern) vor das Konstrukt zu setzen. Das hat auch den Vorteil, daß man sich bei Verschachtelungen nicht so schnell in der richtigen Zahl der Kontrollworte verzählt. Ein Quelltext sieht dann z.B. wie folgt aus:

```
( 1) IF LOGISCHE EBENE 1
( 2) IF LOGISCHE EBENE 2
( 2) ELSE
( 2) THEN
( 1) ELSE WIEDER LOGISCHE EBENE 1
( 1) THEN
```

Herr Michael Beland aus 2206 Sparrieshoop sandte folgenden FORTH-Splitter: **Kopieren der äußeren Laufvariablen in DO LOOPs.** In einer DO LOOP kann man mit 'I' den Wert der Laufvariablen auf den Datenstack kopieren. Öfters besteht jedoch auch der Wunsch, bei einer geschachtelten Schleife aus der inneren Schleife auf die Laufvariable der äußeren Schleife zugreifen zu können. Das nachstehende Wort J ermöglicht dieses. Bei noch tieferer Schachtelung sind mit jeder Stufe +4 zu den Adressen 105 bzw. 106 in der Stackseite zu addieren.

```
CODE J ( ---N)
( E/T: KOPIERT LAUFVARIABL. AEUSS. SCHLEIFE AUF DEN STACK)
XSAVE STX,
TSX,
105 ,X LDA,
PHA,
106 ,X LDA,
XSAVE LDX,
PUSH JMP,
END-CODE

: //  ( N1 N2 ---) ( TESTWORT Z. ERPROBUNG. N1 =END-)
( WERT-1 DER INNEREN, N2 DER AUESSEREN SCHLEIFE)
0 DO DUP 0 DO CR ." J " J . ."   I " I . LOOP LOOP DROP ;
FINIS
```

R.L. #

Dr. A. Schnell, 5100 Aachen

FORTH mit JSR

Einfache Anbindung maschinensprachlicher Routinen an High-Level FORTH

Von der FORTH-Kommandoebene sind Sprünge in Unterprogramme, z.B. des Monitors nicht ohne weiteres möglich. In der Regel muß man sich jeweils ein entsprechendes CODE-Wort mit dem Assembler des FORTH aufbauen. Das im Screen 31 aufgelistete FORTH-Wort JSR hingegen wird nur einmal definiert. Nach Aufruf von der Kommandoebene oder innerhalb einer Wortdefinition wird z.B. in eine Monitorroutine gesprungen und anschließend wieder in das FORTH-System zurückgekehrt. Die Sprungadresse ist zuvor auf dem Datenstack zu übergeben. – Viele BASIC-Versionen bieten eine entsprechende Möglichkeit durch das Schlüsselwort SYS adr., wobei adr. die dezimale Sprungadresse ist. Im Gegensatz zu diesem SYS-Befehl können jedoch beim FORTH-Wort JSR die internen Prozessorregister A, X und Y beliebig vorbereitet werden, indem die Variablen ACCU, XREG und YREG entsprechend gesetzt werden. Nach Abarbeitung des JSR enthalten diese Variablen dann den Inhalt der Prozessorregister bei Ende des Unterprogrammes. Damit ist JSR ein wertvolles Hilfsmittel, um neue FORTH-Worte zu definieren, die von den Möglichkeiten vorhandener Systemroutinen Gebrauch machen.

```
SCR # 31
  0 HEX
  1 0 VARIABLE ACCU
  2 0 VARIABLE XREG
  3 0 VARIABLE YREG
  4
  5 CODE JSR ( ADR. --- )
  6   XSAVE STX,
  7   4C # LDA, N STA,
  8   BOT LDA, N 1+ STA,
  9   BOT 1+ LDA, N 2 + STA,
 10   ACCU LDA, XREG LDX, YREG LDY,
 11   N JSR,
 12   ACCU STA, XREG STX, YREG STY,
 13   XSAVE LDX, POP JMP,
 14 ( END-CODE)
 15 ABORT
```

Anmerkung für AIM-FORTH: Statt BOT TOP einsetzen und in Zeile 14 das Wort END–CODE ohne Kommentarklammern verwenden!

#

FORTH (6)

16. Ausgabeformatierung

Zur Tabulation in Druckspalten stehen uns folgende Befehlsworte zur Zahlenausgabe zur Verfügung:

.R (n1 n2 ---) Tabulation und Ausgabe einer einfach genauen Zahl n1 rechtsbündig in einem Schreibfeld, das um n2 Stellen rechts von der erreichten Stelle endet. Die niederwertigste Ziffer steht also um n2 Stellen weiter rechts.

D.R (d n ----) für doppelt genaue Zahlen. Wirkt wie .R. Die niederwertigste Ausgabeziffer ist um n Stellen nach rechts tabuliert.

Nachfolgend einige Anwendungsbeispiele, die auch zeigen, wie man Worte bildet, die in Kolonnen tabulieren, auch gemischt mit Text.

```
 AIM 65 FORTH V1.3
SOURCE IN=M
." ZAHL" 44 5 .R CR    ( AUSGABE IN GLEICHE ZEILE)
ZAHL   44
33 10 44 10    CR .R .R CR   ( AUSGABE IN FOLGEZEILE)

         44          33
-44444.55 12 CR D.R CR     ( 32 BIT ZAHL)

    -4444455
: .TAB CR 10 .R 10 .R 10 .R ;    ( DRUCKT 3 KOLONNEN)
111 222 333 .TAB CR 444 555 666 .TAB CR

        333         222         111

        666         555         444
: .TTAB CR ." DRUCK" 10 .R SPACE ." TEMPERATUR" 10 .R ;
150 543 .TTAB

DRUCK         543 TEMPERATUR         150
OK
```

Für die Ausgabeformatierung doppelt genauer Zahlen (32 Bit) bietet FORTH zahlreiche Möglichkeiten:

<# (d --- d) leitet die Konvertierung einer vorzeichenfreien Zahl in einen ASCII-String ein. Wird paarweise mit dem nachfolgenden Befehlswort gebraucht.

#> (d --- adr n) schließt die Konversion nach ASCII ab und hinterläßt für den Ausgabebefehl TYPE die Länge und die Anfangsadresse des gebildeten Strings. Die Adresse weist auf das PAD (Arbeitsspeicher des FORTH), so daß hier zwischenzeitlich (vor TYPE oder einer anderen Übernahme) keine Störungen verursacht werden dürfen.

Die beiden vorgenannten Befehle bilden somit die Eckpunkte der Konversion. Die eigentliche Umwandlung und Formatierung geschieht durch die nachstehenden Befehle.

(ud1 --- ud2) Dieser Befehl erzeugt eine Ziffer des Ausgabestrings, codiert in ASCII. Die Zahl ud1 wird dabei um eine Potenz zur Zahlenbasis abgebaut. Oder: Dieser Befehl faßt bei der jeweils niederwertigsten Ziffer an. Er erzeugt eine Null, wenn ud1 bereits zu Null abgebaut ist, kann daher in besonderen Fällen benutzt werden, um z.B. führende Nullen zu erzeugen.

#S (ud —)	'sharp-s' erzeugt einen Ausgabestring für alle bzw. die noch nicht konvertierten Ziffern von ud. Liefert mindestens eine 0 ab, wenn ud=0. Führende Nullen werden unterdrückt.
SIGN (n d — d)	bringt das Vorzeichen von n in den Ausgabestring. Notwendiger Befehl, weil o.a. Konversion vorzeichenfrei arbeitet.
HOLD (ASCII —)	bringt einen ASCII-Wert in den Ausgabestring.

Zu den nachfolgenden Beispielen für den Gebrauch dieser Befehle ist zu bemerken: Bei der Konversion werden führende Nullen immer unterdrückt. Gleichwohl kann man willentlich führende Nullen erzeugen, ebenso Schutzsterne, Währungszeichen, Punkt und Komma oder andere Trenner. In den vorliegenden Befehlen wird jedoch nicht der Inhalt des DPL (Decimal Point Location) verwertet, die Stellung des Dezimalkommas in der ursprünglichen doppelt genauen Zahl d. Eine Abhilfe für variabel zu erzeugende Nachkommastellen ist das WORT STR$ aus 'Strings für FORTH' (Heft 25), das hier zur Vervollständigung aufgeführt wird. Am Aufbau dieses Wortes sieht man, daß zwischen den Endpunkten der Konversion Wiederholungs- und Bedingungskonstrukte enthalten sein dürfen (hier IF ... THEN und BEGIN ... WHILE ... REPEAT). Im Bedarfsfalle wird man also mit einem ähnlichen Wort auch Ausgabebefehle bilden können, die trotz Fließkomma kolonnengerecht drucken.

```
55. ( ZAHL AUF DEM STACK)
<# #S #>   ( NORMALFALL DER KONVERSION, ALLE ZIFFERN)

TYPE   ( AUSGABE DES STRINGS)
55
123.45 <# # # 46 HOLD     ( 2 NACHKOMMASTELLEN UND PUNKT)
          #S #>           ( UEBRIGE STELLEN, ABSCHLUSS)
TYPE   ( AUSGABE STRING)
123.45: VZ SWAP OVER DABS ; ( VORZEICHEN BILDEN, ABS-WERT)

-123.45 VZ   ( VORZEICHENBYTE)
<# # # 44 HOLD #S    ( MIT DEZIMALKOMMA KONVERT.)
SIGN #>   ( VORZEICHEN DAVOR STELLEN)
TYPE
-123,45
-124.89
( VORZEICHEN NACHGESTELLT, KOMMA)
VZ <# SIGN # # 44 HOLD #S
( SCHUTZSTERN 'DM')
42 HOLD 77 HOLD 68 HOLD #>
TYPE
DM*124,89-

: STR$   ( D --- STRING)
( KONVERTIERT MIT FLIESSKOMMA NACH ASCII)
  VZ <# DPL @ 0= NOT
  IF BEGIN DPL @ 0= NOT
     WHILE # DPL @ 1- DPL !
     REPEAT
  44 HOLD
  THEN
  #S SIGN #>
;

98.12345 STR$ CR TYPE CR

98,12345
```

Martin Albrecht, 4350 Recklinghausen

Symbolisches Differenzieren (BASIC)

Das Differenzieren arithmetischer Ausdrücke ist in feste Regeln gefaßt und daher eine echte Computeraufgabe.

Zum Differenzieren ist es erforderlich, den arithmetischen Ausdruck zu zerlegen. Versucht man einen Ausdruck von links nach rechts abzuarbeiten und erreicht dabei einen Operator, dann hängt die Entscheidung, ob er ausgewertet werden darf, auch von dem nachfolgenden Operator ab. Hat dieser nämlich eine höhere Priorität, so muß der erste Operator zwischengespeichert werden, um zu versuchen, zunächst den Nachfolger auszuwerten. Beipiel:

A+B*C Operatorenpaar: +,*
Hier muß zunächst der *-Operator ausgewertet werden: B*C.
Erst danach darf der +-Operator ausgewertet werden: A + B*C.

Klammert man einen Ausdruck in ein Anfangszeichen (') und ein Schlußzeichen (;) und betrachtet man ein Operatorenpaar aus der Menge (: ,Vorzeichen, Funktion, =, +, -, *, /, ↑, (,), ;) , so können die folgenden Fälle eintreten:

1) Die Folge der Operatoren ist nicht erlaubt.
2) Der rechte Operator hat einen höheren Rang. Der linke Operator wird zwischengespeichert und das nächste Operatorenpaar betrachtet.
3) Der Inhalt einer Klammer ist ausgewertet, die Klammern können entfallen.
4) Der Ausdruck ist fertig bearbeitet.
5) Der linke Operator hat einen höheren oder gleichen Rang wie der rechte Operator und kann sofort ausgewertet werden.

Diese Fälle sind durch die Übergangsmatrix in den Zeilen 1240 bis 1330 des u.a. Programmes realisiert. Das Flußdiagramm zeigt die Verarbeitung der Operatoren und Operanden. Zur Zwischenspeicherung werden drei Stapelspeicher mit den Zeigern Q und P verwendet.: Dabei sind Opstack=Operanden, Abstack=abgeleitete Operanden, Delstack=Operatoren

Übersetzung bedeutet hier Zusammenfassung des Operators und seiner Operanden zu einem neuen Operanden. Beispiel:

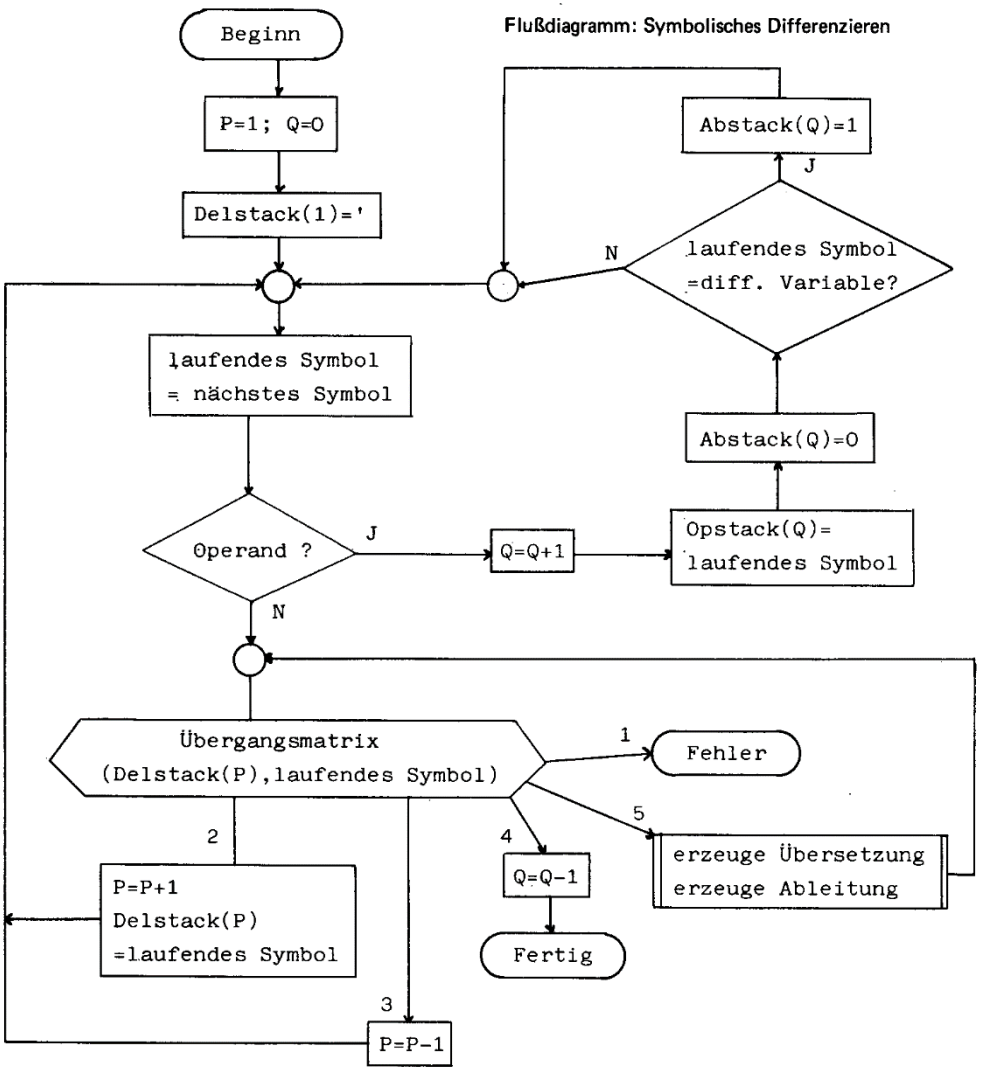
A, B seien die beiden letzten Operanden auf dem Operandenstack, * der letzte Operator auf dem Operatorenstack. Dann wird A*B zu einem neuen Operanden zusammengefaßt. B und * verschwinden vom Stack, A wird durch A*B ersetzt.

Das Zusammenfassen übernimmt das Unterprogramm ab Zeile 4020.

Das vorliegende Programm wurde auf dem AIM 65 entwickelt und läuft daher auch auf anderen Microsoft BASICs. -- Sämtliche Kommentare können weggelassen werden. Die Tabelle der ableitbaren Funktionen (Zeilen 1420 bis 1450) läßt sich nach dem angegebenen Schema erweitern, FZ in Zeile 1470 ist entsprechend zu erhöhen.

```
1000 REM SYMBOLISCHE DIFFERENTIATION ALGEBRAISCHER AUSDRUECKE
1010 REM (C) 1982 MARTIN ALBRECHT
1020 REM          AM BAERENBACH 18
1030 REM          4350 RECKLINGHAUSEN
1040 REM
1050 REM ZUGELASSENE OPERATOREN:
1060 REM MONADISCH: FUNKTIONSOPERATOR,VORZEICHENOPERATOR
1070 REM DYADISCH: =,+,-,*,/,↑
1080 REM
1090 REM VORRANG: ( ),VORZEICHEN,FUNKTION,↑,*/,=+-
1100 REM
1110 REM MEHRFACHZUWEISUNGEN SIND ERLAUBT
```

Flußdiagramm: Symbolisches Differenzieren




```
1120 REM "*"-ZEICHEN DUERFEN NICHT WEGGELASSEN WERDEN
1130 REM EXPONENTIALZAHLEN SIND NICHT ERLAUBT
1140 REM
1150 REM ZUGELASSENE NAMEN
1160 REM NAMEN DUERFEN NICHT MIT FUNKTIONSNAMEN UEBEREIN-
1170 REM STIMMEN, DIESE JEDOCH ENTHALTEN.
1180 REM SIGNIFIKANTE STELLEN <= 255
1190 REM
1200 REM DATENTABELLEN:
1210 REM
```

```
1220 REM UEBERGANGSMATRIX:
1230 REM  & £ = + - * / ↑ ( ) ;
1240 DATA 2,2,2,2,2,2,2,2,2,1,4 :REM '
1250 DATA 2,2,5,5,5,5,5,5,2,2,5,5 :REM &
1260 DATA 1,5,1,5,5,5,5,5,2,5,5 :REM £
1270 DATA 2,2,2,2,2,2,2,2,2,1,5 :REM =
1280 DATA 2,2,1,5,5,2,2,2,2,5,5 :REM +
1290 DATA 2,2,1,5,5,2,2,2,2,5,5 :REM -
1300 DATA 2,2,1,5,5,5,5,5,2,2,5,5 :REM *
1310 DATA 2,2,1,5,5,5,5,5,2,2,5,5 :REM /
1320 DATA 2,2,1,5,5,5,5,5,5,2,5,5 :REM ↑
1330 DATA 2,2,1,2,2,2,2,2,2,3,1 :REM (
1340 REM
1350 REM BEGRENZER:
1360 DATA ',&,£,=,+,-,*,/,↑,(,),;
1370 REM "&"=VORZEICHEN, "£"=FUNKTION ALLGEMEIN
1380 REM "'"=ANFANGSZEICHEN, ";"=SCHLUSSZEICHEN
1390 REM
1400 REM FUNKTIONSTABELLE:
1410 REM AUFBAU: (FUNKTION,ABLEITUNG),(...
1420 DATA SIN,COS(!),COS,-SIN(!),EXP,EXP(!)
1430 DATA SQR,1/(2*SQR(!)),LN,1/(!)
1440 DATA TG,1/(COS(!)↑2),CTG,-1/(SIN(!)↑2)
1450 DATA ARCTG,1/(1+(!)↑2),ARCCTG,-1/(1+(!)↑2)
1460 REM
1470 FZ=9: REM ANZAHL FUNKTIONEN
1480 REM
1490 REM DATENSTRUKTUREN:
1500 DIM UM(10,11)  :REM UEBERGANGSMATRIX
1510 DIM FL$(FZ)    :REM FUNKTIONSLISTE
1520 DIM FA$(FZ)    :REM DIFF. FUNKTIONEN
1530 DIM FS(15)     :REM FUNKTIONSSTACK
1540 DIM O$(12)     :REM OPERATORENLISTE
1550 DIM DS$(30)    :REM BEGRENZERSTACK
1560 DIM OS$(30)    :REM OPERANDENSTACK
1570 DIM AO$(30)    :REM DIFF. OPERANDEN
1580 REM
1590 REM ERZEUGUNG DER TABELLEN:
1600 RESTORE
1610 REM UEBERGANGSMATRIX:
1620 FOR X=1 TO 10: FOR Y=1 TO 11
1630 READ UM(X,Y): NEXT Y,X
1640 REM
1650 REM OPERATORENLISTE:
1660 FOR X=1 TO 12: READ O$(X): NEXT
1670 REM
1680 REM LISTE DER FUNKTIONEN UND IHRER ABLEITUNGEN:
1690 FOR X=1 TO FZ: READ FL$(X),FA$(X): NEXT
1700 REM
1710 REM INITIALISIERUNG DER PARAMETER:
1720 Q=0          :REM STAPELZEIGER OPERANDEN
1730 P=1          :REM STAPELZEIGER BEGRENZER
1740 SZ=0         :REM STAPELZEIGER FUNKTIONSSTACK
1750 DS$(1)="'"   :REM ANFANGSBEGRENZER
1760 LE$="'"      :REM ANFANGSBEGRENZER
1770 IZ=0         :REM EINGABEZEIGER
1780 REM
1790 REM BESCHAFFE FORMEL ZUR UEBERSETZUNG:
```

```
1800 INPUT"FORMEL";F$
1810 INPUT"ABLEITEN NACH";DX$
1820 IF RIGHT$(F$,1)<>";" THEN F$=F$+";"
1830 REM
1840 REM BEGINN HAUPTPROGRAMM
1850 REM
1860 REM SCANNER:
1870 REM DER SCANNER LIEST DEN EINGABESTRING F$ UND
1880 REM ERKENNT OPERANDEN UND BEGRENZER.
1890 REM OPERANDEN UND IHRE ABLEITUNG WERDEN DIREKT AUF DEN
1900 REM JEWEILIGEN STACK GESCHOBEN. DAS ERKENNEN EINES
1910 REM BEGRENZERS FUEHRT ZUM VERLASSEN DES SCANNERS.
1920 I$=""
1930 IZ=IZ+1: CS$=MID$(F$,IZ,1)
1940 CV=0
1950 FOR X=1 TO 12
1960 IF CS$=O$(X) THEN CV=X
1970 NEXT
1980 IF CV<>0 THEN 2010
1990 I$=I$+CS$: GOTO 1930
2000 GOTO 1930
2010 IF I$="" GOTO 2130
2020 IZ=IZ-1
2030 CV=0
2040 FOR X=1 TO FZ
2050 IF I$=FL$(X) THEN CV=X
2060 NEXT
2070 IF CV=0 GOTO 2090
2080 CS$="£": SZ=SZ+1: FS(SZ)=CV: CV=3: GOTO 2130
2090 Q=Q+1: OS$(Q)=I$: AO$(Q)="0"
2100 IF I$=DX$ THEN AO$(Q)="1"
2110 LE$=I$
2120 GOTO 1920
2130 IF CS$<>"-" GOTO 2170
2140 T$=LE$: GOSUB 4470
2150 IF NOT V GOTO 2170
2160 CV=2: CS$="&"
2170 LE$=CS$
2180 REM
2190 REM DER SCANNER HAT EINEN BEGRENZER GEFUNDEN.
2200 REM UEBERGANGSMATRIX(BEGR. AUF DEM STACK,NEUER BEGR.)
2210 REM LEGT DAS WEITERE VORGEHEN FEST (FAELLE 1 BIS 5).
2220 REM
2230 FOR X=1 TO 12
2240 IF DS$(P)=O$(X) THEN DV=X
2250 NEXT
2260 RS=UM(DV,CV-1)
2270 ON RS GOTO 2300,2370,2420,2460,2560
2280 REM
2290 REM -------- RS=1; FEHLER IN DER FORMEL
2300 PRINT" "
2310 PRINT"FEHLER:"
2320 PRINT F$
2330 PRINT TAB(IZ) "↑"
2340 END
2350 REM
2360 REM -------- RS=2; RECHTER OPERAND NICHT ABGEARBEITET
2370 P=P+1
2380 DS$(P)=CS$
```

```
2390 GOTO1920
2400 REM
2410 REM -------- RS=3; KLAMMERPAAR UEBERFLUESSIG
2420 P=P-1
2430 GOTO1920
2440 REM
2450 REM -------- RS=4; ABLEITUNG BEENDET
2460 Q=Q-1
2470 PRINT"ABLEITUNG VON ";LEFT$(F$,LEN(F$)-1);" NACH ";DX$;" :"
2480 PRINTAO$(1)
2490 END
2500 REM
2510 REM -------- RS=5; UEBERSETZUNG UND ABLEITUNG
2520 REM
2530 REM ERZEUGE ZUNAECHST DIE UEBERSETZUNG:
2540 REM
2550 REM FUNKTION:
2560 IF DS$(P)<>"£" GOTO 2610
2570 G$=FL$(FS(SZ))+"("+OS$(Q)+")"
2580 GOTO 2670
2590 REM
2600 REM VORZEICHEN:
2610 IF DS$(P)<>"&" GOTO 2660
2620 UT$=OS$(Q): GOSUB 2820: G$=UT$
2630 GOTO 2670
2640 REM
2650 REM ALLE UEBRIGEN OPERATOREN
2660 L$=OS$(Q-1): DL$=DS$(P): R$=OS$(Q): GOSUB 4020: G$=RT$
2670 AG$=""
2680 REM
2690 REM STELLE OPERATOR FEST
2700 FOR X=1 TO 10
2710 IF DS$(P)=O$(X) THEN JP=X
2720 NEXT
2730 REM
2740 REM ZUR ABLEITUNG DER JEWEILIGEN OPERATOREN
2750 REM IN DER REIHENFOLGE: VORZEICHEN,FUNKTION,=,+,-,*,/,↑
2760 ON JP GOTO 0,2800,2920,3040,3100,3100,3200,3200,3420
2770 REM
2780 REM -------- ABLEITUNG DES VORZEICHENS
2790 REM (-U)' WIRD -(U')
2800 UT$=AO$(Q): GOSUB 2820: AO$(Q)=UT$
2810 OS$(Q)=G$: GOTO 3710
2820 IF UT$="0" THEN RETURN
2830 GOSUB 3810: IF LE<2 THEN UT$="("+UT$+")"
2840 IF LEFT$(UT$,1)<>"-" GOTO 2860
2850 UT$=MID$(UT$,2): RETURN
2860 UT$="-"+UT$: RETURN
2870 REM
2880 REM -------- ABLEITUNG DER FUNKTIONEN
2890 REM DIE AEUSSERE ABLEITUNG WIRD DER TABELLE FA$
2900 REM ENTNOMMEN UND DER FUNKTIONSOPERAND EINGESETZT.
2910 REM DIE INNERE ABLEITUNG WIRD DARAUFHIN ANGEFUEGT
2920 ZW$=FA$(FS(SZ)): Z$=""
2930 FOR X=1 TO LEN(ZW$)
2940 UT$=MID$(ZW$,X,1)
2950 IF UT$="!" THEN UT$=OS$(Q)
2960 Z$=Z$+UT$
2970 NEXT
```

```
2980 L$=Z$: DL$="*": R$=AO$(Q): GOSUB 4020
2990 OS$(Q)=G$: AO$(Q)=RT$: SZ=SZ-1
3000 GOTO 3710
3010 REM
3020 REM -------- ABLEITUNG DES "="-OPERATORS
3030 REM (U=V)' WIRD U'=V'
3040 AG$=AO$(Q-1)+DS$(P)+AO$(Q)
3050 AO$(Q-1)=AG$
3060 GOTO 3690
3070 REM
3080 REM -------- ABLEITUNG DER "+" UND "-"-OPERATOREN
3090 REM (U+V)' WIRD U'+V' BZW. U'-V'
3100 L$=AO$(Q-1): DL$=DS$(P): R$=AO$(Q): GOSUB 4020
3110 AO$(Q-1)=RT$: GOTO 3690
3120 REM
3130 REM -------- ABLEITUNG DER "*" UND "/"-OPERATOREN
3140 REM ABLEITUNG NACH PRODUKTREGEL FUER "*":
3150 REM (U*V)' WIRD U'*V+U*V'
3160 REM (U/V)' WIRD (U'*V-U*V')/V↑2
3170 REM IM "/"-FALL WIRD NACH PRODUKTREGEL ABGELEITET, WOBEI
3180 REM DAS "+" DURCH EIN "-" ERSETZT WIRD UND DER NENNER
3190 REM V↑2 ERGAENZT WIRD.
3200 Z$="+"
3210 IF DS$(P)="/" THEN Z$="-"
3220 L$=AO$(Q-1): DL$="*": R$=OS$(Q): GOSUB 4020: ZW$=RT$
3230 L$=OS$(Q-1): DL$="*": R$=AO$(Q): GOSUB 4020
3240 L$=ZW$: DL$=Z$: R$=RT$: GOSUB 4020
3250 IF DS$(P)="*" GOTO 3300
3260 UT$=OS$(Q)
3270 GOSUB 3810
3280 IF LE<3 THEN UT$="("+UT$+")"
3290 L$=RT$: DL$="/": R$=UT$+"↑2": GOSUB 4020
3300 AO$(Q-1)=RT$
3310 GOTO 3690
3320 REM
3330 REM
3340 REM -------- ABLEITUNG DES "↑"-OPERATORS
3350 REM UNTERTEILUNG IN 3 FAELLE (ABLEITUNG NACH X):
3360 REM 1) (X↑Y)' WIRD Y*X↑(Y-1)
3370 REM 2) (Y↑X)'=EXP(X*LN(Y))'
3380 REM (Y↑X)' WIRD ALSO EXP(X*LN(Y))*X'*LN(Y)
3390 REM 3) (X↑X)'=EXP(X*LN(X))'
3400 REM (X↑X)' WIRD EXP(X*LN(X))*(X'*LN(X)+X*LN(X)')
3410 REM
3420 IF AO$(Q-1)<>"0" OR AO$(Q)<>"0" GOTO 3440
3430 AO$(Q-1)="0": GOTO 3690
3440 IF AO$(Q)<>"0" GOTO 3520
3450 REM FALL 1: X↑Y
3460 L$=OS$(Q): DL$="-": R$="1": GOSUB 4020
3470 L$=OS$(Q-1): DL$="↑": R$=RT$: GOSUB 4020
3480 L$=OS$(Q): DL$="*": R$=RT$: GOSUB 4020
3490 L$=RT$: DL$="*": R$=AO$(Q-1): GOSUB 4020
3500 AO$(Q-1)=RT$: GOTO 3690
3510 REM FAELLE 2 UND 3: Y↑X UND X↑X
3520 Z$="LN("+OS$(Q-1)+")"
3530 L$=OS$(Q): DL$="*": R$=Z$: GOSUB 4020
3540 AG$="EXP("+RT$+")"
3550 IF AO$(Q-1)<>"0" GOTO 3610
```

```
3560 REM FALL 2: Y↑X
3570 L$=Z$: DL$="*": R$=AO$(Q): GOSUB 4020
3580 L$=AG$: DL$="*": R$=RT$: GOSUB 4020
3590 AO$(Q-1)=RT$: GOTO 3690
3600 REM FALL 3: X↑X
3610 L$=AO$(Q): DL$="*": R$=Z$: GOSUB 4020: Z$=RT$
3620 L$="1": DL$="/": R$=OS$(Q-1): GOSUB 4020
3630 L$=RT$: DL$="*": R$=AO$(Q-1): GOSUB 4020
3640 L$=OS$(Q): DL$="*": R$=RT$: GOSUB 4020
3650 L$=Z$: DL$="+": R$=RT$: GOSUB 4020
3660 L$=AG$: DL$="*": R$=RT$: GOSUB 4020
3670 AO$(Q-1)=RT$
3680 REM
3690 OS$(Q-1)=G$
3700 Q=Q-1
3710 P=P-1
3720 GOTO 2230
3730 REM
3740 REM UNTERPROGRAMM ZUR KLASSIFIZIERUNG EINES STRINGS
3750 REM IN 3 KLASSEN IN ABHAENGIGKEIT DER VORHANDENEN
3760 REM OPERATOREN
3770 REM KLASSE 1: "+" UND "-" OPERATOR
3780 REM KLASSE 2: "*" UND "/" OPERATOR
3790 REM KLASSE 3: ZAHLEN,VARIABLEN,GEKLAMMERTE AUSDRUECKE
3800 REM SOWIE DIE OPERATOREN "=","↑" UND VORZEICHEN
3810 LE=3: LA$="'"
3820 IF LEN(UT$)=1 THEN LA$="A"
3830 KZ=0: R1=0: R2=0: R3=0: H=0
3840 FOR X=1 TO LEN(UT$)
3850 K$=MID$(UT$,X,1)
3860 KZ=KZ+(K$=")")-(K$="(")
3870 T$=LA$: V=0: IF K$="-" THEN GOSUB 4470
3880 IF K$="+" OR (K$="-" AND NOT V) THEN R1=R1-(KZ=0)
3890 IF K$="*" OR K$="/" THEN R2=R2-(KZ=0)
3900 LA$=K$: IF K$="↑" THEN H=1
3910 NEXT
3920 IF R2>0 THEN LE=2
3930 IF R1>0 THEN LE=1
3940 RETURN
3950 REM
3960 REM
3970 REM UNTERPROGRAMM ZUR KLAMMERUNG VON AUSDRUECKEN.
3980 REM ZWEI OPERANDEN WERDEN NACH IHRER KLASSE
3990 REM GEKLAMMERT UND DURCH EINEN OPERATOR VERKNUEPFT.
4000 REM SIND BEIDE OPERANDEN ZAHLEN, SO WIRD DAS
4010 REM ERGEBNIS FUER DIE OPERATOREN +,-,*,/,↑ BERECHNET.
4020 UT$=L$: GOSUB 3810: L1=LE: H1=H
4030 UT$=DL$: GOSUB 3810: L2=LE
4040 UT$=R$: GOSUB 3810: L3=LE: H3=H
4050 IF DL$="=" GOTO 4200
4060 IF L1<3 OR L3<3 OR (H1+H3) GOTO 4090
4070 V1=VAL(L$): V2=VAL(R$)
4080 IF (V1<>0 OR L$="0") AND (V2<>0 OR R$="0") GOTO 4350
4090 IF L2>1 GOTO 4190
4100 IF LEFT$(R$,1)<>"-" GOTO 4140
4110 IF DL$="+" GOTO 4130
4120 IF DL$="-" THEN R$="+"+MID$(R$,2)
4130 DL$=""
4140 RT$=L$+DL$+R$
```

```
4150 IF L$="0" THEN RT$=DL$+R$
4160 IF R$="0" THEN RT$=L$
4170 IF LEFT$(RT$,1)="+" THEN RT$=MID$(RT$,2)
4180 RETURN
4190 IF L2<2 GOTO 4300
4200 RT$=L$: IF L1<L2 THEN RT$="("+RT$+")"
4210 RT$=RT$+DL$
4220 IF (L3<L2) OR (DL$="/" AND L3<3) THEN UT$="("+UT$+")"
4230 RT$=RT$+UT$
4240 IF L2<2 THEN RETURN
4250 IF L$="1" THEN RT$=R$
4260 IF L$="1" AND DL$="/" THEN RT$="1/"+UT$
4270 IF R$="1" THEN RT$=L$
4280 IF L$="0" OR R$="0" THEN RT$="0"
4290 RETURN
4300 GOSUB 4200
4310 IF DL$<>"↑" GOTO 4340
4320 IF R$="0" THEN RT$="1"
4330 IF R$="1" THEN RT$=L$
4340 RETURN
4350 IF DL$="+" THEN RT=V1+V2
4360 IF DL$="-" THEN RT=V1-V2
4370 IF DL$="*" THEN RT=V1*V2
4380 IF DL$="/" THEN RT=V1/V2
4390 IF DL$="↑" THEN RT=V1↑V2
4400 RT$=STR$(RT)
4410 IF RT<>0 THEN RT$=MID$(RT$,2)
4420 RETURN
4430 REM
4440 REM
4450 REM UNTERPROGRAMM ZUR UNTERSCHEIDUNG
4460 REM DES VORZEICHENS VOM DYADISCHEN "-" OPERATOR.
4470 A=ASC(T$): V=0: IF T$="" THEN RETURN
4480 V=-1: IF A=35 OR A=41 THEN V=0
4490 IF (A<91)*(A>64)+(A<123)*(A>96)+(A<58)*(A>47) THEN V=0
4500 RETURN
```

Beispiele:

```
ABLEITUNG VON X↑2 NACH X :
2*X

ABLEITUNG VON X↑3 NACH X :
3*X↑2

ABLEITUNG VON X↑Y NACH X :
Y*X↑(Y-1)

ABLEITUNG VON A↑X NACH X :
EXP(X*LN(A))*LN(A)

ABLEITUNG VON (X↑2)↑(3*X) NACH X :
EXP(3*X*LN(X↑2))*(3*LN(X↑2)+3*X*1/X↑2*2*X)

ABLEITUNG VON SIN(X↑2)↑2 NACH X :
2*SIN(X↑2)*COS(X↑2)*2*X

ABLEITUNG VON SQR(X-COS(X)) NACH X :
1/(2*SQR(X-COS(X)))*(1+SIN(X))
```

Hans-Georg Lange, 1000 Berlin 30

Symbolisches Differenzieren (LISP)

Bekanntlich gibt es eine ganze Reihe von Verfahren zur numerischen Differentiation. Hier soll nun eine Methode zur 'symbolischen' Differentiation vorgestellt werden, deren Implementierung sich einige Eigenschaften der Sprache LISP zunutze macht. Wir gehen von folgenden grundlegenden Differentiationsregeln aus:

$$\frac{dX}{dX} = 1$$

$$\frac{dY}{dX} = 0 \qquad X \neq Y$$

$$\frac{d(X \pm Y)}{dX} = \frac{dY}{dX} \pm \frac{dX}{dX}$$

$$\frac{d\ (U*V)}{dX} = U\ \frac{dV}{dX} + \frac{dU}{dX}\ V$$

$$\frac{d(1/X)}{dX} = -\ (1/X^2)$$

Zusammengesetzte Funktionen, z.B. Polynome, lassen sich durch wiederholtes Anwenden dieser Regeln differenzieren: Man zerlege die Funktion in einzelne Terme, die den o.g. Regeln genügen, wende diese an und konstruiere die Ableitung aus den Teilergebnissen. Damit läßt sich in LISP ein Funktional DIFF formulieren, welches ausgiebig Gebrauch von sich selbst macht:

```
(DEFINEQ DIFF
  (LAMBDA (E X)
    (COND ((ATOM E)
           (COND ((EQ E X)
                  (QUOTE 1))
                 (T (QUOTE 0))))
          ((EQ (CAR E) (QUOTE +))
           (LIST (QUOTE +) (DIFF (CADR E) X) (DIFF (CADDR E) X)))
          ((EQ (CAR E) (QUOTE -))
           (LIST (QUOTE -) (DIFF (CADR E) X)))
          ((EQ (CAR E) (QUOTE 1/))
           (LIST (QUOTE -) (LIST (QUOTE *) (LIST (QUOTE 1/) (LIST (QUOTE *)
  (CADR E) (CADR E))) (DIFF (CADR E) X))))
          ((EQ (CAR E) (QUOTE *))
           (LIST (QUOTE +) (LIST (QUOTE *) (CADDR E) (DIFF (CADR E) X)) (LIS
  T (QUOTE *) (CADR E) (DIFF (CADDR E) X)))))))
```

Als Beispiel sei nun die nachfolgende Funktion betrachtet:

$$Y = 2X^3 + 5X^2 + 7X + 2$$

Sie muß in die Form eines S-Ausdruckes in INFIX-Notation gebracht werden. Dazu gibt es ein Programm, hier jedoch nur das Verfahren:

Term 1:	$2X^3$	wird zu	(* (* (* X X) X) 2)
Term 2:	$5X^2$		(* (* X X) 5)
Term 3:	7X		(* 7 X)
Term 4:	2		2

Wir definieren nun die Funktion Y als Summe dieser Terme, bilden die erste Ableitung YSTRICH, die zweite Ableitung Y2STRICH nach X und evaluieren beide für Werte von X, z.B. für 3 und 4.

```
 RESULT OF  Y

(+ (+ (+ (* (* (* X X) X) 2) (* (* X X) 5)) (* 7 X)) 2)


 RESULT OF  (SETQ YSTRICH (DIFF Y (QUOTE X)))

(+ (+ (+ (+ (* 2 (+ (* X (+ (* X 1) (* X 1))) (* (* X X) 1))) (* (* (* X X)
  X) 0)) (+ (* 5 (+ (* X 1) (* X 1))) (* (* X X) 0))) (+ (* X 0) (* 7 1))) 0)


 RESULT OF  (SETQ Y2STRICH (DIFF YSTRICH (QUOTE X)))

(+ (+ (+ (+ (+ (* (+ (* X (+ (* X 1) (* X 1))) (* (* X X) 1)) 0) (* 2 (+ (+
  (* (+ (* X 1) (* X 1)) 1) (* X (+ (+ (* 1 1) (* X 0)) (+ (* 1 1) (* X 0))))
```

Das Ergebnis von Y2STRICH ist hier nur für die ersten Terme ausgedruckt. Man erkennt daran schon den Nachteil dieses Verfahrens: Durch die stur rekursive Anwendung der Differentiationsregeln entstehen Terme der Form (* X 0) oder (+ 1 1). Bei Bildung der 2. Ableitung werden die Regeln wiederum stur auf an sich eindeutige Terme angewandt, im Resultat entsteht weiterer 'Müll'. Es empfiehlt sich also, die S-Ausdrücke zu kürzen. Hierzu sei ein einfaches Schema vorgestellt:

> Sind beide Operanden eines arithmetischen Ausdruckes numerisch, dann ersetze den Ausdruck durch das Ergebnis: SMP1
>
> Ist ein Operand symbolisch, der andere jedoch numerisch und entweder 0 oder 1 dann ersetze: (* X 0) zu 0, (* X 1) zu X und (+ X 0) zu X : SMP2
>
> Zerlege zusammengesetzte Ausdrücke solange, bis sie entweder einer der genannten Regeln genügen oder bis zu Atomen abgebaut sind: SHORT

Ein so gekürzter Ausdruck läßt sich u.U. durch wiederholte Anwendung dieser Regeln weiter vereinfachen (TRY). Abbruchkriterium ist die globale Variable TFLAG, die durch die Prozedur FFLAG gesetzt wird, wenn eine der ersten beiden Regeln angewandt werden konnte. Ein Testlauf mit obigem Beispiel zeigt die annehmbare Effizienz dieses Verfahrens:

```
(DEFINEQ FLAGG
  (LAMBDA (X)
    (PROGN (SETQ TFLAG 1)
           (RETURN X))))

(DEFINEQ SMP1
  (LAMBDA (X)
    (COND ((AND (§ (CAR (CDR X)))
                (§ (CAR (CDR (CDR X)))))
           (FLAGG (EVAL X))))))

(DEFINEQ SMP2
  (LAMBDA (X)
    (COND ((§ (CAR (LAST X)))
           (COND ((EQ (CAR X) (QUOTE *))
                  (COND ((EQ (CAR (LAST X)) 0)
                         (FLAGG 0))
                        ((EQ (CAR (LAST X)) 1)
                         (FLAGG (CAR (CDR X))))))
                 ((EQ (CAR X) (QUOTE +))
                  (COND ((EQ (CAR (LAST X)) 0)
                         (FLAGG (CAR (CDR X))))))))))
```

```
(DEFINEQ SHORT
  (LAMBDA (FIRST)
    (COND ((EQ FIRST)
           NIL)
          ((ATOM FIRST)
           FIRST)
          ((SMP1 FIRST)
          )
          ((SMP2 FIRST)
          )
          (T (CONS (SHORT (CAR FIRST)) (SHORT (CDR FIRST)))))))

(DEFINEQ TRY
  (LAMBDA (X)
    (PROGN (SETQ TFLAG 0)
           (SETQ X (SHORT X))
           (COND ((>TFLAG 0)
                  (TRY X))
                 (T X)))))
```

RESULT OF YSTRICH

```
(+ (+ (+ (+ (* 2 (+ (* X (+ (* X 1) (* X 1))) (* (* X X) 1))) (* (* (* X X)
  X) 0)) (+ (* 5 (+ (* X 1) (* X 1))) (* (* X X) 0))) (+ (* X 0) (* 7 1))) 0)
```

RESULT OF (SETQ YSTRICH (TRY YSTRICH))

```
(+ (+ (* 2 (+ (* X (+ X X)) (* X X))) (* 5 (+ X X))) 7)
```

RESULT OF (SETQ Y2STRICH (TRY (DIFF YSTRICH (QUOTE X))))

```
(+ (+ 0 (* 2 (+ (+ (+ X X) (* X 2)) (+ X X)))) 10)
```

RESULT OF (SETQ X 3)

```
3
```

RESULT OF (EVAL YSTRICH)

```
91
```

RESULT OF (EVAL Y2STRICH)

```
46
```

RESULT OF (SETQ X 4)

```
4
```

RESULT OF (EVAL YSTRICH)

```
143
```

RESULT OF (EVAL Y2STRICH)

```
58
```

RESULT OF ENDE

```
NIL
```

Vorstehendes Programm wurde unter LISP auf CBM entwickelt (Implementierung des Autors).

Dipl.-Ing. Wolfgang Seer, 1000 Berlin 26

Der UNASS, ein erster Test

In Heft 28 des MICRO MAG stellte Herr Kirchgäßner den UNASS vor. Der nachstehende Artikel kann als die logische Fortsetzung betrachtet werden. Er hat das Ziel,

1. die praktische Anwendbarkeit des UNASS zu demonstrieren,
2. die Bereitschaft für den Einsatz zu erhöhen und
3. die Grenzen für den individuellen Einsatz des UNASS besser zu erkennen.

Gleichsam als Testfall soll hier über die (aller)erste Erfahrung des Autors mit dem ausgezeichneten Programm UNASS berichtet werden. Es wurde auf die im Heft 23 des 65xx MICRO MAG beschriebene ISAM-Routine angesetzt, die 2 KBytes umfaßt. Es war dabei das Ziel,

1. die Lösung des ISAM-Problemes im einzelnen studieren zu können,
2. den bei der Assemblierung entstehenden Objectcode durch Adreßvorgabe in jeden Speicherblock des CBM ablegen zu können,
3. den Objectcode für jedes der vier Betriebssysteme des CBM anpassen zu können und
4. den Objectcode für jede DOS-Version der CBM-Floppy durch Einsatz des interaktiven MAE anpassen zu können.

Zum letzten Punkt sei gesagt, daß die zu untersuchende Routine auf das Inhaltsverzeichnis der Diskette direkt zugreift, das je nach Floppy auf Spur 18 oder 39 liegen kann.

Wenn man einmal die allgemeinen Voraussetzungen für den Einsatz des UNASS geschaffen hat, läßt sich von den genannten Punkten der dritte mühelos verwirklichen. In den anderen Fällen müssen entweder die Gedanken des Entwicklers der Routine nachvollzogen werden oder zumindest nach der 'Methode des scharfen Ansehens' diejenigen Teile, die vom UNASS nicht erkannt werden, einer individuellen Nachbehandlung unterzogen werden.

Doch zurück zu den erwähnten allgemeinen Voraussetzungen: Etwas mühsam war das Eingeben der Label-Namen mit den Hex-Adressen nach den ROM-Routinen von H.-J. Koch. Es kann nur wärmstens empfohlen werden, diese als fertige Diskette zu beziehen (I. M. Koch, Hessenring 79, 6367 Karben). – Der UNASS konnte nach der Erstellung der Label-Files als BASIC Programm geladen werden. Die ISAM-Routine befand sich bereits im üblichen Bereich von hex A000-AFFF. Mit dem Disassembler des NEWTIM wurden dort zwei kleine Tabellen erkannt und notiert. Wegen der optimalen Größe von 1 KByte für jedes UNASS-Modul wurden folgende Bereiche gewählt: Modul 1: A010-A426 und Modul 2: A427-A7ED. Die ISAM-Buffer als 'working storages' lagen bei A800-AFFF, also hinter der Routine, und brauchten damit nicht berücksichtigt zu werden.

Nun konnte der UNASS mit RUN gestartet werden. Die Eingabe der Parameter für Programmnamen, Anzahl der Module, Start- und Endadressen sowie für das Betriebssystem bereitete keinerlei Schwierigkeiten. Die ROM-Labels 3001 wurden relativ schnell eingelesen und das Control File geschrieben. Danach dauerte der Zusammenbau der Labeltabelle im PASS1 mit etwa 30 Min. recht lange.

Im PASS2 werden nacheinander die Module gebildet und auf die Diskette zurückgeschrieben (Zeitdauer je Modul etwa 12 Min.). Wem das zulange dauert, der sollte den UNASS compilieren. Beim ersten Versuch empfiehlt es sich jedoch, die BASIC-Version zu verwenden. Zum Schluß schreibt der UNASS das Programlabel-File.

Ein Blick in das Inhaltsverzeichnis der Diskette zeigt die erwarteten Files (Bild 1). Etwas gespannt wird der Editor des MAE geladen und das Control-File zur Inspektion aufgerufen. Es enthält für die Neu-Assemblierung alle erforderlichen Files (Bild 2).

Ein weiterer Blick in das File mit den Labels der Routine (ISAM.LAB) zeigt, daß alle Labels vom UNASS als extern (.DE) deklariert werden und durch Vorsetzen des Buchstabens Z vor die Adresse gebildet werden (Bild 3a). Die Labels Z4D und Z5620 werden mit dem FIND des Editors schnell

gefunden und als Text entlarvt und im Label-File gleich gelöscht. Das Augenmerk gilt hier den Labels des Betriebssystems ZE0F9, ZF164. Ersterer erweist sich als ein Schein-Label: Eine Subroutine hat zwei Entry Points mit je einem Ladeparameter, die mit dem (Dummy-)BIT Befehl entkoppelt sind.

Bei E0F9 ist eine Datentabelle im Betriebssystem, aus der die CHARGET-Routine in den RAM-Bereich kopiert wird Das Label ZF164 wurde als Routine für den IEEE-Bus erkannt. – An dieser Stelle könnte man durch Vorgabe eines anderen Label-Files und der normalen Nacharbeit in den vorgenannten Punkten das Quellprogramm für ein anderes Betriebssystem, bei gleicher Lage des Objectcodes, neu assemblieren.

Wie sieht nun das erzeugte Source-Listing aus? Bild 5a zeigt den Beginn des (un)assemblierten Maschinenprogramms. Zunächst besticht die optische Gliederung gegenüber dem einfachen Disassembler-Listing (Bild 5). Man erkannt die von UNASS eingesetzten Pseudo-Labels 'Z' sowie die entsprechenden Label-Namen aus dem Label-File. Wer kann sich nach solcher Vorarbeit dem Zwang des Weiter-Entschlüsselns entziehen? Der Autor jedenfalls nicht.

Die Bilder 5a und 5b zeigen in einer ausschnittsweisen Gegenüberstellung, wie das fertige Source-Listing aussehen kann.

Es soll hier nicht verschwiegen werden, daß bis zum endgültigen Source-Listing eine gehörige Portion Akribie und Ausdauer gehört. Aber durch die im Editor vorhandenen FIND und REPLACE-Funktionen können für erkannte Z-Labels symbolische Namen fast mühelos ersetzt werden und gelegentliche Irrtümer beim nächsten Listing ausgemerzt werden. – Dies nur als Motivation zum Weitermachen, falls jemand auf halber Strecke aufgeben will. Facit: Wunder kann der UNASS nicht vollbringen. Jedoch entlastet er den Programmierer, der z.B. eine Routine entschlüsseln will, weitgehend von zeitraubender, formaler Aufbereitungsarbeit.

Literatur:
65xx MICRO MAG, Heft 23: ISAM, ein Dateityp
65xx MICRO MAG, Heft 28: Vom Code zum Text: UNASS
Koch, H.-J.: Speicherbelegung und ROM-Routinen, Garbsen 1981
West, Raeto C.: Programming the PET/CBM
Assembler-Listing CBM 3032, Commodore, Neu-Isenburg 1979

```
1  [redacted]                    [redacted]
1     "isam.ctl"          prg
22    "isam.m01"          prg
21    "isam.m02"          prg
5     "isam.lab"          prg
615 blocks free.
```

Bild 1: Vom UNASS erzeugte Textfiles

```
]
0000; 'ISAM.CTL'
0010
0020              .CT
0030              .OS
0040              .CE
0050
0060              .FI  "ISAM.LAB"
0070              .FI  "ISAM.M01"
0080              .FI  "ISAM.M02"
0090              .FI  "LABEL.3001"
0100
0110              .EN
//
```

Bild 2: Das Control File

```
0050
0060Z4D       .DE  $4D
0070ZB1       .DE  $B1
0080ZB2       .DE  $B2
0090ZB3       .DE  $B3
0100ZB4       .DE  $B4
0110Z03E2     .DE  $03E2
0120Z05A9     .DE  $05A9
0130Z5620     .DE  $5620
0140ZA000     .DE  $A000
0150ZA084     .DE  $A084
0160ZA800     .DE  $A800
0170ZA802     .DE  $A[illegible]
0180ZA805     .[illegible]
0190ZA806       [illegible]A85
0200ZA9[illegible]  .DE  $AA86
021[illegible]8  .DE  $AA88
[illegible]ZAA89  .DE  $AA89
0640ZAAA0     .DE  $AAA0
0650ZE0F9     .DE  $E0F9
0660ZF164     .DE  $F164
//
```

Bild 3a: Das Prog-Labelfile (vorher)

CMD.TABL

```
., A000  0A          ASL
., A001  64          ???
., A002  55 31       EOR $31,X
., A004  3A          ???
., A005  55 32       EOR $32,X
., A007  3A          ???
., A008  42          ???
., A009  2D 41 3A    AND $3A41
., A00C  42          ???
., A00D  2D 50 3A    AND $3A50
., A010  AD 03 AA    LDA $AA03
., A013  09 60       ORA #$60
., A015  85 D3       STA $D3
., A017  A9 08       LDA #$08
., A019  85 D4       STA $D4
., A01B  20 B6 F0    JSR $F0B6
., A01E  A5 D3       LDA $D3
., A020  4C 64 F1    JMP $F164
```

Bild 5: Beginn der Routine, disassembliert

```
0000; 'ISAM.M01'
0010
0020; A010 TO A426
0030
0040            .BA $A010
0050            .MC $A010
0060
0070ZA010       LDA ZAA03
0080            ORA #$60
0090ZA015       STA *SA
0100            LDA #8
0110            STA *DN
0120            JSR TALK
0130            LDA *SA
0140            JMP ZF164
```

Bild 5a: Beginn der Routine, vom UNASS erzeugt

```
0230;
0240CMD.TABL    .BY $0A $64
0250            .BY 'U1:'           ; #2
0260            .BY 'U2:'           ; #5
0270            .BY 'B-A:'          ; #8
0280            .BY 'B-P:'          ; #12
0290
0300CMD.TALK    LDA USR.FILENR
0310            ORA #$60            ;Daten empfangen
0320CMD.TALK1   STA *SA
0330            LDA #8
0340            STA *DN
0350            JSR TALK
0360            LDA *SA
0370            JMP TKSA
0380
0390CMD.LISTN   LDA USR.FILENR
```

Bild 5b: Beginn der Routine, nach manueller Bearbeitung

Bücher

Schumny, H. (Hrsg.): Taschenrechner + Mikrocomputer Jahrbuch 1983, (4. Ausgabe), Vieweg-Verlag, Braunschweig 1982, 296 S. ISBN 3-528-04208-7, DM 29,80. Die früheren Jahrbücher wurden bereits als nützliche Nachschlagewerke besprochen. Wir finden diesmal eine besondere Ausrichtung des Textteiles auf Taschen- und Tischrechner von HP und TI und weniger eine Besprechung oder Programmvorschläge für Mikrocomputer. Gleichwohl sind die bewährten Übersichten enthalten für Mikroprozessorbausteine, Lieferanten, Informationsquellen, Zeitschriften, Bücher.

Dipl.-Ing. (FH) Rudolf Kirchgäßner, 6831 Brühl

Drei Disk Utilities

INPUT.ASM ist eine BASIC-freundliche Routine, die einen String mit vorgegebener Länge vom IEEE-Bus liest. Vorausgesetzt wird, daß das entsprechende File geöffnet wurde. Der Aufruf erfolgt durch:

SYS (IN) A, B$, C

Dabei sind: A Die logische File-Nummer, B$ die Stringvariable und C die vorgesehene Stringlänge.

Der Vorteil dieser Routine ist neben der Schnelligkeit die Vermeidung der Garbage-Collection. Bei Dateiende oder Fehlern wird bis zur vorgesehenen Länge mit binären Nullen aufgefüllt. Das bei 'Null-Strings' notwendige IF B$ = "" THEN B$ = CHR$(0) kann entfallen. Der String wird direkt im Speicher angelegt, er muß deshalb auch nicht umkopiert werden.

FIRST-FREE gibt den 1. File-Eintrag im Directory der Diskette frei. Dies kann von Interesse sein, weil sich für das erste File im Inhaltsverzeichnis eine vereinfachte Lademöglichkeit ergibt durch das Betätigen der Tasten SHIFT/STOP bzw. durch LOAD "",8.*

Der erste File-Eintrag kommt an die nächstmögliche Stelle, d.h. er wird an das Ende des Inhaltsverzeichnisses angefügt oder ein gelöschter Eintrag wird überschrieben. Sollten alle 8 möglichen Einträge eines Sektors belegt sein, so muß die Zuweisung und Verkettung eines neuen Sektors im Directory durch SAVE und anschließendes SCRATCH eines Dummy-Files gelöst werden. Auf die Zeilen 50 und 60 sei besonders hingewiesen, dort wird der Disk-Typ festgestellt. Es bedeuten:

TD=1 CBM 2040 DOS 1.0
TD=2 CBM 4040 DOS 2.0
TD=3 CBM 8050 DOS 2.5

Die SYS-Routine ist natürlich in diesem wie auch im nächsten Beipiel das oben beschriebene INPUT.ASM, diesmal als Data-Statements.

PRO-MENUE sollte den durch FIRST-FREE freigemachten Platz einnehmen und damit bequem ladbar sein. Es schreibt ein Menue aller auf der Diskette vorhandenen Programme auf den Schirm. Nach Eingabe einer Ziffer wird das entsprechende Programm geladen und gestartet.

Literatur:
Collins, John: Commodore Magazin, Oktober 1981. West, Raeto Collins: Programming the PET/CBM, Level Limited, London 1982.

```
                    0010 ;INPUT.ASM  LIEST STRING VON IEEE-BUS  14.11.82
                    0020
                    0030 ;    AUFRUF : SYS(IN)A,B$,C
                    0040 ;         A : LOGISCHE FILENUMMER
                    0050 ;        B$ : STRINGVARIABLE
                    0060 ;         C : EINZULESENDE BYTEZAHL
                    0070
                    0080
                    0081 ;    ADRESSEN IM KOMMENTAR SIND FUER BASIC 4
                    0082
                    0085              .BA $7FC2
                    0086              .OS
                    0087
                    0088 FAC1         .DE $5E
                    0089 STATUS       .DE $96
                    0090
7FC2- 20 78 D6      0099 IN           JSR $D678             ;$CBD4  ARGBYT
7FC5- 20 C6 FF      0100              JSR $FFC6             ;CHKIN
7FC8- 20 F8 CD      0110              JSR $CDF8             ;$BEF5  KOMMA
7FCB- 20 6D CF      0120              JSR $CF6D             ;$C12B  FVAR
7FCE- 20 90 CC      0130              JSR $CC90             ;$BD89  STRTYP
7FD1- 85 46         0140              STA *$46
7FD3- 84 47         0150              STY *$47
```

```
7FD5- 20 CC D6  0160             JSR $D6CC              ;$C927  KOMMA+GETBYT
7FD8- 8A        0170             TXA                    ;GEWUENSCHTE LAENGE
7FD9- 20 4F D3  0180             JSR $D34F              ;$C59E  PLATZ?
7FDC- A0 00     0190             LDY #0
7FDE- 20 E4 FF  0200 LOOPIN      JSR $FFE4              ;GETIN
7FE1- A6 96     0210             LDX *STATUS
7FE3- D0 12     0220             BNE LOOPIN2
7FE5- 91 5F     0230             STA (FAC1+1),Y
7FE7- C8        0240             INY
7FE8- C4 5E     0250             CPY *FAC1              ;LAENGE
7FEA- D0 F2     0260             BNE LOOPIN
7FEC- 20 A4 D3  0270 ENDIN       JSR $D3A4              ;$C5F3  DESCR > STRINGTAB
7FEF- 20 E2 C8  0280             JSR $C8E2              ;$B965  LET-ROUTINE
7FF2- 4C CC FF  0290             JMP $FFCC              ;CLRCH
7FF5- A9 00     0310 LOOPIN1     LDA #0
7FF7- 91 5F     0360 LOOPIN2     STA (FAC1+1),Y
7FF9- C8        0370             INY
7FFA- C4 5E     0380             CPY *FAC1
7FFC- D0 F7     0390             BNE LOOPIN1
7FFE- F0 EC     0400             BEQ ENDIN
                0405
                0410             .EN
END OF MAE PASS!
```

```
******************                                  **********************
CBM 3032  10.01.83                                  FIRST-FREE  SEITE: 1
******************                                  **********************

  1 ' FIRST-FREE  10.01.83
  3 ' R.KIRCHGAESSNER
  4 ' NIBELUNGENSTR.4
  5 '  D 6831 BRUEHL
  6 '
 10 PRINT"CLR.RVS.FIRST-FREE  VER2DOWN.DOWN.
 20 POKE 53,126:CLR:I=32706:GOSUB 240
 30 INPUT"DRIVE   OLEFT.LEFT.LEFT.";D:IF D<>0 AND D<>1 THEN 30
 40 OPEN 15,8,15:PRINT#15,"I"D:OPEN 1,8,2,"#"
 50 PRINT#15,"M-R"CHR$(255)CHR$(255):GET #15,X$:'  DISKTYP FESTSTELLEN
 60 X=ASC(X$):TD=1:IF X AND 16 THEN TD=3:IF X AND 1 THEN TD=2
 70 X=0:S=1:H=18:IF TD=3 THEN H=39
 80 T=H:PRINT#15,"U1";2;D;T;S:'                     AUF SEKTOR POSITIONIEREN
 90 GET #1,T$:SYS (I)1,S$,1:'                       LINK-BYTES LESEN
100 SYS (I)1,FF$,30:'                               1.FILE-EINTRAG MERKEN
110 X=0:FOR F=1 TO 8:'                              8 FILE-EINTRAEGE LESEN
120 PRINT#15,"B-P";2;32*F-30
130 GET #1,X$:IF X$="" THEN X=F:F=8:GOTO 150:'      EINTRAG FREI
140 IF ASC(X$)<129 THEN X=F:F=8:'                   EINTRAG GELOESCHT
150 NEXT:IF X>0 THEN 190
160 IF T$="" THEN GOSUB 230:GOTO 70:'               SEKTOR VOLL BELEGT
170 T=ASC(T$):S=ASC(S$):PRINT#15,"U1";2;D;T;S:'     NAECHSTER SEKTOR
180 GET #1,T$:SYS (I)1,S$,1:GOTO 110:'              NAECHSTE LINK-BYTES LESEN

190 PRINT#15,"B-P";2;32*X-30:PRINT#1,FF$;:'         1.FILE IN FREIE POSITION
200 PRINT#15,"U2";2;D;T;S:'                         EINTRAGEN UND SCHREIBEN
210 PRINT#15,"U1";2;D;H;1:PRINT#15,"B-P";2;2
215 PRINT#1,CHR$(0);:PRINT#15,"U2";2;D;H;1:'        1.EINTRAG LOESCHEN
220 CLOSE 1:CLOSE 15:PRINT"OK":END

230 SAVESTR$(D)+":DUMMY",8:PRINT#15,"S"D":DUMMY":RETURN:'  SEKTOR-KETTUNG

240 FOR X=I TO I+61:READ S:POKE X,S:NEXT:RETURN:'  SYS(I)A,B$,C
```

```
250 DATA 32,120,214,32,198,255,32,248,205,32,109,207:'    A=LOGISCHE FILE#
260 DATA 32,144,204,133,70,132,71,32,204,214,138,32:'    B$=STRINGVARIABLE
270 DATA 79,211,160,0,32,228,255,166,150,208,18,145:'     C=STRINGLAENGE
280 DATA 95,200,196,94,208,242,32,164,211,32,226,200
290 DATA 76,204,255,169,0,145,95,200,196,94,208,247,240,236

1581 BYTES SPEICHERBEDARF (STAT.)
```

```
*******************
CBM 3032  14.11.82
*******************
```

```
**********************
PROG-MENUE  SEITE: 1
**********************
```

```
  1 ' PROG-MENUE  13.11.82
  3 ' R.KIRCHGAESSNER
  4 ' NIBELUNGENSTR.4
  5 '  D 6831 BRUEHL
  6 '
 10 POKE 53,126:CLR:GOSUB 170:O=32706:GOSUB 180
 20 INPUT"DRIVE   OLEFT.LEFT.LEFT.";D:IF D<>0 AND D<>1 THEN 20
 30 DIM A$(144):OPEN 15,8,15:PRINT#15,"I"D:OPEN 1,8,2,"#"
 40 Y=0:P=0:S=1:T=18:GOSUB 170:'  8050:  T=39
 50 PRINT#15,"U1";2;D;T;S:'        AUF SEKTOR POSITIONIEREN
 60 GET #1,T$:SYS (O)1,S$,1:'      LINK-BYTES
 70 FOR F=1 TO 8:SYS (O)1,X$,1:'   8 EINTRAEGE LESEN
 80 IF ASC(X$)<>130 THEN SYS (O)1,X$,31:GOTO 110:' KEIN PROGRAMM
 90 SYS (O)1,X$,2:Y=Y+1:SYS (O)1,A$(Y),16:SYS (O)1,X$,13
 95 IF P=2 THEN P=0:'              8032: IF P=4...
100 PRINT TAB(20*P)"RVS."RIGHT$(STR$(Y),2)"OFF. "A$(Y)" ";:P=P+1
110 NEXT:B=B+1:IF T$="" OR B=5 THEN GOSUB 130:GOSUB 170:IF T$="" THEN 40
115 '          8032: ...OR B=9 ...
120 T=ASC(T$):S=ASC(S$):GOTO 50:' NAECHSTER SEKTOR

130 PRINT:PRINT TAB(20)"RVS.IHRE WAHL? >=0 ";
140 INPUT B:IF B<1 OR B>Y THEN RETURN
150 PRINT"CLR.LOAD"CHR$(34)STR$(D)":"A$(B)CHR$(34)",8DOWN.DOWN.DOWN.DOWN.DOWN
    .RUN"
160 POKE 623,19:POKE 624,13:POKE 625,13:POKE 158,3:END

170 B=0:PRINT"CLR."TAB(12)"RVS.PROG-MENUEDOWN.":RETURN

180 FOR X=0 TO O+61:READ S:POKE X,S:NEXT:RETURN

190 DATA 32,120,214,32,198,255,32,248,205,32,109,207
200 DATA 32,144,204,133,70,132,71,32,204,214,138,32
210 DATA 79,211,160,0,32,228,255,166,150,208,18,145
220 DATA 95,200,196,94,208,242,32,164,211,32,226,200
230 DATA 76,204,255,169,0,145,95,200,196,94,208,247,240,236

1120 BYTES SPEICHERBEDARF (STAT.)
```

Dipl.-Ing. Uwe Kornnagel, 6096 Raunheim

Multi-TASK bei Mikrozessoren

Dieser Beitrag soll aufzeigen, daß auch die heutigen 8-Bit Mikrocomputer in der Lage sind, Probleme wie das Multiprogramming zu lösen. Die Beispiele mögen mögliche Lösungen anregen.

Multi-TASK beschreibt das quasi gleichzeitige Ausführen von unabhängigen Programmen auf der gleichen Hardware. Mit ihm lassen sich z.B. das Mitlaufen einer Uhr, automatisches Datensichern oder die Ausgabe über Spooling lösen. Multi-TASK wird hier mittels maskierbarem Interrupt vorgestellt. Durch eine Hardware (Timer) wird alle 20 ms ein Interrupt erzeugt, der die Kontrolle an den TASK-Prozessor übergibt. Dazu auch das nachfolgende Struktogramm.

Der TASK-Prozessor überstreicht bei jedem Interrupt alle aktiven Tasks (die Aktivität ist im Flag markiert) und dekrementiert bei ihnen den jeweiligen CONTER. Wenn nun ein Counter zu Null geworden ist, dann soll die Task ausgeführt werden. Zunächst wird der Counter aus dem Speicherwort RCL-TIME neu vorgeladen. Ein niedriger Wert für diese Vorgabe RCL-TIME bedeutet also, daß die Task häufig abgearbeitet wird und damit Priorität hat. Dann wird Bit 2 im Flag zu 0 gesetzt (Task in Arbeit), und es wird das Programm, dessen Anfügeadresse in ADRESSE steht, ausgeführt. Ist das Programm beendet, so wird dieses Bit 2 im Flag wieder auf 1 gesetzt.

RCL-TIME		COUNTER		NAME						ADRESSE		FLAG	
													Ø
													1
													2
													3
													4
													5
													6
													7
													8
													9
Ø	1	2	3	4	5	6	7	8	9	10	11	12	

Bild 1 : TASK-Tabelle

RCL-TIME: Enthält einen Wert zwischen 1 und FFFF. Er gibt den Reziprokwert an, mit welcher relativen Frequenz ein Programm angesprungen wird.

COUNTER: Er dient als Zähler für den Ansprung der Routine. Wird sein Wert zu 0, so wird er aus RCL-TIME neu geladen und das Programm wird ausgeführt.

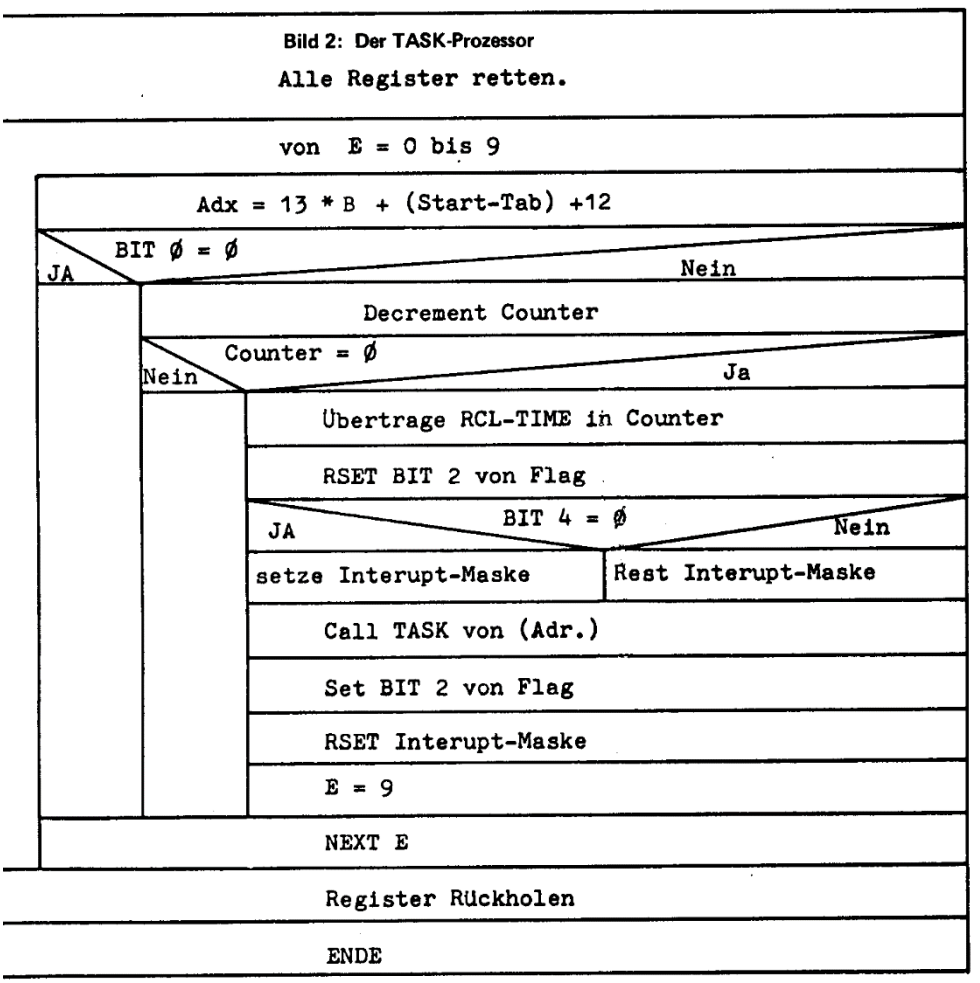
TASK-Name: Enthält den Namen des Programmes.

ADRESSE enthält die physikalische Startadresse des Programmes, beim 6502 die Startadresse–1.

FLAG: Gibt den Task-Status an. Die Bitpositionen haben folgende Bedeutung

Bit 0	= 0 Task nicht vorhanden	= 1 Task vorhanden
Bit 1	= 0 Task aktiv	= 1 Task inaktiv
Bit 2	= 0 Task in Arbeit	= 1 Task frei
Bit 3	= 0 automatischer Task	= 1 Einmal-Task
Bit 4	= 0 höchste Priorität	= 1 normale Aktivität
Bit 5 bis 7	frei für Anwender	

Bild 2: Der TASK-Prozessor



```
TASK-PROCESSOR FUER 6502
0000            0001          *=0
0000            0002 PNTL     *=*+2             ;POINTER IN ZERO PAGE

0002            0004 ;INTERRUPTROUTINE FUER 10 TASKS (# 0-9)
0002            0005          *=$200            ;IRQ VEKTOR MUSS
0200            0006 START                       ;AUF START ZEIGEN
0200 48         0007          PHA               ;SAVE 3 REGISTER
0201 8A         0008          TXA               ;GGFS. VON HIER AN PHXY
0202 48         0009          PHA
0203 98         0010          TYA               ;GGFS. FUER R65C02 PHY
0204 48         0011          PHA               ;Y AUF DEN STACK
0205            0012
0205 A991       0013          LDA #<TASKTB      ;TABELLENANFANG
0207 A002       0014          LDY #>TASKTB
0209 8500       0015          STA PNTL          ;POINTER INITIALISIEREN
020B 8401       0016          STY PNTL+1
020D A200       0017          LDX #0            ;ZAEHLER VON 0 AN
020F            0018
020F A00C       0019 LOOP     LDY #12           ;PRUEFE OB TASK AKTIV
0211 B100       0020          LDA (PNTL),Y      ;FLAG-BYTE
0213 2C8F02     0021          BIT EINS          ;ALS BIT 'IMMEDIATE'
0216 F003       0022          BEQ ENDT1         ;DE-AKTIVIERT
0218 203202     0023          JSR TASKO         ;TASK PRUEFEN GGFS. BEARB
021B 18         0024 ENDT1    CLC
021C A500       0025          LDA PNTL          ;AUF NAECHSTE TASK
021E 690C       0026          ADC #12
0220 8500       0027          STA PNTL
0222 A501       0028          LDA PNTL+1
0224 6900       0029          ADC #0            ;HI BYTE ; CARRY
0226 8501       0030          STA PNTL+1
0228 E8         0031          INX               ;ALLE TASKS UEBERSTRICHEN
0229 E00A       0032          CPX #10           ;ZAHL+1 ?
022B D0E2       0033          BNE LOOP          ;NEIN, WEITER PRUEFEN
022D            0034
022D 68         0035          PLA               ;REGISTER WIEDER ABLIEFERN
022E A8         0036          TAY
022F 68         0037          PLA TAX
0230 68         0038          PLA
0231 40         0039          RTI               ;ENDE INTERRUPT-ROUTINE
0232            0040
0232 48         0041 TASKO    PHA               ;REGISTER FUER SUBROUTINE
0233 98         0042          TYA
0234 48         0043          PHA
0235 8A         0044          TXA               ;RETTEN
0236 48         0045          PHA
0237 A002       0046          LDY #2            ;TASK IST AKTIV
0239 38         0047          SEC               ;JETZT IHREN COUNTER -1
023A B100       0048          LDA (PNTL),Y
023C E901       0049          SBC #1            ;LO BYTE
023E 9100       0050          STA (PNTL),Y
0240 C8         0051          INY               ;FUER HI BYTE
0241 B100       0052          LDA (PNTL),Y
0243 E900       0053          SBC #0            ;GGF. BORGEN
0245 9100       0054          STA (PNTL),Y
```

```
0247 88       0055          DEY
0248 1100     0056          ORA (PNTL),Y     ;BEIDE BYTES ODERN
024A D02C     0057          BNE ENDTO        ;COUNTER NOCH NICHT 0
024C          0058
024C A000     0059          LDY #0           ;RCL-TIME NACH COUNTER
024E 207C02   0060          JSR SETO         ;UEBERTAGEN
0251 A001     0061          LDY #1
0253 207C02   0062          JSR SETO         ;2. BYTE
0256 A00C     0063          LDY #12          ;ADRESSIERE DAS FLAG
0258 B100     0064          LDA (PNTL),Y
025A 29FD     0065          AND #$FD         ;TASK IN ARBEIT SETZEN
025C 9100     0066          STA (PNTL),Y
025E 58       0067          CLI              ;NORMAL MIT INTERRUPT
025F 2C8E02   0068          BIT ZEHN         ;HOECHSTE PRIORITAET?
0262 D001     0069          BNE *+3          ;NEIN, NORMALE TASK
0264 78       0070          SEI              ;INTERRUPTE AUSSCHLIESSEN
0265 208302   0071          JSR TASK1        ;TASK AUSFUEHREN
0268 A00C     0072          LDY #12          ;FLAG ADRESSIEREN
026A B100     0073          LDA (PNTL),Y     ;HOLEN
026C 0904     0074          ORA #4           ;SETZE 'TASK FREI'
026E 9100     0075          STA (PNTL),Y
0270 68       0076          PLA              ;ALTES X ENTFERNEN
0271 A909     0077          LDA #9           ;SIMULIERE ENDE VON 'LOOP'
0273 AA       0078 ENDT2    TAX              ;NACH X ALS ZAEHLER
0274 68       0079          PLA              ;Y UND A ZURUECK
0275 A8       0080          TAY
0276 68       0081          PLA
0277 60       0082          RTS
0278          0083
0278 68       0084 ENDTO    PLA              ;ALTES X
0279 4C7302   0085          JMP ENDT2
027C          0086
027C B100     0087 SETO     LDA (PNTL),Y     ;UEBERTRAGE RCL NACH COUNT
027E C8       0088          INY
027F C8       0089          INY              ;2 ZELLEN WEITER
0280 9100     0090          STA (PNTL),Y
0282 60       0091          RTS
0283          0092
0283 A00B     0093 TASK1    LDY #11          ;HOLE ADRESSE DER TASK
0285 B100     0094          LDA (PNTL),Y
0287 48       0095          PHA              ;AUF DEN RETURN-STACK
0288 88       0096          DEY              ;NUN LO BYTE
0289 B100     0097          LDA (PNTL),Y
028B 48       0098          PHA
028C 60       0099          RTS              ;TASK ANSPRINGEN
028D          0100
028D 01       0101 EINS     .BYT 1
028E 10       0102 ZEHN     .BYT $10
028F          0103
028F          0104 TASKTB   *=*+108          ;TABELLE NACH BILD 1
02FB          0105          .END

  ERRORS= 0000
```

Michael Zimmermann, 6102 Pfungstadt

Produktbesprechung: EDTASM+

Color-Computer Editor Assembler mit ZBUG

Einleitung

Der TRS-8- Color-Computer wurde als eine auf dem MC 6809 basierende Maschine bereits vom Verfasser in Heft 23 dieser Zeitschrift vorgestellt. Eine Schwachstelle bei diesem System war, daß zunächst kein Monitor für die Arbeit in Maschinensprache zur Verfügung stand. Diese Lücke konnte jetzt durch Direkteinkauf des EDTASM+ bei Tandy in den USA geschlossen werden. Es handelt sich um ein ROM-Pack, das den Color-Computer um einen Editor, einen Assembler und einen ZBUG genannten Monitor erweitert. Diese drei Softwareprodukte sollen hier kurz vorgestellt werden, weil der Color-Computer nach Preissenkungen eine kostengünstige Möglichkeit darstellt, sich mit dem 6809 näher zu befassen.

Der Editor

Ein Editor ist Voraussetzung für das Abfassen von Quelltexten. Bei eingelegtem ROM-Pack meldet er sich mit einem Stern als Prompt. Er ist zeilenorientiert, die Zeilen sind numeriert und können nach Zeilenummern aufgeschlagen werden. Mit den Sonderzeichen Zahlenkreuz, Stern und Punkt werden die erste, die letzte und die laufende Zeile aufgeschlagen, und es können auch Zeilenbereiche angezeigt werden. Der Editor hat folgende Kommandos:

- *Laden, Speichern und Verifizieren von Textfiles*
- *Einfügen und Ersetzen von Zeilen und Zeilenbereichen*
- *Kopieren, Übertragen und Löschen von Zeilen und Bereichen*
- *Aufsuchen von Zahlenstrings*
- *Neu-Numerieren*
- *Ausgabe von Zeilen und Bereichen auf Bildschirm und Drucker*
- *Aufrufen von BASIC und ZBUG*
- *Aufschlagen einer Zeile zur zeichenweisen Editierung, Möglichkeiten zum Einfügen, Löschen und Ändern von Zeichen*

Alle diese Befehle, ebenso wie die später zu beschreibenden Kommandos des ZBUG-Monitors, sind Eintasten-Befehle, ergänzt evtl. um Angaben zur angesprochenen Zeile etc.. In jedem Fall sind sie zur Ausführung mit der Enter-Taste abzuschließen.

Der Editor bietet sich auch an, um BASIC-Programme zu erstellen. Das Bandformat ist zu dem des Assemblers kompatibel.

Der Assembler

Ebenfalls unter Steuerung des Editors wird der Assembler angesprochen. In der Grundeinstellung erlaubt er die Assemblierung eines im Textpuffer enthaltenen Quellenprogrammes auf ein Magnetbandfile. Die Symboltafel wird dabei direkt im Anschluß an den Textpuffer aufgebaut. Es können aber auch andere Optionen gewählt werden. Die wichtigste ist sicher die Direktablage des Objektprogrammes im RAM-Speicher, womit es sofort ausführbar wird. Weitere Optionen betreffen die Startadresse des Maschinenprogrammes und den Umfang und das Ziel des Ausdruckes.

Während z.B. der Assembler des AIM ein freies Format des Quelltextes zuläßt, ist für den hier besprochenen Assembler eine feldweise Eingabe erforderlich: Zeilen die nicht mit einem Label beginnen, müssen mit zumindest einer Leerstelle beginnen. Für eine bessere Lesbarkeit ist in jedem Falle eine Tabulation zu empfehlen, die auch auf dem Bildschirm Label, Operation und Operand sauber einteilt. - Die Opcodes entsprechen in ihrer Darstellung dem Motorola-Standard, dies gilt ebenso für die Pseudo-Operationen und Operanden.

Der ZBUG-Monitor

Der Monitor wird vom Editor her aufgerufen und meldet sich mit einem #-Prompt. Er stellt dem Benutzer eine Vielzahl von Befehlen zur Verfügung. Man kann Maschinenprogramme von der Cassette laden, sie dorthin abspeichern und verifizieren. Weitere Befehle betreffen die Ausgabe des Speicherinhaltes. Die Anzeige ist zum einen schrittweise vor- und rückwärts möglich, zum anderen blockweise, in letzterem Modus auch auf dem Drucker. Man kann Bytes und Worte als Zahl ausgeben, Bytes als ASCII-Zeichen, und man kann den Speicherinhalt disassemblieren. Bei der schrittweisen Anzeige kann der Speicherinhalt auch geändert werden, im mnemonischen Mode ist freilich keine symbolische Eingabe möglich.

Statt mit der Vor- oder Folgeadresse kann die Anzeige auch mit jener Adresse fortgesetzt werden, die den Operand des gerade disassemblierten Befehles oder den Inhalt der angezeigten Adresse darstellt. Für alle ein- und auszugebenden Zahlen ist die Basis wählbar (oktal, dezimal, hexadezimal). Eine Überschreibung der einmal getroffenen Zuordnung ist bei einzelnen Zahlen durch Prefix oder Suffix möglich. Wenn das Programm direkt assembliert wurde, so befindet sich die Symboltabelle noch im Speicher. Man kann dann die Adressen einmal mit ihrem Wert oder mit ihrer symbolischen Bezeichnung ansprechen. Die Anzeige der Adressen kann demzufolge in 3 Formen erfolgen:

- *numerisch* — *Speicheradresse und Operand rein numerisch*
- *symbolisch* — *Speicheradresse und Operand rein symbolisch*
- *halb symbolisch* — *Speicheradresse als Symbol, Operand numerisch*

Die hier angesprochene symbolische Form kann nicht nur für die Anzeige des Speicherinhaltes, sondern ebenfalls für die Adressierung von Breakpoints oder für das Starten von Programmen an bestimmten Adressen benutzt werden. Dadurch wird ein echtes symbolisches Debugging möglich, der Benutzer braucht sich im Prinzip nicht einmal mehr um die Speicherzuordnung zu kümmern Insgesamt sind bis zu 8 Breakpoints möglich. Darüber hinaus kann ein Programm schrittweise ausgeführt werden.

Hinzu kommen Befehle zur Anzeige und Veränderung von Registerinhalten und zur Verschiebung von Speicherblöcken. Auch ein Rechner für 16-Bit Intergerzahlen ist implementiert. In ihm sind die vier Grundrechenarten, Modulus, die logischen Operationen, Shift und Komplement enthalten. Mit diesen Grundoperationen können durchaus komplexe Ausdrücke formuliert werden, Klammern können verwendet werden.

Die Dokumentation

Mit dem EDTASM+ wird ein 70-seitiges englisches Handbuch geliefert. Dieses erläutert die Benutzung von Editor, Assembler und ZBUG ausreichend und mit einer Reihe von Beispielen. Es gibt natürlich keine Einführung in die Assemblersprache, hierzu wird auf entsprechende Radio Shack-Publikationen verwiesen. Für die Arbeit mit dem EDASM+ selber sind die Angaben vollkommen ausreichend. Die Fehlermeldungen, ihre Ursachen und deren Behebung sind hinreichend beschrieben.. Was man hier noch mehr als bei den BASIC-Handbüchern vermißt, das sind nähere Angaben zu den Systemadressen. Lediglich die allernotwendigsten Unterprogramme der Ein- und Ausgabe sind dokumentiert. Es bleibt zu hoffen, daß Tandy hierzu bald genauere Informationen liefert.

Bewertung

Wie jedes andere Softwareprodukt, so erfordert auch dieses eine gewisse Eingewöhnung in Möglichkeiten und Darstellungsformen. Nach dieser Lernphase zeigt sich der EDTASM als ein in sich konsistentes und ausgewogenes Produkt. Die Optionen von Editor und Assembler sind umfangreich und gehen über das z.B. beim AIM gebotene hinaus. Die Fähigkeiten des ZBUG-Monitors übertreffen alles, was dem Verfasser von anderen Microcomputer-Monitoren her bekannt ist.Insbesondere ist hier die Möglichkeit des symbolischen Debug hervorzuheben. Insgesamt ist der EDTASM ein ausgewogenes und vernünftiges System für eine ernsthafte maschinenorientierte Arbeit. Er sollte den Color-Computer als preiswertes System auf der Grundlage des MC6809 für viele Leser dieser Zeitschrift interessant machen.

Peter W. Arps, Brockdorffstraße 5, 2000 Hamburg 73

Einkommensteuerberechnung 1982

Der nachfolgende Programmausschnitt bringt das in Heft 19 (Juni 1981) und Heft 21 veröffentlichte Programm auf den neuesten Stand. Es kann in der aktuellen Version auf CBM-Tonbandcassette beim Autor bezogen werden, und zwar gegen Voreinsendung von DM 22,; auf Postscheckkonto Hamburg 3719 49-208. Die jetzt berücksichtigten wesentlichen Änderungen sind der Wegfall des erm. Steuersatzes für Nebeneinkünfte aus schriftstellerischer und wissenschaftlicher Tätigkeit und der sog. Progressionsvorbehalt bei Arbeitslosengeld/-hilfe, Kurzarbeitsgeld und Schlechtwettergeld, die bei der Berechnung der Steuer mit berücksichtigt werden.

```
110 DIM  AL(1),VB(1),WK(1),FB(5),KV(3),VV(1),LR(1),LF(1),GW(1) : REM
    VZ'82
115 DIM  AE(3),PV(3),GB(1),X(18) : REM VZ'82
520 FOR I=0 TO 1: Y=0: X=AL(I): IF VB(I)<=0 THEN 540: REM VZ'82
525 Y=VB(I)*P0+.9: Y=INT(Y): IF Y>4800 THEN Y=4800: REM VZ'82
530 X=X-Y: IF X<0 THEN X=0: REM VZ'82
540 X=X-600-480-564: IF X<0 THEN IF PV(I)>0 THEN PV(I)=PV(I)-(-X): REM
    VZ'82
545 IF X<0 THEN X=0: REM VZ'82
550 IF WK(I)>564 THEN X=X-(WK(I)-564): REM VZ'82
760 Y=2340-2340*(SK=3)+600*(KD+KU/2): SY=Y: REM VZ'82 (KU BIS VZ'85)
850 DEF  FNY(Y)=-(Y*(Y=<56400)+56400*(Y>56400)): REM VZ'82
920 Z=-(3000*(SK<3)+6000*(SK=3))-X: IF Z<0 THEN Z=0: REM VZ'82
1160 IF Y<270 THEN Y=270: REM VZ'82
1170 IF SK=3 THEN IF Y<540 THEN Y=540: REM VZ'82
1495 IF Z<0 THEN Z=0: REM VZ'82
1500 PRINT "Einkuenfte aus BAFOEG des Kindes";: GOSUB 2170: Z=Z+X: REM
    VZ'82
1780 PRINT "↓↓Verlust-Vortrag/-Ruecktrag 1977-1983": GOSUB 2160:
    REM VZ'82
1782 Y=5000000: X=-(X=<Y)*X-(X>Y)*Y: REM VZ'82
1784 X=-(X=<ZE)*X-(X>ZE)*ZE: ZE=ZE-X: REM VZ'82
1850 REM VZ'82 DELETE
1860 IF SK=2 THEN Y=4212: REM VZ'82
2030 PX=PV(0)+PV(1): IF PX<0 THEN PX=0: REM VZ'82
2040 PX=PX+PV(2)+PV(3): IF PX=0 THEN 2090: REM VZ'82
2050 Y=ZE: ZE=ZE+PX: GOSUB 2210: IF SK=3 THEN ZX=ZX*2: REM VZ'82
2060 PX=ZX: PS=INT(SS/ZX*100000): PS=PS/1000: REM VZ'82
2070 ZE=Y: GOSUB 2210: IF SK=3 THEN ZX=ZX*2: REM VZ'82
2080 SS=ZX*PS/100: SS=INT(SS): GOTO 2100: REM VZ'82
2430 REM VZ'82 DELETE
2440 REM VZ'82 DELETE
2721 PRINT "clr↓Progressionsvorbehalt fuer": REM VZ'82
2722 PRINT "↓Arbeitslosengeld, Kurzarbeitergeld,": REM VZ'82
2723 PRINT "Schlechtwettergeld u.Arbeitslosenhilfe.": REM VZ'82
2724 PRINT "Hoehe";: GOSUB 2170: PV(I)=X: REM VZ'82
2725 PRINT "↓Ausl.Einkuenfte gem $32b (1)2 EStG": REM VZ'82
2726 PRINT "Hoehe";: GOSUB 2170: PV(I+2)=X: REM VZ'82
2740 INPUT "2.1.1918 geboren (j=ja/n=nein)  n←←←←←";X$: IF X$="n"
    THEN 2770: REM VZ'82
2770 REM VZ'82 DELETE
2780 REM VZ'82 DELETE
2790 REM VZ'82 DELETE
2800 REM VZ'82 DELETE
2810 REM VZ'82 DELETE
3372 IF PX=0 THEN 3380: REM VZ'82
3374 PRINT#4,T$;"nach einem Steuersatz von";STR$(PS);" v.H.": REM VZ'82
3376 PRINT#4,T$;"berechnet auf";STR$(PX): REM VZ'82
```

Bücher

Floegel, E.: FORTH Handbuch, Hofacker Verlag, Holzkirchen 1982, 189 S. ISBN 3-911 682-88-6, DM 49,--. Mit dem Untertitel 'Grundlagen, Einführung, Beispiele dürfte es sich um das erste deutsche FORTH-Buch handeln. Wir finden folgende Einteilung der Kapitel: 1. Was ist FORTH, 2. Sprachelemente, 3. Programmschleifen und Verzweigungen, 4. Zahlensysteme, 5. Textfeld (Screen), 6. Wörterbuch, 7. Texteingabe, 8. Beispiele auf Heim-Computern, 9. Programmbeispiele, Anhang mit Befehlswort-Übersicht. — Dieses Buch wird dem gestellten Anspruch gerecht, nämlich eine Einführung und Übersicht besonders für den Anfänger in dieser Sprache zu geben. Es werden alle wesentlichen Eigenschaften und Leistungsmerkmale der Sprache in einer Form dargestellt, die man leicht mitvollziehen kann. Beim Studium des Buches sollte man natürlich auch das vorwiegend besprochene FIG-FORTH auf seinem Computer unter den Fingern haben. Zur Didaktik in der Stoffanordnung mag man unterschiedliche Auffassungen haben, es sollte aber besonders für diese Sprache in einer späteren Auflage eine 'Cross Reference' im Buch enthalten sein, die zeigt, auf welcher Seite die jeweiligen Befehle vorgeführt oder in einer besonderen Art benutzt worden sind.

Kane, G., Hawkins, D. und Leventhal, L.: 68000 Assembly Language Programming, Verlag Osborne/MacGraw-Hill, Berkely 1981, ca. 540 S., ISBN 0-931 988-62-4, ca. DM 56,-. Im Konzept und in den Beispielen folgt das Buch dem Aufbau ähnlicher Leventhal-Bücher. Durch das Hinzutreten des früheren Motorola-Mannes Hawkins und von Kane, der das Buch '68000 Microprocessor Handbook' schrieb, ist ein Lehrbuch entstanden, das sehr gut die spezifischen (auch alternativen) Programmiermöglichkeiten darstellt, die der 68000 bietet. Anhand vieler kleinerer grundlegender Aufgaben wird mit Nennung der Voraussetzungen, Flußdiagrammen, Assembler-Listen der eingeschlagene Lösungsweg aufgezeigt. Im Anschluß daran folgt eine Würdigung der jeweils benutzten neuen oder besonderen Befehle und ihrer Auswirkungen. Zusammen mit einem Computer sollte damit die Einarbeitung zuverlässig möglich sein, vor allem wenn man schon etwas Übung mit irgendeinem Assembler hat.

Kruttschnitt, F. und Maier, W.: Löten in der Elektrotechnik und Elektronik, Verlag Markt & Technik, Haar 1982, 144 S. kart., ISBN 3-922120-17-2, DM 24,--. Das Buch ist eine Zusammenfassung der Kriterien der modernen Löttechnik und gibt eine Übersicht über die Vorgänge während des Lötprozesses. Schwerpunktmäßig wird das Handlöten abgehandelt und hierzu die Werkzeuge und Geräte, die die optimalen Wirkungen erzielen, insbesondere die temperaturgeregelten Löt- und Entlötgeräte. Der Anhang ist den Metallen gewidmet: Leichtmetalle, Eisen, Nickel, Blei und Zinn, Kupfer, Edelmetalle. Mit seinen übersichtlichten Tabellen und dem ausgebreiteten Erfahrungsschatz wird das Buch dem Praktiker viele nützliche Hinweise geben.

Alt, H. (Hrsg.): Programmierung mathematischer Algorithmen, Vieweg Programmbibliothek, Taschenrechner 1, Vieweg-Verlag, Braunschweig 1982, 92 S., ISBN 3-528-04211-7, DM 19,80. Es handelt sich um eine sauber aufgemachte Programmsammlung vorwiegend für die Taschenrechner TI 59 und HP 41C sowie HP 12C mit folgenden Themen: Berechnung von Fakultäten, Extrapolationsverfahren, Hypergeometrische Verteilung, Exponentielle Wachstumsbeschreibung, Allgemeines Iterationsverfahren.

Bretschneider, Th., Planen und kalkulieren mit VISICALC auf Apple II-Computern, Verlag Markt & Technik, Haar 1982, 133 S. kart. ISBN 9-922120-19-9, DM 29,80. Dieses Buch soll dem Anwender den Umgang mit dem leistungsfähigen Berechnungsprogramm VisiCalc erleichtern. Es führt Schritt für Schritt in die vielfältigen Möglichkeiten ein. Von einem einfachen Modell ausgehend führt das Buch bis hin zu besonderen Kniffen. Beispiele aus verschiedenen Anwendungsbereichen geben Anregungen für eigene Einsatzmöglichkeiten. Ein umfangreiches Stichwortverzeichnis am Schluß mit Seitenverweisen erleichtern das Zurechntfinden.

FIG-Chapter

Ein norddeutsches FIG-Chapter soll am Sa., den 26.2.83 gegründet werden. Treffpunkt BLIMP, Müggenkampstr. 63, Hamburg 19 (U2 bis Lutterothstr.), 14 Uhr. Danach regelm. Treffen am 4. Samstag im Monat. Kontakt: FIG, c/o Klaus Schleisiek, Postfach 20 22 64 in 2000 Hamburg 20, Tel. 040 - 480 81 54.

Editorial

Mit der Titelseite 'Machine of the Year – The Computer Moves In' wählte die Zeitschrift TIME in Ausgabe 1/83 den Computer und seinen Einzug in die Betriebe und Haushalte als 'Ereignis des Jahres mit historischer Tragweite'. Die Titelstory füllt mit weiteren Berichten die ersten 23 Seiten.

Viele Leser schreiben angesichts der stürmischen Weiterentwicklung im dem Sinne '... braucht man auch in Zukunft das Engagement eines so vorzüglichen Micro-Forums, wie es das 65xx MICRO MAG ist'. Solches Lob wird, weil es die Bemühungen bestätigt, gelegentlich gerne entgegengenommen, es ist natürlich in erster Linie an die Autoren weiterzuleiten.

Mit 'Micro-Forum' ist wohl das Entscheidende angesprochen: Es hat in den nun bald fünf Jahren viele Formen des Gedankenaustausches gegeben, nicht nur durch die Veröffentlichungen in der Zeitschrift, sondern auch in zahllosen Telefonaten, Beratungen und persönlichen Begegnungen mit den Lesern und der Leser untereinander, die den erreichten gemeinsamen Informationsstand befördert haben. Viele der angesprochenen Grundlösungen wurden im Laufe der Zeit von anderen Lesern aufgegriffen und mit verbesserter Leistungsfähigkeit wieder vorgestellt, was sehr zu begrüssen war. Zur Zeit erleben wir bereits auch eine lebhafte Addition der Erfahrungen und Vorschläge für den Gebrauch der Sprache FORTH, die in der Leserschaft wie es scheint eine sehr gute Aufnahme findet. – Neue Gegebenheiten in der Programmierung oder in der Hardware werden heute viel schneller aufgegriffen als noch vor wenigen Jahren.

'Call for Papers': Bei der Vielzahl der Aufgaben, für die der Computer bereits eingesetzt wird, schlummern ganz sicher zahlreiche Vorschläge und Programme noch unerkannt oder unbekannt in der Leserschaft. Soweit sie oder ihr Prinzip von breiterem Interesse sein könnten, sind die Leser ausdrücklich ermuntert, gelegentlich auch aus ihren Arbeitsgebieten zu schreiben, damit das 'Forum' ein möglichst breites Spektrum abdeckt und sich letztlich für den Autor auch wieder fördernt auswirkt. Im Zweifel stimme man sich im Vorwege kurz mit dem Herausgeber ab.

MC 68000, 16 Bit: Bei Abschluß dieses Heftes traf die CPU-Platine SYS68K/CPU-1 (mit VME-Bus-Interface) der FORCE-Computers beim Herausgeber ein. Sie wird in einigen Tagen in Betrieb genommen, besprochen und betreut werden. Damit beginnt für diese Zeitschrift (in der Architektur aufwärtskompatibel) die 16-Bit-Zeit. Aus Gesprächen hört man heraus, daß zahlreiche Leser dieses System oder ein ähnliches bestellt haben oder bestellen wollen. Man sollte so früh wie möglich den Gedankenaustausch untereinander aufnehmen, der Herausgeber ist gern Mittler. #

REMON 3.2: Verbessertes Monitorprogramm für AIM 65, z.B. Erweiterung von Editor- u. Monitorbefehlen, Tastenrepeat, Kleinschreibung, deutscher Zeichensatz u. vieles mehr. Software in 2 2332-EPROMs für DM 88,-- einfach in die Steckplätze Exxx und Fxxx auf dem AIM einsetzen. Bestellung bei Dipl.-Ing. R. Wollenberg, Stockumer Str. 234, 4600 Dortmund 50, T.0231/753477.

AIM 65, Assembler, BASIC, FORTH mit 16k RAM, SK-Busverstärkung, Videointerface VIC I und Netzteile, Gehäuse von MSB abzugeben. Preis VHS. Weinstein, Tel. (0721) 68 53 66.

AIM 65, 4K-RAM, Assembler- und BASIC-ROM, Schroff-Stromversorung, Literatur, DM 1000,-. Dr. P. Winde, Donnerburgweg 10, 3300 Braunschweig, Tel.: 0531 - 32 15 20.

AIM 65 mit 4k RAM (Zweitgerät) 900,-- DM. H. J. Regge, 2800 Bremen , Fesenfeld 57.

Schnelle EPROM-Löscher

Für **3 EPROM** mit Batterie oder ext. Netzgerät betrieben DM 100,-. Für **6 EPROM** mit Schaltuhr, Schublade und Netzbetrieb DM 180,-. Preise inkl. MWSt/ohne Versandkosten

Real Time-Debugger

Box für 8 bit Prozessoren (6502, 6802, 6809, weitere auf Anfrage) zum Debuggen Ihrer Ass-Programme. Trigger auf Adresse oder auf Adresse+Daten. Verschiedene Modes wie: Fetch, Trap, Single Step und Read Register.

Entwicklungsbüro F. Krickl, Schauinslandweg 27 7730 VS-Schwenningen, Tel.: 07720/ 61 233.

Video: 2 Fernseh-SW-Geräte, 31 cm, m. zus. Eingang für Video-Signale preiswert zu verkaufen. **Zeitschriften-Jahrgänge FUNKSCHAU 1965-81, Elektronik 76-81, Elektor 74-77,** jeweils bis 77 o. Reklame geb., abzugeben. R. Löhr, Teil.: 04 102 - 55 816.

Neu

Neu

CPU-Karte NIKO 89

Die CPU-Karte NICO 89 ist
eine vollständig in sich geschlossene Rechnerkarte.

Als CPU sind wahlweise 6502 oder 6809
je nach Art der Bestückung möglich.

Auf der Karte befinden sich weiterhin

der Adreßdekoder
drei freischwingende Oszillatoren (MC14584)
dre Fassungen wahlweise für RAM oder EPROM
eine VIA 6522
eine PIA 6821
eine ACIA 6850
Counter 6840

Alle I/O-Anschlüsse sind auf eine VG 64 Steckerleiste geführt.
Der Rechnerbus ist nicht ausgeführt.
Die RAMs sind für Batteriepufferung
durch die auf der Karte mögliche Batterie vorgesehen.

Der restliche Platz auf der CPU-Karte wird
durch ein Lochrasterfeld belegt.
Dadurch können eigene Anpassungsschaltungen
auf der Karte realisiert werden.

Stromversorgung: 5 V, ca. 600 mA, je nach Bestückung.

Preis: je nach Bestückung 280,-- bis 340,-- zzgl. MWSt.

ELEKTRONISCHE BAUGRUPPEN
MICROPROZESSOR-TECHNIK
Ohmann

Im Wiehagen 15 - Tel. (05241) 67480
4830 Gütersloh 12

Assembler unter FORTH

CROSS--09

Cross-Assembler für MC6809 (Motorola). Tonbandcassette (6,5K), Diskette (CBM 3040/4040) oder 2 EPROMs, deren Inhalt nach hex 200-1B00 kopiert werden muß. DM 380,-- + MWSt.

CROSS--05

Cross-Assembler für MC6805/68705xx (Motorola), auf Cassette, Diskette oder EPROM für D000: DM 320,-- + MWSt.

Alle Assembler sind komfortabel ausgelegt, benutzen die Mnemonics des Herstellers und laufen unter FORTH des AIM 65 (für CBM-FORTH von Lowinski oder PHS ggfs. auf Anfrage). Sie erzeugen EPROM-fähigen Code für beliebige Speicherräume und enthalten alle üblichen Kontrollstrukturen wie BEGIN, WHILE, REPEAT, IF, ELSE, THEN usw. aber auch alle Branches (6809: Longbranches). Es kann mit externen Symbolen und Labels im Programmablauf gearbeitet werden. Zusätzlich sind viele Assembler-Anweisungen implementiert: Byte- Word- und Stringablage, .OPTLIS, Reserve Memory Bytes und binäre Argumente.

Mathe- ROM für 6502

Implementierung von P. Rix (s. Textteil) für Sockel D000. Fließkommaarithmetik und höhere mathem. Funktionen (wie in Microsoft-BASIC) für AIM 65-FORTH(komfortabler eigener Fließkommastapel) und für jedes 6502-Assemblerprogramm (dokumentierte Einsprungspunkte und Argumente). EPROM lieferbar ab Jan. 1983. DM 110,- + MWSt.

R65C02: Für den erweiterten Befehlssatz der CMOS-CPU ist ein echter 2-Pass-Assembler in Vorbereitung, nachdem es sich als unzweckmäßig erwiesen hat, den 6502-Assembler unter FORTH des AIM entsprechend zu erweitern. Lieferbar ab etwa März 1983. – Eine Vorbestellung ist möglich. Sie erhalten dann bei der Fertigstellung zunächst ein genaueres Angebot.

Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg, T. 04102 - 55 816.

Kleinanzeigen

32K-Bytes CMOS Speicherplatine

mit Pufferbatterie, Adresse 1000-9FFF für AIM 65 und PC 100, wird auf dem Rechnerboard angeschlossen, so daß der Expansionsstecker frei bleibt und die Platine im jeweiligen Rechnergehäuse untergebracht ist. Die Platine kann auch mit EPROMs vom Typ 2716 bestückt werden.

Preis DM 945,00

Komplett bestückt mit TMM 5517AP, Anschlußstecker und Kabel, Info gegen Rückporto. Jessica Stecker, Gustav Cods Str. 7, 5000 Köln 60

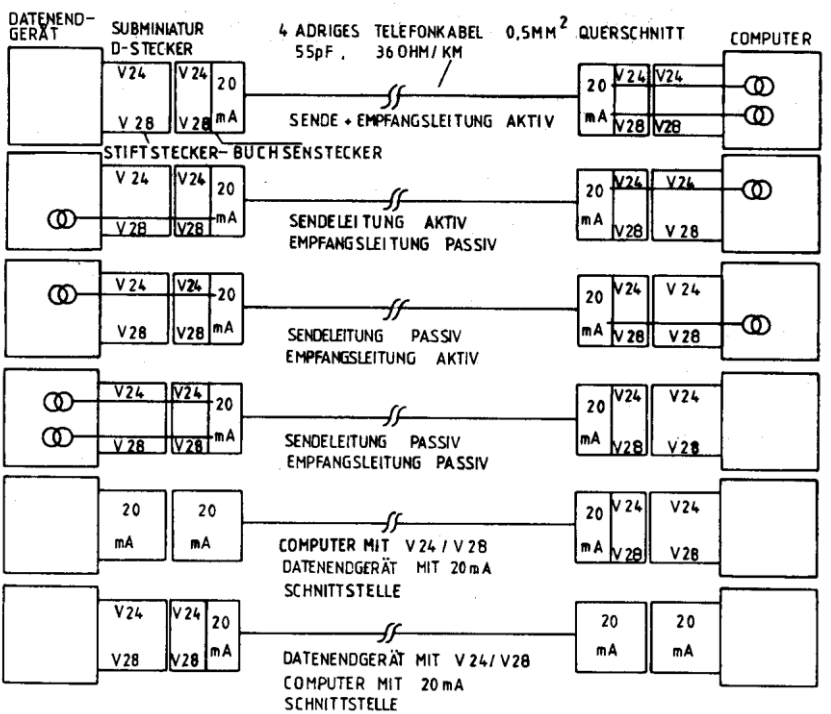
"Starting FORTH" von Leo Brodie, die beste und umfassendste Einführung in FORTH für DM 52,80. Dr. J. Schrenk, Postf. 904, Karlsruhe 41.

Kleinanzeigen kosten DM 10,-- für 2 Zeilen. Betrag bitte bei Auftragserteilung überweisen.

Die Stecker

von Stecker haben es in sich!

Die Wandlung der V.24/V.28 Spannungspegel in einen 20 mA Strompegel erfolgt durch eine Schaltung, die in die serienmäßigen Griffschalen der 25-poligen Subminiatur D-Stecker von AMP, Amphenol, Harting usw. eingebaut wird. Sie haben keine lästigen externen Geräte mehr herumstehen und müssen sich auch nicht mehr um deren Spannungsversorgung kümmern.



Der Stecker

von Stecker wird einfach in die V.24 Buchsenleiste eingesteckt und schon können Sie mit der Datenübertragung bis zu

5 km

-je nach Widerstand und Kapazität der Leitung und der Baudrate- ihre Daten zwischen Rechner und Datenendgerät übertragen. Die Spannungsversorgung für

die Stecker

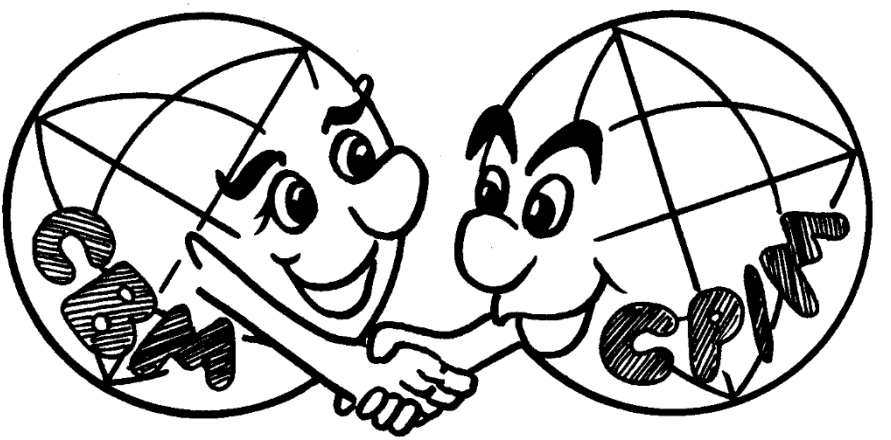
von Stecker erfolgt mit plus und minus 12 Volt über die in der DIN 66020 nicht genormten Steckkontakte 9 und 10. Beide Spannungen werden nach CCITT Empfehlung für V.24 und V.28 typischerweise in Ihren Datenendgeräten und Computern bereitgehalten.
Alle Leitungen sind über Optokoppler galvanisch getrennt, so daß Einstreuungen über die Leitung nicht zur Zerstörung Ihrer Geräte und Anlagen führt. Die Datenübertragungsgeschwindigkeit beträgt max. 20 Kilobits/sec im vollduplex Mode.
Der Preis: 339,00/Stück incl. Umsatzsteuer, 1 Jahr Garantie.

STECKER
INGENIEURBÜRO

Postfach 60 07 66 5000 Köln 60 Tel.: (0221) 7 12 40 18

Neue Telefon-Nummer!!

DATA BECKER
hat die Lösung:



Zwei Software-Welten endlich miteinander verbunden!

CP/M und Commodore Software auf einem Rechner

Welcher Commodore Besitzer hat sich nicht schon einmal gewünscht, auch auf CP/M, das meistverbreitete Betriebssystem der Welt, zurückgreifen zu können und damit gleichzeitig auf die größte Softwarebibliothek der Welt. Auf über 100 verschiedenen Mikrocomputern ist CP/M installiert. Die Zahl der vorhandenen Programme geht in die Tausende. Darunter sind so bekannte wie das Textverarbeitungsprogramm WORDSTAR, die Kalkulationsprogramme SUPERCALC und MULTIPLAN, der T/Maker, die Sprachcompiler BASIC, COBOL, FORTRAN und PASCAL von MICROSOFT und viele mehr. Jedes Programm, das unter CP/M erstellt wurde, ist auch auf allen anderen Rechnern unter CP/M lauffähig. Ein unschätzbarer Vorteil, der Software portabel und Software-Investitionen rentabel macht.

Auch Commodore-Besitzer können jetzt die Vorteile der CP/M-Welt nutzen, ohne allerdings die Vorteile der Commodore-Welt aufgeben zu müssen: mit dem CP/M Maker. Mit dem CP/Maker verwandeln Sie Ihr System im Handumdrehn in einen vollwertigen CP/M Rechner mit Z80 Prozessor und 64 K RAM. Selbstverständlich können Sie weiter auch im Commodore-Modus arbeiten. Der CP/Maker stellt Ihnen hier sogar 64 K Speichererweiterung zur Verfügung. Weitere Ausbaumöglichkeiten des CP/Maker sind eine einfache V24 Schnittstelle zur direkten Kommunikation mit anderen CP/M Rechnern und ein schneller Arithmetikprozessor.

Damit macht der CP/Maker aus den Commodore-Computern 4000 und 8000 leistungsfähige Universalrechner, mit denen bei erhöhtem Leistungsvermögen die Vorteile zweier großer Software-Welten auf einem System genutzt werden können. Kennen Sie noch einen Computer, der das kann?

So erfahren Sie mehr: Einfach bei DATA BECKER unter dem Stichwort „CP/Maker Info" komplette kostenlose Informationen über CP/Maker und unsere große Auswahl lieferbarer CP/M Software im Commodore-Diskettenformat anfordern . . .

Wenn Sie zwei Welten miteinander verbinden wollen, kostet das nicht die Welt

CP/Maker nur DM 1998,-*

* CP/Maker incl. Betriebssystem CP/M 2.2 DM 1768,14 zzgl. MwSt. — DM 1998,- incl. MwSt.
erhältlich für die CBM-Serien 4000 und 8000 (bitte bei Bestellung angeben).

CP/Maker Importeur DATA BECKER GmbH · Merowingerstr. 30 · 4000 Düsseldorf · Tel. 0211/312085

Sonderangebot für MICRO MAG-Leser

Lieferung solange Vorrat reicht

Matrix-Drucker C. ITOH 8510A statt DM 1950,- **DM 1695,-**

120 Zeichen/Sek.
203 mm Druckbreite; 40-136 Zeichen/Zeile (6 Stufen)
7x9 Dot Matrix; nx9 Proportionaldruck
Unterstreichen, Unterlängen, hervorgehobener Druck
Proportionalschrift
Vollgrafikfähig und Blockgrafik
ASCII 96 und nationale Zeichensätze US/GB/D/S
griechisch/mathematisch und freiladbarer Zeichensatz
Papierverarbeitung: Einzelblatt-Friktion und
Endlos-Friktion und Traktor
3 KB Datenpuffer
Schnittstelle: Centronix parallel (V24 gegen Aufpreis)

Matrixdrucker C. ITOH 1550 stattt DM 2650,- **DM 2395,-**

wie Modell 8510 A jedoch
345 mm Druckbreite; 68-230 Zeichen/Zeile (6 Stufen)

Daten Monitor CD 12 G statt DM 598,- **DM 449,-**

12'', grün, 22 MHz Auflösung, Stahlblechgehäuse
BAS-Videoeingang, Industrieausführung

Daten Monitor CD 12L statt DM 698,- **DM 498,-**

wie CD 12 G, jedoch orange

Antireflexätzung für CD 12L und G Aufpreis **DM 48,-**

Typenrad Schreibmaschine-Drucker **Preis auf Anfrage**

Schönschriftdrucker, deutscher Zeichensatz
bis zu 8 KB Datenbuffer
19 verschiedene Schriften, Sonderzeichensätze
Schnittstellen V24/RS232, Centronix parallel, IEEE 488

Angebot freibleibend, gültig bis 15.03.83. Alle Preise inkl. MWSt.

Dipl. Ing. (FH) Bernhard Heckl
Alte Wallensteinstraße 146
8500 Nürnberg 80
Telefon 0911/65 17 47

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

Herausgeber:
Dipl.-Volkswirt Roland Löhr
Hansdorfer Straße 4
D–2070 Ahrensburg
Tel.: 04 102 - 55 816

65xx MICRO MAG erscheint zweimonatlich, jeweils Mitte Februar, April usw.. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber. - Offsetdruck: Druckartist Gerhard M. Meier, Hamburg 70.

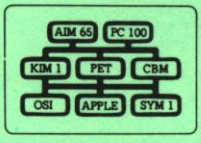
Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- (Inlandsendpreis). Ausland/foreign via surface mail DM 59,--, USA air 26 Dollar. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu zwei Wochen vor deren Ablauf.

Nachliefermöglichkeiten: Hefte 1-13 sollten als Buch nachbezogen werden (s. Anzeige unten). Solange Vorrat reicht, können auch noch einzelne Hefte der Nos. 7-13 geliefert werden. Hefte 14-28 sind unbeschränkt nachlieferbar zu DM 7,80/Stück + DM 2,50 je Sendung.

Private Besteller werden um Überweisung/Scheck (auch Auslandsschecks) zusammen mit der Bestellung gebeten. Konto Roland Löhr, Nr. 654 70-202 Postscheckamt Hamburg, BLZ 200 100 20.

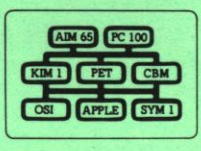
Leser-Service des Herausgebers

Das Buch 1-6 des 65.. MICRO MAG



230 Seiten, DM 26,--

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Thermokopf (Printerplatte für AIM 65 und PC 100
mit Anleitung für den leichten Einbau. Auffrischung des Druckes DM 28,--

FORTH User's Guide
Rockwell-Handbuch für das FIG-FORTH des AIM 65. Mit der Erläuterung des Befehlssatzes auch für andere FORTH-Betreiber geeignet. Ca. 300 S., engl. DM 24,--

Mathe-ROM nach P. Rix für FORTH des AIM + Assemblerprog. m. Doku DM 124,30

Vorstehende Preise inkl. MWSt, zuzüglich DM 2,50/Sendung + ggfs. NN + DM 2,--

Cross-Assembler für Motorola 6809 DM 380,-- und für 6805 DM 320,-- + MWSt,
inkl. Dokumentation (DM 10,--), beide unter FORTH des AIM laufend.

2-Pass-Assembler für R65C02 (erw. Befehlssatz) noch in Vorbereitung.