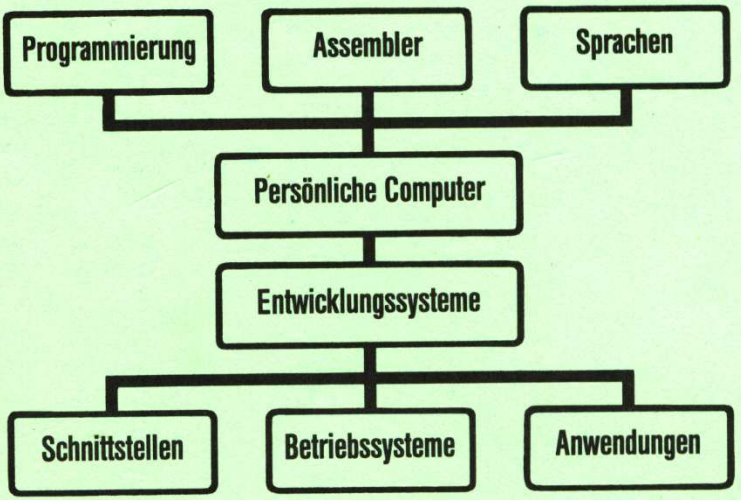
MICRO MAG

DM 9,50 Nr. 48 Januar 1987



Inhaltsverzeichnis

ATARI ST

ASSEMBLER-PRAXIS AUF ATARI ST

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

Erklärt Grundlagen:

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

Zeigt Anwendungen:

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

Entwickelt Hilfsprogramme:

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

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

ca. 300 Seiten, Softcover, DM 59,–

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

Roland Löhr

Strukturen

C und Assembler
Wer sich heute für die Software eines der aktuellen 16-Bit Computer interessiert, stellt schnell fest, daß die Systemprogrammierung weitgehend in der Sprache C erfolgte. Und das gilt auch für viele Anwenderprogramme. Als Computerbetreiber wird man sich jetzt mit C befassen müssen, wenn man schon ein solches System hat, ziemlich sicher aber doch in einiger Zeit, wenn man ein Nachfolgesystem beschafft. In Heft 43 war bereits ein Artikel mit dem Titel ´Literatur zu C´ enthalten, auf den wir rückverweisen. Inzwischen sind einige weitere gute Bücher erschienen, deren Besprechung man in diesem Heft findet. So besteht kein Anlaß, hier eine Artikelreihe zur Einführung in C zu beginnen. Gleichwohl ist es interessant, sich hier einmal vorwiegend für den 65xxx mit C-ähnlichen Strukturen und verbundenen Listen zu beschäftigen. Einerseits findet man für Computer mit solcher CPU nur wenige Implementierungen der Sprache, andererseits sind mit dem Thema Programmiertechniken verbunden, die man vorteilhaft auf vorhandenen und künftigen Computern unter Assembler benutzen kann.

C ist eine Compilersprache. Aus den editierten Quelltexten wird in mehreren Durchgängen ausführbarer Maschinencode erzeugt. Es gibt dabei Parallelen zur Obersetzung eines Assemblerprogrammes. Für die Phase der Obersetzung kennen wir in Assembler die Erklärungen (Deklarationen) z.B. POINTER=55. Im zweiten Assemblerlauf können Speicherzellen weiterhin initialisiert d.h. vorbesetzt werden, z.B. *=POINTER und danach .WOR ZEIGER. In diesem Falle wird in den Speicherzellen des Pointers die Adresse eines anderen Objektes Zeiger abgelegt. Davon muß man gedanklich die dritte Phase unterscheiden: Wenn das Programm zur Ausführung gelangt, kann es mit Befehlen wie LDA (POINTER),Y auf den Inhalt eines ggfs. per Assembler vorbesetzten Pointers zurückgreifen.

In den Assemblern zum 68000 gibt es ähnliche Möglichkeiten. Oft unterscheidet man zwischen den drei Programmsegmenten TEXT, DATA und BSS. TEXT nimmt dabei den erzeugten Maschinencode auf. In DATA findet man die initialisierten Daten (Konstanten), die typisch mit Assembleranweisungen wie POINTER DC.L ZEIGER erzeugt werden: Lege an der symbolischen Adresse POINTER ein Langwort ab, das ade Adresse von ZEIGER enthält. - Das Segment BSS schließlich enthält die nicht initialisierten Variablen, den Arbeitsspeicher zur Laufzeit.

Auch in C haben wir, nicht nur für Strukturen, die Erklärungen an den Compiler, die wahlfreie Vorbesetzung im Compilerlauf und schließlich das Arbeiten mit den Objekten zur Laufzeit. Strukturen sind dabei Datensätzen oder Records in anderen Sprachen vergleichbar, sie enthalten nach einem vom Programmierer bestimmten Schema der Anordnung Felder, in denen Daten durchaus verschiedenen Typs enthalten sein sollen, so z.B. Zeiger auf die nächste Struktur des gleichen Typs, Zeiger auf Text und andere Daten sowie in die Struktur unmittelbar aufgenommene Konstanten oder Variablen. Dazu ein einfaches Beispiel:

```
struct student (
     char *nachname;
     int  matrikelnummer;
     char fach;
)
```

Diese Strukturerklärung gilt für den Typ student. Sie enthält drei Felder, nämlich den Zeiger auf eine Textkette nachname, eine Ganzzahl mit der Matrikelnummer und schließlich ein Byte, in das die Fachrichtung geschrieben wird. Mit dieser Erklärung wird noch kein Speicherplatz reserviert oder belegt. Die Reservierung von Speicher geschieht erst mit einer Anweisung wie:

```
struct student dozent, vorlesung[500];
```

Hier wird Platz für einen Datensatz mit Namen dozent und für ein Array von 500 Datensätzen mit Namen vorlesung[i] reserviert, wobei i eine Laufvariable ist. Alle Datensätze sollen dabei die Struktur student haben. Wenn wir einmal von der möglichen Vorbesetzung einer Struktur im Zuge der Compilierung absehen, so werden im Programmverlauf einzelne Felder als Teilmenge ihrer Struktur angesprochen: dozent.fach="m" weist dem Feld fach in der Struktur dozent den Wert "f" zu, entsprechend erfolgt eine Zuweisung an die i-te Struktur mit z.B. vorlesung i .fach="j". Der Bestandteil dozent. (mit Punkt als Trenner) bezeichnet also den Namen einer bestimmten Struktur. Was dahinter folgt, ist der Name ihres Feldes.

Strukturen im Assembler

Auch in Assembler können wir Strukturen erklären, sie initialisieren und mit ihnen arbeiten. In den Zeilen 6 und 15 der nebenstehenden Liste wird der Zuordnungszähler des Assemblers mit *=0 definiert auf Null gebracht, damit die nachfolgenden Label einen relativen Abstand zum Beginn der beiden Strukturen struct1 und struct2 haben und nicht einen zufälligen Wert annehmen. Die Label struct1l und struct2l am Ende der Strukturen nehmen damit automatisch den Längenwert der Strukturen auf. Die Zeilen bis 21 sind damit reine Strukturerklärungen, die wir weiterbenutzen können.

In den Zeilen 35...43 nehmen wir eine Speicherreservierung vor. Dabei wird beim Weitersetzen des Zuordnungszählers die jeweilige Länge der Strukturen benutzt, wie z.B. in *=*+struct2l. Wir sind übrigens nicht gezwungen, Sätze gleicher Struktur unbedingt zusammenhängend aufzureihen.

In den Zeilen 50...101 wird nun eine Initialisierung der Strukturen per Assembler vorgenommen. Wenn man zur Liste auch die Symboltafel betrachtet, dann sieht man sehr schön, wie in satz1...satz3 die Zeiger auf Texte und Adressen des gleichartigen Folgesatzes abgelegt werden. Es ist üblich, einen Zeiger im letzten Glied einer Kette mit 00 zu füllen, zum Zeichen, daß nichts mehr folgt. Man geht dabei von der Voraussetzung aus, daß an einer Adresse 00 normalerweise keine nachfolgende Struktur stehen kann.

In den Zeilen 74...85 werden die Datensätze record1...3 vom Typ struct2 mit Zeigern auf Nachfolger und Vorgänger gefüllt. Es entsteht eine lineare Liste. In den Zeilen ab 93 wird alternativ eine zirkulare Liste der Strukturen aufgebaut: In ersten Datensatz wird im Feld Vorgänger auf den letzten Datensatz verwiesen. Der letzte Datensatz hat dementsprechend den ersten als Nachfolger.

Vom Assembler gefüllte Strukturen

Solche vom Assembler erzeugte und auch gefüllte Strukturen wird man benutzen, wenn man feststehende Zuordnungen benötigt, z.B. in Sprachübersetzern, Disassemblern, zur Ausgabe von Fehlermeldungen mit nachfolgender gezielter Aktion/Weiterbehandlung. Die folgenden Beispiele gehen zunächst darauf ein, wie man solche statischen Strukturen in Programmen anspricht. Im weiteren Verlauf soll dann auf dynamische Bedingungen eingegangen werden, wo neue Strukturen im Programmlauf entstehen. - Bei den bisherigen Beispielen ist es nicht unbedingt nötig, daß

man die Art der Struktur in die Struktur selbst hineinschreibt. Es kann hilfreich sein, wenn gleiche Strukturen nicht streng hintereinander abspeichert sind. Und man kann jeder Struktur natürlich auch einen Namen geben, der entweder in einem Feld der Struktur steht (wenn eine maximale Länge nicht überschritten wird) oder auf dessen Textkette ein Pointer in der Struktur weist.

```
0002 beispiele fuer c-aehnliche strukturen und verbundene listen
0003 r. loehr, 12/86
0004 */
0005
0006 000000                    *=0               alle label relativ zu struct1
0007 000000          struct1                     ;erklaerung einer ersten stru
                                                 ktur
0008 000000          item1_0   *=*+1             merker strukturart
0009 000001          item1_1   *=*+2             zeiger auf einen nachfolger
0010 000003          item1_2   *=*+2             zeiger auf einen string
0011 000005          item1_3   *=*+2             nutzinformation1
0012 000007          item1_4   *=*+6             nutzinformation2
0013 00000d          struct11                    marke am ende der struktur=la
                                                 enge
0014
0015 00000d                    *=0               alle label relativ zu struct2
0016 000000          struct2                     ;erklaerung einer zweiten str
                                                 uktur
0017 000000          item2_0   *=*+1             merker strukturart
0018 000001          item2_1   *=*+2             zeiger auf einen nachfolger
0019 000003          item2_2   *=*+2             zeiger auf einen vorgaenger
0020 000005          item2_3   *=*+4             nutzinformation
0021 000009          struct21                    marke am ende der struktur
0022
0023 000009                    *=$1000           start irgendeines programmes
0024 001000 ea                 nop
0025 001001 00                 brk
0026
0027                           ;einrichtung eines textbereiches per assembler
0028
0029 001002                    *=$2000
0030 002000          textber   *=*               label fuer textbereich
0031 002000 48494552 text1     .byt 'hier ist text1',0
0031 00200e 00
0032 00200f 48494552 text2     .byt 'hier ist text2',0
0032 00201d 00
0033 00201e 48494552 text3     .byt 'hier ist text3',0
0033 00202c 00
0034
0035 00202d                    *=$3000           beginn des variablenbereiches
0036 003000          structber *=*               bereich der strukturen
0037 003000          satz1     *=*+struct11      reservierung fuer 1 datensatz
                                                  typ1
0038 00300d          satz2     *=*+struct11      dito
0039 00301a          satz3     *=*+struct11      dito
0040
0041 003027          record1   *=*+struct21      reservierung fuer 1 datensatz
                                                  typ2
0042 003030          record2   *=*+struct21      dito
0043 003039          record3   *=*+struct21      dito
0044
0045 003042          structend *=*               ende des benutzten bereiches
0046
```

```
0047                            ;intitialisierung der variablen per assembler
0048                            ;zunaechst die zeiger auf nachfolger setzen
0049
0050 003042                     *=satz1+item1_0     zeiger auf das erste feld sat
                                                    z1
0051 003000 01                  .byt 1              alle datensaetze von struct1
                                                    erhalten die 1
0052 003001 0d30                .wor satz2          zeige auf beginn von satz2
0053                            ;hier koennte sofort die initialisierung der fo
0053                            lgefelder stehen
0054
0055 003003                     *=satz2+item1_0     zeiger auf das erste feld sat
                                                    z2
0056 00300d 01                  .byt 1              alle datensaetze von struct1
                                                    erhalten die 1
0057 00300e 1a30                .wor satz3          zeige auf beginn von satz2
0058 003010                     *=satz3+item1_0     zeiger auf das erste feld sat
                                                    z3
0059 00301a 01                  .byt 1              alle datensaetze von struct1
                                                    erhalten die 1
0060 00301b 0000                .wor 0              mit null wird das ende der ke
                                                    tte bezeichnet
0061
0062                            ;jetzt die textzeiger setzen
0063 00301d                     *=satz1+item1_2     feld fuer den zeiger auf stri
                                                    ngs, satz1
0064 003003 0020                .wor text1          pointeradresse ablegen
0065 003005                     *=satz2+item1_2     feld fuer den zeiger auf stri
                                                    ngs, satz2
0066 003010 0f20                .wor text2          pointeradresse ablegen
0067 003012                     *=satz3+item1_2     feld fuer den zeiger auf stri
                                                    ngs, satz3
0068 00301d 1e20                .wor text3          pointeradresse ablegen
0069
0070                            ;die nutzfelder itemx_3 und itemx_4 analog ausf
0070                            uellen
0071
0072                            ;die felder von record1...record3 mit zeigern
0073                            ;auf nachfolger und vorgaenger besetzen
0074 00301f                     *=record1+item2_0   feld, das auf den nachfolger
                                                    zeigt
0075 003027 02                  .byt 2              alle datensaetze von struct2
                                                    erhalten die 2
0076 003028 3030                .wor record2        zeige auf den nachfolger
0077 00302a 0000                .wor 0              es gibt keinen vorgaenger
0078 00302c                     *=record2+item2_0   feld, das auf den nachfolger
                                                    zeigt
0079 003030 02                  .byt 2              alle datensaetze von struct2
                                                    erhalten die 2
0080 003031 3930                .wor record3        zeige auf den nachfolger
0081 003033 2730                .wor record1        das ist der vorgaenger
0082 003035                     *=record3+item2_0   feld, das auf den nachfolger
                                                    zeigt
0083 003039 02                  .byt 2              alle datensaetze von struct2
                                                    erhalten die 2
0084 00303a 0000                .wor 0              es gibt keinen nachfolger
0085 00303c 3030                .wor record2        vorgaenger
0086
```

```
0087 /* man kann die datensaetze vom typ struct2 jedoch auch als
0088 zirkulare liste anlegen. der erste satz zeigt dann im feld vorgaenger
0089 auf den letzten satz und der letzte satz
0090 weist im feld nachfolger auf den ersten
0091 hier die alternative disposition
0092 */
0093 00303e                    *=record1+item2_1  feld, das auf den nachfolger
                                                   zeigt
0094 003028 3030               .wor record2       zeige auf den nachfolger
0095 00302a 3930               .wor record3       vorgaenger ist zirkular der l
                                                   etzte satz
0096 00302c                    *=record2+item2 1  feld, das auf den nachfolger
                                                   zeigt
0097 003031 3930               .wor record3       zeige auf den nachfolger
0098 003033 2730               .wor record1       das ist der vorgaenger
0099 003035                    *=record3+item2_1  feld, das auf den nachfolger
                                                   zeigt .
0100 00303a 2730               .wor record1       nachfolger ist zirkular der e
                                                   rste record
0101 00303c 3030               .wor record2       vorgaenger
0102 00303e                    .end
```

SYMBOLTAFEL

item1_0	000000	#0008	item1_1	000001	#0009	item1_2	000003	#0010
item1_3	000005	#0011	item1_4	000007	#0012	item2_0	000000	#0017
item2_1	000001	#0018	item2_2	000003	#0019	item2_3	000005	#0020
record1	003027	#0041	record2	003030	#0042	record3	003039	#0043
satz1	003000	#0037	satz2	00300d	#0038	satz3	00301a	#0039
struct1	000000	#0007	struct11	00000d	#0013	struct2	000000	#0016
struct21	000009	#0021	structber	003000	#0036	structend	003042	#0045
text1	002000	#0031	text2	00200f	#0032	text3	00201e	#0033
textber	002000	#0030						

Vom Programm benutzte Strukturen
In der beifolgenden Programm-Liste haben wir uns ab Zeile 115 die Aufgabe gestellt, einmal den zu satz2 gehörenden Text "hier ist text2" auszugeben. Für den Ablauf bedeutet das: Es ist die erste Struktur aufzufinden, die zum Typ "1" gehört. Beim Label MARKE2 wird ggfs. um die Länge einer Struktur2 weiterpositioniert, falls die gleichartigen Strukturen einmal nicht dicht hintereinander gepackt sind. Vom ersten Datensatz mit dem richtigen Typ "1" können wir uns nun mit den Zeigern auf den Nachfolger durchhangeln, bis wir in COUNT den richtigen Wert erreicht haben. Das Durchhangeln erfolgt per Umladen des Zeigers auf den Nachfolger in den Laufzeiger STRUCTPTR ab Zeile 126.

Wenn wir die Struktur gefunden haben, wird aus ihr der Zeiger auf den Text in item1_2 in die Zero Page umgeladen (ab Zeile 139) und per TEXTOUT zur Ausgabe gebracht. - Es wurde schon erwähnt, daß man den zu einem Typ (Strukturart) gehörenden Datensatz auch unter dem Gesichtspunkt aufsuchen kann, daß in seinen Feldern eine Nummer, ein Name (mit maximaler Länge) oder der Zeiger auf einen Namen enthalten ist. Und man kann in seinen Feldern auch den Zeiger auf eine verbundene Struktur anbringen. Verbundene Listen sind damit ein vielseitiges Hilfsmittel.

Man soll sich in diesem und auch im nachfolgenden Beispiel vor Augen halten, daß der 6502 eine Einadreß-Maschine ist, die jeweils nur an einem Byte arbeitet. Die Dinge werden etwas besser, wenn man den 65802 oder den 65816 nimmt. Hier kann man wenigstens 1 ganzes Adreßwort mit einem Befehl laden. Befriedigend ist das allerdings auch nicht ganz, weil eine volle Adresse

hier 3 Bytes ausmacht (einschl. Bank). Es gäbe allerdings wohl Techniken, mit denen man halbwegs elegant arbeiten könnte. - Wirklich angenehm werden die Verhältnisse dann auf dem 68000. Er kann volle Langwortadressen transportieren, für deren Ansprache es eine vielseitige Bildung der effektiven Adresse gibt, so z.B. auch "Adreßregister indirekt mit Index" und ggfs. noch mit einem Offset dazu. Und der 68000 ist eine Zweiadreß-Maschine: Mit einem Befehl wird etwas von der Quelle zum Ziel gebracht.

```
0105                            ;erkl rung von variablen
0106 00303e                     *=$43               bereich in der zero page
0107 000043          structptr  *=*+2               zeiger auf den beginn der str
                                                    ukturen
0108 000045          structfr   *=*+2               zeiger auf den freien speiche
                                                    r hinter den strukturen
0109 000047          textptr    *=*+2               zeiger auf auszugebende texte
0110 000049          count      *=*+1               hilfszaehler
0111
0112 00004a          kbsout     =$ffd2              ausgabe des cbm
0113
0114 00004a                     *=$3800             start des programmes
0115 003800 204438              jsr setup           strukturzeiger einrichten
0116 003803 204d38              jsr setbot          ende merken
0117                            ;setze auf satz2 und gib seinen text aus
0118 003806 a902                lda #2              hilfszaehler einrichten
0119 003808 8549                sta count
0120 00380a a000                ldy #0              fuer pointeradressierung
0121 00380c b143     marke2     lda (structptr),y   welche struktur-art?
0122 00380e c902                cmp #2
0123 003810 f029                beq erhoeh2
0124 003812 c649     markel     dec count
0125 003814 f014                beq ausgabe         wir sind im richtigen satz
0126 003816 a001                ldy #item1_1        adressiere das feld mit der a
                                                    dresse des nachfolgers
0127 003818 b143                lda (structptr),y
0128 00381a 48                  pha                 den pointer noch nicht zersto
                                                    eren
0129 00381b c8                  iny
0130 00381c b143                lda (structptr),y   high byte
0131 00381e a000                ldy #0
0132 003820 8544                sta structptr+1     adresse des nachfolgers
0133 003822 68                  pla                 low byte
0134 003823 8543                sta structptr
0135 003825 0544                ora structptr+1     sehen, ob ende mit 00 00
0136 003827 d0e9                bne markel          nein
0137 003829 00                  brk                 aktionen, wenn nichts mehr fo
                                                    lgt
0138
0139 00382a a003     ausgabe    ldy #item1_2        den textpointer umladen
0140 00382c b143                lda (structptr),y
0141 00382e 8547                sta textptr
0142 003830 c8                  iny
0143 003831 b143                lda (structptr),y
0144 003833 8548                sta textptr+1       high byte
0145 003835 207238              jsr textout
0146 003838 4c8038              jmp programm2       mache 2. demo
0147
0148 00383b 206438   erhoeh2    jsr adds2           erhoehe um laenge struktur2
0149 00383e 205638              jsr amend?          ende der strukturen erreicht?
```

```
0150 003841 90c9            bcc marke2          nein
0151 003843 00              brk                 was soll dann geschehen?
0152
0153 003844 a900    setup   lda #<structber     pointer auf bereich der struk
                                                turen
0154 003846 8543            sta structptr
0155 003848 a930            lda #>structber     high byte
0156 00384a 8544            sta structptr+1
0157 00384c 60              rts
0158 00384d a942    setbot  lda #<structend     merke das ende
0159 00384f 8545            sta structfr
0160 003851 a930            lda #>structend
0161 003853 8546            sta structfr+1
0162 003855 60              rts
0163 003856 18      amend?  clc                 pruefe auf ende strukturberei
                                                ch
0164 003857 a543            lda structptr
0165 003859 e545            sbc structfr
0166 00385b a544            lda structptr+1
0167 00385d e546            sbc structfr+1
0168 00385f 60              rts
0169 003860 a90d    addsl   lda #structl1       addiere laenge einer struktur
                                                1
0170 003862 d002            bne adds            verzweige immer
0171 003864 a909    adds2   lda #struct21       addiere laenge einer struktur
                                                2
0172 003866 18      adds    clc
0173 003867 6543            adc structptr       alter zeiger in die strukture
                                                n
0174 003869 8543            sta structptr
0175 00386b a544            lda structptr+1
0176 00386d 6900            adc #0
0177 00386f 8544            sta structptr+1
0178 003871 60              rts
0179 003872 a000    textout ldy #0              ausgabe eines textes
0180 003874 b147    textoutl lda (textptr),y    hole zeichen
0181 003876 f007            beq textout2        ende mit 00
0182 003878 20d2ff          jsr kbsout          ausgabe commodore
0183 00387b c8              iny
0184 00387c 4c7438          jmp textoutl
0185 00387f 60      textout2 rts
0186
```

Dynamische Anlage von Strukturen
Ab Zeile 187 der beifolgenden Liste ist einmal vorgeführt, wie man eine zusätzliche Struktur vom Typ struct2 als einen record4 an vorhandene Strukturen speichermäßig anhängt und wie man bei einem Neuzugang die Zeiger auf den Nachfolger und auf den Vorgänger (die Links) umhängt. Wir gehen willkürlich von der Voraussetzung aus, daß die neue Struktur auf Grund eines vorausgegangenen Größenvergleiches oder einer Sortierung ordnungsmäßig vor den record1 gehört. Wir hätten auch auf jeden anderen Datensatz der Struktur setzen können.

Im Programm wird von der Tatsache vorteilhafter Gebrauch gemacht, daß wir bei beiden Strukturtypen in das erste Byte das Familien-Merkmal "1" oder "2" hineingeschrieben haben. Wenn eine vorgefundene Struktur nicht zur gesuchten Familie gehört, wird der Laufzeiger STRUCTPTR um den dann zutreffenden Wert erhöht. In Zeile 200 sind wir dann beim ersten beim ersten Datensatz, der zur Familie gehört. Sein Nachfolger bleibt gleich. Sein Vorgänger wird sich

am Ende des bisher benutzten Speicherplatzes befinden (Pointer in STRUCTFR). Zunächst müssen wir jedoch den Zeiger auf den Vorgänger (auf dem Stack sichern). In linearen Listen hat der Vorgänger die Adresse 0000, d.h. es gibt keinen Vorgänger. Wenn wir jedoch mit den Programmzeilen ab 93 eine zirkulare Liste angefertigt haben, dann ist der Vorgänger der bisher letzte Datensatz record3. Es ist also schon besser, wenn man den Zeiger auf den Vorgänger sichert und ihn ab Zeile 212 in den neu angelegten Datensatz umlädt.

Man kann sich nun vorstellen, daß eine vorhandene Struktur mit ihren Informationen einmal in Fortfall kommen soll. In diesem Fall muß man im Vorgänger und Nachfolger die entsprechenden Zeiger umhängen. Das ist nicht besonders schwierig. Schwieriger wird es, wenn man den dann freiwerdenden Speicherplatz dem System wieder zur Verfügung stellen will. Das ist typisch eine Aufgabe in der "künstlichen Intelligenz" (LISP). In diesen Fällen muß man wissen, wo wieviele Speicherstellen frei geworden sind (ob eine neue Struktur dort hineinpassen würde) und wo die nächstfolgenden freien Plätze sind. Das ist dann auch wieder eine verbundene Liste anderer Art. Es braucht dann eine Speicherverwaltung. Und wenn die Lücken zu groß werden, muß man die noch gültigen Strukturen aneinanderrücken (garbage collect).

Zusammenfassung
Die vorstehenden Beispiele mögen gezeigt haben, daß man C-ähnliche Strukturen in allen Assemblern erklären, initialisieren, benutzen und erweitern kann. Wesentlich ist dabei, daß eine Struktur Felder durchaus unterschiedlichen Datentyps enthalten darf, daß Strukturen in der Reihenfolge des Bedarfes auch mit Untermischung anderer Strukturtypen angelegt werden können. Strukturen kann man an ihrer Zugehörigkeit zu einer Familie erkennen, ggfs. auch am Namen, den man ihnen zuteilt und den man, ob zutreffend, dann absucht. - Bei immer wieder vorkommenden Sortierungsarbeiten muß man keine ganze Datensätze umspeichern, man muß nur die Zeiger umhängen. - Die in Assembler gegebenen Ausführungen mögen auch das Verständnis dafür wecken, was Strukturen in C letztlich sind. - Ohne Kenntnisse von so oder ähnlich angelegten Strukturen wird man in die Betriebssoftware und Programmierung der aktuellen Computer kaum noch eindringen können.

Als Beispiel für den Einsatz solcher Strukturen wird in einem anderen Teil dieses Heftes ein Disassembler-Programm abgedruckt. Es soll nicht unbedingt eine Lücke schließen, jedoch ein Vorbild dafür sein, wie man einen Disassembler je nach Bedarf für jede CPU aufbauen könnte.

```
0187 /* es soll nun ein record4 dynamisch hinzugefuegt werden
0188 er kommt ab adresse structend in den speicher
0189 zeiger auf diesen platz ist in structfr
0190 (muss anschliessend erhoeht werden)
0191 record4 gehoere in der ordnung vor record1
0192 die zeiger in den records berichtigen
0193 */
0194 003880 204438   programm2 jsr setup          basis der strukturen einstell
                                                  en
0195 003883 a000              ldy #0
0196 003885 b143     prog2    lda (structptr),y   finde ersten datensatz nach s
                                                  truktur2
0197 003887 c902              cmp #2
0198 003889 d028              bne erhoeh1         um laenge einer struktur1 erh
                                                  oehen
```

```
0199                                ;den zeiger auf den alten vorgaenger entnehmen
0200 00388b a003                    ldy #item2_2        zeiger auf vorgaenger
0201 00388d b143                    lda (structptr),y   low
0202 00388f 48                      pha
0203 003890 c8                      iny
0204 003891 b143                    lda (structptr),y   high
0205 003893 48                      pha                 erst altes retten
0206 003894 a546                    lda structfr+1      zeiger auf den freispeicher/n
                                                        euer record
0207 003896 9143                    sta (structptr),y   y zeigt noch auf high byte
0208 003898 88                      dey                 y zeigt relativ wieder auf it
                                                        em2 2
0209 003899 a545                    lda structfr
0210 00389b 9143                    sta (structptr),y   auf den vorgaenger ist umgeha
                                                        engt
0211
0212                                ;record4 soll auf alten vorgaenger zeigen
0213 00389d c8                      iny                 y zeigt jetzt relativ auf ite
                                                        m2_2 high byte
0214 00389e 68                      pla
0215 00389f 9145                    sta (structfr),y    das gelangt in den neuen date
                                                        nsatz
0216 0038a1 88                      dey                 als zeiger auf alten vorgaeng
                                                        er
0217 0038a2 68                      pla
0218 0038a3 9145                    sta (structfr),y
0219                                ;jetzt muss record4 noch auf den nachfolger zei
0219                                gen
0220 0038a5 a001                    ldy #item2_1        feld des nachfolgers
0221 0038a7 a543                    lda structptr
0222 0038a9 9145                    sta (structfr),y
0223 0038ab c8                      iny
0224 0038ac a544                    lda structptr+1
0225 0038ae 9145                    sta (structfr),y
0226                                ;jetzt waeren weitere felder in record4 zu fuel
0226                                len
0227 0038b0 20bc38                  jsr addend2         erhoehe den pointer auf das e
                                                        nde
0228
0229 0038b3 206038    erhoeh1      jsr adds1           erhoehe um laenge strukturl
0230 0038b6 205638                  jsr amend?          ende der strukturen erreicht?
0231 0038b9 90ca                    bcc prog2           nein
0232 0038bb 00                      brk                 was soll dann geschehen?
0233
0234 0038bc 18        addend2       clc                 erhoehe den pointer auf das e
                                                        nde der strukturen
0235 0038bd a545                    lda structfr
0236 0038bf 6909                    adc #struct21       fuer 1 struktur2
0237 0038c1 8545                    sta structfr
0238 0038c3 a546                    lda structfr+1
0239 0038c5 6900                    adc #0
0240 0038c7 8546                    sta structfr+1
0241 0038c9 60                      rts
0241 0038ca                                             rts

fehlerzahl insgesamt:
0000
```

Roland Löhr

Disassembler-Modell

In den vergangenen Jahren hat man gelegentlich Quelltexte für Disassembler in den Zeitschriften gefunden, so zuletzt auch in Heft 41 des MICRO MAG. Dort erschien ein Programm, in dem die Befehlssätze der CPU-Typen 6502, 65C02 und 65816 wahlweise eingestellt werden können, auch die verschiedenen Registerbreiten beim 65816. - Für das Studium von Disassemblern sind u.a. auch die sog. ROM-Listings interessant, z.B. für den PC-128 von Commodore. Disassembler für solche Computer haben den Nachteil, daß ihre Texte erstens nachkommentiert sind, daß sie zweitens den Befehlssatz nur einer CPU berücksichtigen und daß sie drittens stark auf das Betriebssystem des Computers ausgerichtet sind, wobei dann auch auf Platzersparnis in den Festwertspeichern geachtet wird, womit die leichte Übertragbarkeit beeinträchtigt wird.

Das hier verkürzt und einfach abgedruckte Modell eines Disassemblers soll von solchen Beschränkungen befreien. Es benutzt nur eine Systemroutine zur Eingabe von der Tastatur und eine zur Bildschirmausgabe. Damit ist leichte Anpaßbarkeit auch für andere Computer gegeben. Die wesentliche Absicht ist es jedoch, einmal den Umgang mit strukturierten Listen vorzuführen, die als ein Array angeordnet sind. In den einzelnen Feldern der Liste sind dabei fast alle Informationen der Zuordnung eines Befehles für die wohlgeformte Ausgabe enthalten. Die eigentliche Logik zur Aufbereitung eines Befehles zu Disassembler-Text vermindert sich damit auf etwa 400 Bytes zwischen den Zeilen 134 und 352.

Das vorgestellte Programm unterscheidet sich damit von früheren Konzepten, die die logische Zerlegung eines zu disassemblierenden Maschinenbefehles vor allem in einer langen Folge von Befehlen erzielten, wobei die immer notwendigen Tabellen dann kürzer sein konnten. Mit jeder Unregelmäßigkeit in der Architektur der Befehle wurde das Programm aufwendiger und unübersichtlicher.

Wer sich im Laufe der Zeit auch mit Mikropozessoren anderer Typen oder Hersteller beschäftigt, steht oft genug vor der Aufgabe, einen entsprechenden Disassembler für die eigenen Zwecke einzusetzen. Der hier vorgestellte Quelltext kann dabei eine bequeme Hilfe sein, denn für solche Anpassungen muß man am logischen Programmfluß wenig ändern. Die wesentlichen notwendigen Arbeiten betreffen die Liste, die man nach der Matrix für die Opcodes lt. Datenblatt anpaßt. Es sind zu den hexadezimalen Kombinationen die mnemonischen Befehlsbezeichnungen zu ändern, die die Adressierungsart kennzeichnenden Folgetexte, die Befehlslänge und ggfs. Extras für die Art der Aufbereitung. Mit einem guten Textsystem ist die Liste dabei wesentlich schneller richtig aufgestellt, als man die veränderte Logik in ein Programm geschrieben hat.

Im vorliegenden Beispiel konnten wir aus dem mit 11 multiplizierten Opcode (Routine MALELF) direkt in eine vollständige Liste adressieren, in der alle 256 möglichen Opcodes des 65816 behandelt sind. In anderen Fällen gibt es große Lücken in der 16*16-Matrix der Befehle, so z.B. bei 6502 allein. Hier mag es sich empfehlen, von gewissen Eckpunkten ausgehend in der (entsprechend kürzeren) Liste zu suchen, ob der zu zerlegende Opcode mit dem relativen Feld CODE übereinstimmt. Erst in diesem Fall wird das Unterprogramm ENTNAHME aufgerufen, um die Parameter zu übertragen. Aus diesem Grund ist das Feld CODE hier enthalten, obwohl es keinem

unmittelbaren Zweck dient. Komprimierungen kann man natürlich auch für die drei Felder LAENGE, CTYP und EXTRAS der Struktur vornehmen. Wenn Speicherplatz kein Problem ist, sollte man darauf nicht zuviel Mühe verwenden.

Das Modell soll solche Anpassungen für andere Zwecke erleichtern. Je nach Computer und Wünschen wird man dabei auch folgendes berücksichtigen: Wenn man re-assemblierbaren Quelltext gewinnen will, wird man die Ausgabe des Programmzählers ab Zeile 137 unterbinden. Man kann stattdessen auch Kunstlabel einsetzen. Man kann den Disassembler auf 2-Pass einstellen: Im ersten Durchgang werden alle Sprung-/Verzweigungsziele und Operanden in eine Tabelle eingetragen und im 2. Durchgang als Label benutzt. Man kann vorsehen, externe Symbole mit Namen zu versehen. Der entstehende Text kann in den Speicher oder auf Floppy abgelegt werden. - Man wird ferner vorsehen, daß Verzweigungsbefehle entweder Label oder absolute Adressen der Ziele ausweisen (Vorbilder z.B. in Heft 41). - Bei Computern mit mehreren Speicherbänken (mit CPU 65816, CBM 7xxx, PC-128) wird man den VIRTPC als Zuordnungszähler verbreitern und die Bankadresse beim Holen der Bytes berücksichtigen.

Zum Programmaufbau
Das Hauptrogramm fragt in den Zeilen von 55...120 zunächst die vom Benutzer gewünschten Betriebsparameter ab, die Adressen von...bis, CPU-TYP? und Registerbreiten bei 65816. (Befehle 65C02 sind vorbereitet, jedoch nicht implementiert). Die Ausgabeschleife zur Disassemblierung bis zur Adresse BIS liegt ab Label AUSGABE. Hier wird das Unterprogramm DISLIN mit seinen Zuträgern benutzt: MALELF zur Umsetzung des Opcodes in eine Adressierung der Parameterliste (je Strukturelement sind 11 Bytes benutzt). ENTNAME besorgt den Parametertransport. MNETEXT gibt den in der Struktur enthaltenen mnemonischen Text per Zeiger direkt aus. Der Opcode und Bytes der Adressierung (je nach Befehleslänge BEFLEN) kommen mit NUMA zur Ausgabe. An die hexadezimale Darstellung von Direktoperanden und Adressen (1...3 Bytes) können sich Texte zur Adressierungsart des Befehles anschließen, wenn der Zeiger ATEXTPTR ungleich Null ist. - Unbekannte Befehle werden per .byt ausgegeben.

Ab Zeile 425 ist eine sehr verkürzte Strukturliste angefügt, die nur für die Opcode-Bereiche von 00...0f und 50...5f typische Formulierungsbeispiele zeigt. Hier nicht aufgeführte Opcodes sind entsprechend mit Parametern zu versehen. Der vollständige Abdruck der Liste hätte dieses Heft geprengt.

```
0004
0005 000000          start     =$2000
0006
0007 000000                    .fil f0:struct-disass0
0008 000000                    .opt lis
0009 /*@ko"0:struct-disass0"
0010 ;disassembler 6502/65816 mit liste arbeitend
0011 roland loehr, 12/1986
0012 */
0013                           ;definition der listenstruktur
0014 000000                    *=0                alle label relativ zu struct
0015 000000          struct                       ;erklaerung einer struktur
0016 000000          code      *=*+1              hexadezimaler opcode
0017 000001          mnemonic  *=*+5              mnemonischer text des befehle
                                                  s + adressierungsart
0018 000006          folgetext *=*+2              was zur adressierungsart ange
                                                  haengt wird
```

```
0019 000008           laenge    *=*+1
0020 000009           ctyp      *=*+1              cputyp zum befehl
0021 00000a           extras    *=*+1              adresse relativer befehle umr
                                                   echnen, wenn <>0
0022 00000b           structl                      laenge der struktur
0023
0024                            ;verkehrszeichen
0025 00000b           cr        =$0d
0026 00000b           escape    =$1b
0027 00000b           space     =$20
0028 00000b           rubout    =$14
0029                            ;variable in der zero page
0030 00000b                     *=$43
0031 000043           suchptr   *=*+2              zeigt auf tafel der befehle
0032 000045           textptr   *=*+2              zeigt auf text der mnmemonics
0033 000047           atextptr  *=*+2              zeigt auf text der adressieru
                                                   ngsart
0034 000049           virtpc    *=*+2              zuordnungszaehler des disasse
                                                   mblers
0035 00004b           bis       *=*+2              endadresse fuer disassembler
0036 00004d           beflen    *=*+1              laenge des befehles
0037 00004e           cputyp    *=*+1              vorgefundene zulaessigkeit
0038 00004f           besonders *=*+1              besonderheiten der adressieru
                                                   ng
0039 000050           cputyp?   *=*+1              vom anwender eingestellte cpu
0040 000051           akkbreit  *=*+1              vom anwender eingestellte akk
                                                   ubreite 65816
0041 000052           xbreit    *=*+1              vom anwender eingestellte bre
                                                   ite der indexregister
0042 000053           opcode    *=*+1              opcode des befehles merken
0043 000054           ergebnis  *=*+2              berechnen der position in der
                                                    struktur
0044 000056           temp      *=*+1              zwischenspeicher ausgabe
0045 000057           binwert   *=*+1              binaerwert einer zifferneinga
                                                   be
0046 000058           count     *=*+1              hilfszaehler
0047 000059           bytflag   *=*+1              00=implementierter befehl, $f
                                                   f=nicht implementiert
0048
0049                            ;systemroutinen commodore
0050 00005a           kgetin    =$ffe4             zeichen aus tastaturpuffer
0051 00005a           kbsout    =$ffd2             bildschirmausgabe
0052 00005a           monitr    =$ee16             einsprung monitor
0053
0054 00005a                     *=start
0055 002000 a977                lda #<text0        beginn interaktiver abfragen
0056 002002 a022                ldy #>text0
0057 002004 209f21              jsr diatext        clear sreen
0058 002007 a979                lda #<textl
0059 002009 a022                ldy #>textl
0060 00200b 209f21              jsr diatext        abfrage ab adresse
0061 00200e 202722              jsr getaddr        hole adresse
0062 002011 b003                bcs startl         normale eingabe
0063 002013 4c16ee              jmp monitr         abbruch mit escape
0064 002016 a554      startl    lda ergebnis
0065 002018 a455                ldy ergebnis+1
0066 00201a 8549                sta virtpc
0067 00201c 844a                sty virtpc+1
0068 00201e 20b121              jsr crlf           abfrage adresse von fertig
```

```
0069 002021 a995            lda #<text2
0070 002023 a022            ldy #>text2
0071 002025 209f21          jsr diatext       abfrage adresse bis
0072 002028 202722          jsr getaddr       hole adresse
0073 00202b b003            bcs start2        normale eingabe
0074 00202d 4c16ee          jmp monitr        abbruch mit escape
0075 002030 a554    start2  lda ergebnis      vorgabe umladen
0076 002032 a455            ldy ergebnis+1
0077 002034 854b            sta bis
0078 002036 844c            sty bis+1
0079 002038 20b121  start3  jsr crlf
0080 00203b a9a3            lda #<text3
0081 00203d a022            ldy #>text3
0082 00203f 209f21          jsr diatext       abfrage cpu-typ
0083 002042 204722          jsr getkey
0084 002045 c90d            cmp #cr           default 6502
0085 002047 d004            bne start4
0086 002049 a900            lda #0
0087 00204b f008            beq start6        cr wird zu 00
0088 00204d c943    start4  cmp #'c'          c fuer 65c02?
0089 00204f f004            beq start6        ist auch okay
0090 002051 c938            cmp #'8'          cpu 65816
0091 002053 d0e3            bne start3        nicht getroffen
0092 002055 8550    start6  sta cputyp?       merken
0093 002057 c938            cmp #'8'          registerabfragen ggfs. ueberg
                                              ehen
0094 002059 f008            beq start6a
0095 00205b a900            lda #0
0096 00205d 8551            sta akkbreit
0097 00205f 8552            sta xbreit        registerbreiten default 8
0098 002061 f032            beq ausgabe
0099 002063 20b121  start6a jsr crlf
0100 002066 a9c9            lda #<text4
0101 002068 a022            ldy #>text4
0102 00206a 209f21          jsr diatext       abfrage akkubreite
0103 00206d 204722          jsr getkey
0104 002070 c90d            cmp #cr
0105 002072 d004            bne start7
0106 002074 a900            lda #0
0107 002076 f002            beq start8        branch always
0108 002078 a980    start7  lda #$80
0109 00207a 8551    start8  sta akkbreit      akku in 8/16 bit eingestellt
0110 00207c 20b121          jsr crlf
0111 00207f a9f4            lda #<text5
0112 002081 a022            ldy #>text5
0113 002083 209f21          jsr diatext       abfrage indexbreite
0114 002086 204722          jsr getkey
0115 002089 c90d            cmp #cr
0116 00208b d004            bne start9
0117 00208d a900            lda #0
0118 00208f f002            beq starta        branch always
0119 002091 a980    start9  lda #$80
0120 002093 8552    starta  sta xbreit        indexregister eingestellt
0121
0122 002095 20a420  ausgabe jsr dislin
0123 002098 38              sec               sind wir am ende?
0124 002099 a549            lda virtpc
0125 00209b e54b            sbc bis
0126 00209d a54a            lda virtpc+1
```

```
0127 00209f e54c              sbc bis+1
0128 0020a1 90f2              bcc ausgabe
0129
0130 0020a3 00                brk                 ende des hauptprogrammes
0131                          ;*****************************
0132
0133                          ;dislin bringt eine disassemblierte zeile zur a
0133                          usgabe
0134 0020a4 20b121   dislin   jsr crlf            neue zeile
0135 0020a7 a900              lda #0              voreinstellung: implementiert
                                                  er befehl
0136 0020a9 8559              sta bytflag
0137 0020ab a54a              lda virtpc+1        gib zuordnungszaehler aus
0138 0020ad 20d421            jsr numa
0139 0020b0 a549              lda virtpc
0140 0020b2 20d421            jsr numa
0141 0020b5 a920              lda #space          space als trenner
0142 0020b7 20d2ff            jsr kbsout
0143 0020ba a000              ldy #0
0144 0020bc b149              lda (virtpc),y      hole opcode an laufender adre
                                                  sse
0145 0020be 8553              sta opcode
0146 0020c0 20f621            jsr malelf          auf die lange liste adressier
                                                  en
0147 0020c3 203821            jsr entnahme        zeiger etc aus struktur nehme
                                                  n
0148 0020c6 a550              lda cputyp?
0149 0020c8 c938              cmp #'8'            brk-befehl bei 65816 2 bytes
0150 0020ca d008              bne disli           andere cpu
0151 0020cc a54f              lda besonders
0152 0020ce c902              cmp #2
0153 0020d0 d002              bne disli           nicht brk-befehl
0154 0020d2 e64d              inc beflen
0155 0020d4 207021   disli     jsr mnetext         mnemonischen text schreiben
0156 0020d7 a44d              ldy beflen
0157 0020d9 2459              bit bytflag         nach .byt noch $ausgeben
0158 0020db 100a              bpl dislin0         zugelassene befehle
0159 0020dd a924              lda #'$'
0160 0020df 20d2ff            jsr kbsout          ausgabe
0161 0020e2 a553              lda opcode          byte fuer .byt
0162 0020e4 20d421            jsr numa            y ist noch 1
0163 0020e7 88       dislin0  dey                 y=0...3
0164 0020e8 f040              beq addvirtpc       implied, dann nur noch zaehle
                                                  r erhoehen
0165 0020ea a924     dislin1  lda #'$'            dollarzeichen
0166 0020ec 20d2ff            jsr kbsout          ausgeben
0167 0020ef a550              lda cputyp?         bei 65816 haengen befehle imm
                                                  ediate
0168 0020f1 f026              beq dislin2         von der eingestellten registe
                                                  rbreite ab
0169 0020f3 a54f              lda besonders
0170 0020f5 c901              cmp #01             codierung fuer immdediate-bef
                                                  ehle
0171 0020f7 d020              bne dislin2         trifft nicht zu
0172 0020f9 a551              lda akkbreit        wie fuer akku?
0173 0020fb f00d              beq dislin1a        keine verfuegungen
0174 0020fd a553              lda opcode
0175 0020ff 290f              and #$0f
0176 002101 c909              cmp #$09            nur auf $09 endend betreffen
                                                  akku
```

```
0177 002103 d005            bne dislinla      nicht den akku abtreffend
0178 002105 e64d            inc beflen
0179 002107 c8              iny
0180 002108 d00f            bne dislin2       akku-# erledigt
0181 00210a a552  dislinla  lda xbreit       und wie fuer indexregister?
0182 00210c f00b            beq dislin2
0183 00210e a553            lda opcode
0184 002110 290f            and #$0f
0185 002112 c909            cmp #$09          nur nicht auf $09 endend betr
                                              effen indexregister
0186 002114 f003            beq dislin2       den akku abtreffend, nicht x,
                                               v
0187 002116                 inc beflen
0188 002118 c8              iny
0189 002119 b149  dislin2   lda (virtpc),y
0190 00211b 20d421          jsr numa          hex als 2 ziffern ausgeben
0191 00211e 88              dey               folgebyte der adressierung
0192 00211f d0f8            bne dislin2
0193 002121 a547            lda atextptr      gibt es folgetext?
0194 002123 0548            ora atextptr+1
0195 002125 f003            beq addvirtpc     kein text der adressierungsar
                                              t
0196 002127 20a321          jsr atextout
0197 00212a 18    addvirtpc clc               erhoehe zuordnungszaehler
0198 00212b a54d            lda beflen
0199 00212d 6549            adc virtpc
0200 00212f 8549            sta virtpc
0201 002131 a54a            lda virtpc+1
0202 002133 6900            adc #0
0203 002135 854a            sta virtpc+1
0204 002137 60    dislinz   rts
0205
0206 002138 18    entnahme  clc               zur struktur relative adresse
0207 002139 a554            lda ergebnis      in absolute umrechnen
0208 00213b 6956            adc #<liste0      start der liste
0209 00213d 8543            sta suchptr
0210 00213f a555            lda ergebnis+1
0211 002141 6923            adc #>liste0
0212 002143 8544            sta suchptr+1
0213
0214 002145 18              clc               information der struktur entn
                                              ehmen
0215 002146 a543            lda suchptr
0216 002148 6901            adc #1            zeiger auf mnemon text
0217 00214a 8545            sta textptr
0218 00214c a544            lda suchptr+1
0219 00214e 6900            adc #0
0220 002150 8546            sta textptr+1
0221 002152 a006            ldy #folgetext    gleich die begleitinformation
                                               umladen
0222 002154 b143            lda (suchptr),y   textpointer adressierungsart
0223 002156 8547            sta atextptr      umladen
0224 002158 c8              iny
0225 002159 b143            lda (suchptr),y
0226 00215b 8548            sta atextptr+1
0227 00215d a008            ldy #laenge
0228 00215f b143            lda (suchptr),y   byte der befehlslaenge
0229 002161 854d            sta beflen
0230 002163 a009            ldy #ctyp
```

```
0231 002165 b143            lda (suchptr),y    cpu-typ zum befehl
0232 002167 854e            sta cputyp
0233 002169 a00a            ldy #extras
0234 00216b b143            lda (suchptr),y    besonderheiten
0235 00216d 854f            sta besonders
0236 00216f 60              rts
0237
0238 002170 a54e    mnetext lda cputyp         befehl zur cpu zugelassen?
0239 002172 f01e            beq mnetext0       ja, immer
0240 002174 2980            and #$80           merkmal der 65816
0241 002176 f004            beq mnebyt         muss 65c02 sein, nicht implem
                                               entiert
0242 002178 a550            lda cputyp?        welchen befehlssatz will der
                                               anwender?
0243 00217a c938            cmp #'8'           merkmal 65816 trifft zu, okay
0244 00217c f014    mnebyt  beq mnetext0       implementierter befehl
0245 00217e c659            dec bytflag        =$ff, nicht implementierter b
                                               efehl
0246 002180 a91f            lda #<text6        ausgabe lenken auf .byt
0247 002182 8545            sta textptr
0248 002184 a923            lda #>text6
0249 002186 8546            sta textptr+1
0250 002188 a901            lda #$01           zeige die besonderheit fuer .
                                               byt
0251 00218a 854d            sta beflen         nur 1 byte ausgeben
0252 00218c a900            lda #0
0253 00218e 8547            sta atextptr       kein folgetext
0254 002190 8548            sta atextptr+1
0255 002192 a000    mnetext0 ldy #0            nun 5 byte mnemonics ausgeben
0256 002194 b145    mnetext1 lda (textptr),y
0257 002196 20d2ff           jsr kbsout
0258 002199 c8               iny
0259 00219a c005             cpy #5
0260 00219c d0f6             bne mnetext1
0261 00219e 60               rts
0262
0263 00219f 8547    diatext  sta atextptr      gib dialogtext aus
0264 0021a1 8448             sty atextptr+1
0265 0021a3 a000    atextout ldy #0            gib text der adressierungsart
                                                aus
0266 0021a5 b147    atext1   lda (atextptr),y
0267 0021a7 f007             beq atextz        ende
0268 0021a9 20d2ff           jsr kbsout
0269 0021ac c8               iny
0270 0021ad 4ca521           jmp atext1
0271 0021b0 60      atextz   rts
0272
0273 0021b1 a90d    crlf     lda #cr           ein cr(lf) ausgeben
0274 0021b3 4cd2ff           jmp kbsout        rts dort
0275
0276                         ;wandle 1 hexbyte in 2 ascii-bytes
0277                         ;beim verlassen der routine:
0278                         ;hohes byte im accu, niedriges in temp
0279
0280 0021b6 48      numwa    pha
0281 0021b7 2c5523           bit bit6          setze v-flag
0282 0021ba 290f             and #$0f
0283 0021bc 20c421           jsr numwa2
```

```
0284 0021bf 68              pla
0285 0021c0 4a              lsr a              a isoliere h herwertigen teil
0286 0021c1 4a              lsr a
0287 0021c2 4a              lsr a
0288 0021c3 4a              lsr a              a 4x rechts
0289 0021c4 0930    numwa2  ora #$30           mach zu ascii
0290 0021c6 c93a            cmp #$3a           berlauf?
0291 0021c8 9004            bcc numwa3
0292 0021ca 08              php                v-bit nicht d. adc zerschiess
                                               en lassen!
0293 0021cb 6906            adc #6             f r a...f
0294 0021cd 28              plp
0295 0021ce 5003    numwa3  bvc numwa4         beim 2. durchgang
0296 0021d0 8556            sta temp           niederwertigen teil
0297 0021d2 b8              clv                reset v-flag
0298 0021d3 60      numwa4  rts
0299
0300 0021d4         numa                       ausgabe einer hexzahl
0301 0021d4 20b621          jsr numwa          umwandlung
0302 0021d7 20d2ff          jsr kbsout
0303 0021da a556            lda temp
0304 0021dc 4cd2ff          jmp kbsout         rts dort
0305
0306                        ;rechnende hilfsroutinen
0307 0021df a900    nullergeb lda #0           ergebnisfeld=0
0308 0021e1 8554            sta ergebnis
0309 0021e3 8555            sta ergebnis+1
0310 0021e5 60              rts
0311
0312 0021e6 a003    addergeb  ldy #3           addiere hexziffer zu ergebnis
0313 0021e8 0654    adder1    asl ergebnis
0314 0021ea 2655              rol ergebnis+1
0315 0021ec 88                dey
0316 0021ed 10f9              bpl adder1
0317 0021ef a557              lda binwert
0318 0021f1 0554              ora ergebnis
0319 0021f3 8554              sta ergebnis
0320 0021f5 60                rts
0321
0322                          ;malelf setzt opcode in relative adressierung u
0322                          m
0323                          ;ziel ist multiplikation mit 11 gemaess struktu
0323                          rlaenge
0324 0021f6 a900    malelf    lda #00            setze per opcode in die struk
                                                 tur
0325 0021f8 8555              sta ergebnis+1
0326 0021fa a553              lda opcode
0327 0021fc 0a                asl a              a *2
0328 0021fd 2655              rol ergebnis+1
0329 0021ff 0a                asl a              a *4
0330 002200 2655              rol ergebnis+1
0331 002202 0a                asl a              a *8
0332 002203 2655              rol ergebnis+1
0333 002205 8554              sta ergebnis       opcode nun *8 genommen
0334 002207 a553              lda opcode         opcode *1 dazu
0335 002209 18                clc
0336 00220a 6554              adc ergebnis
0337 00220c 8554              sta ergebnis
0338 00220e 9002              bcc malelf1
```

```
0339 002210 e655                inc ergebnis+1      ergebnis nun opcode*9
0340 002212 a553      malelf1   lda opcode          nun *2 dazu
0341 002214 0a                  asl a               a *2
0342 002215 08                  php                 carry merken
0343 002216 18                  clc
0344 002217 6554                adc ergebnis
0345 002219 8554                sta ergebnis
0346 00221b 9002                bcc malelf2
0347 00221d e655                inc ergebnis+1
0348 00221f 28        malelf2   plp
0349 002220 a555                lda ergebnis+1
0350 002222 6900                adc #0              uebertrag
0351 002224 8555                sta ergebnis+1
0352 002226 60                  rts
0353
0354 002227 20df21    getaddr   jsr nullergeb       ergebnis=0, adresse einholen
0355 00222a a203                ldx #3
0356 00222c 8658                stx count
0357 00222e 204722    getad1    jsr getkey
0358 002231 c90d                cmp #cr             leere eingabe fuehrt zum abbr
                                                    uch
0359 002233 f00e                beq getad2
0360 002235 c91b                cmp #escape
0361 002237 f00c                beq getad3
0362 002239 20d2ff              jsr kbsout
0363 00223c 20e621              jsr addergeb        ziffer hinzuodern
0364 00223f c658                dec count
0365 002241 10eb                bpl getad1          4 ziffern einholen
0366 002243 38        getad2    sec
0367 002244 60                  rts
0368 002245 18        getad3    clc                 bei escape
0369 002246 60                  rts
0370                            ;getkey nimmt nur hexziffern, cr und escape an
0371 002247 20e4ff    getkey    jsr kgetin          hole ein zeichen von der tast
                                                    atur
0372 00224a c91b                cmp #escape         abbruch mit escape
0373 00224c d001                bne getkey0
0374 00224e 60                  rts
0375 00224f c90d      getkey0   cmp #cr             zeilenabschluss?
0376 002251 d001                bne getkey1
0377 002253 60                  rts
0378 002254 c930      getkey1   cmp #'0'            keine steuerzeichen
0379 002256 90ef                bcc getkey
0380 002258 c947                cmp #'g'            keine hexzahl
0381 00225a b0eb                bcs getkey
0382 00225c c941                cmp #'a'
0383 00225e 900a                bcc getkzif
0384 002260 8556                sta temp            hexziffer merken
0385 002262 e937                sbc #$37            gleich zu binaerwert machen
0386 002264 8557                sta binwert
0387 002266 a556                lda temp
0388 002268 60                  rts
0389 002269 60        getkeyzif rts                 hexziffer mit carry set
0390 00226a c93a      getkzif   cmp #':'            band der nicht-hexziffern
0391 00226c b0d9                bcs getkey          keine ziffer 0...9
0392 00226e 8556                sta temp            merken
0393 002270 290f                and #$0f            binaerwert isolieren
0394 002272 8557                sta binwert
0395 002274 a556                lda temp
0396 002276 60                  rts
```

```
0398 002277 9300      text0     .byt 147,0          clear screen,0
0399 002279 44494153 text1      .byt 'diassemblieren ab adresse $',0
0399 002294 00
0400 002295 42495320 text2      .byt 'bis adresse $',0
0400 0022a2 00
0401 0022a3 57454c43 text3      .byt 'welche cpu? cr=6502, c=65c02, 8=65816',0
0401 0022c8 00
0402 0022c9 52454749 text4      .byt 'registerbreite akku? cr=8 bit, else 16 bit
                                     ',0
0402 0022f3 00
0403 0022f4 52454749 text5      .byt 'registerbreite x/y   cr=8 bit, else 16 bit
                                     ',0
0403 00231e 00
0404 00231f 2e425954 text6      .byt '.byt ',0      auagabe eines bytes
0404 002324 00
0405                            ;texte zur adressierungsart
0406 002325 2c5829    indx      .byt',x)',0
0406 002328 00
0407 002329 2c58      iks       .byt ',x',0
0407 00232b 00
0408 00232c 2c4c29    indl      .byt ',l)',0
0408 00232f 00
0409 002330 2c4c292c indly      .byt ',l),y',0
0409 002335 00
0410 002336 29        indir     .byt ')',0
0410 002337 00
0411 002338 2e41      akku      .byt '.a',0
0411 00233a 00
0412 00233b 292c59    indy      .byt '),y',0
0412 00233e 00
0413 00233f 2c58      iks       .byt ',x',0
0413 002341 00
0414 002342 2c59      yps       .byt ',y',0
0414 002344 00
0415 002345 2c53      srel      .byt ',s',0
0415 002347 00
0416 002348 2c53292c srely      .byt ',s),y',0
0416 00234d 00
0417 00234e 2c4c      absl      .byt ',l',0
0417 002350 00
0418 002351 2c4c58    abslx     .byt ',lx',0
0418 002354 00
0419
0420 002355 40        bit6      .byt $40            bit-operator
0421 002356                     .efi
0422 002356                     .fil f0:struct-disass
0423 /*@ko"0:struct-disass"
0424 */
0425 002356           struktur                      ;bereich der strukturen
0426 002356           liste0
0427 002356 00                  .byt $00            hexwert des opcodes
0428 002357 42524b20            .byt 'brk  '        menomonischer opcode
0429 00235c 0000                .wor  0             text der adressierungsart
0430 00235e 01                  .byt  1             befehlslaenge
0431 00235f 00                  .byt  0             cpu-typ
0432 002360 02                  .byt $02            besonderheit, wenn <>0
0433
0434 002361 01                  .byt $01            hexwert des opcodes
0435 002362 4f524120            .byt 'ora  '        mnemonischer opcode
```

```
0436 002367 2523          .wor  indx          text der adressierungsart
0437 002369 02            .byt  2             befehlslaenge
0438 00236a 00            .byt $0             cpu-typ
0439 00236b 00            .byt $00            besonderheit, wenn <>0
0440
0441 00236c 02            .byt $02            hexwert des opcodes
0442 00236d 434f5020      .byt 'cop  '        mnemonischer opcode
0443 002372 0000          .wor  0             text der adressierungsart
0444 002374 02            .byt  2             befehlslaenge
0445 002375 80            .byt $80            cpu-typ
0446 002376 00            .byt $00            besonderheit, wenn <>0
0447
0448 002377 03            .byt $03            hexwert des opcodes
0449 002378 4f524120      .byt 'ora  '        mnemonischer opcode
0450 00237d 4523          .wor  srel          text der adressierungsart
0451 00237f 02            .byt  2             befehlslaenge
0452 002380 80            .byt $80            cpu-typ
0453 002381 00            .byt $00            besonderheit, wenn <>0
0454
0455 002382 04            .byt $04            hexwert des opcodes
0456 002383 54534220      .byt 'tsb  '        mnemonischer opcode
0457 002388 0000          .wor  0             text der adressierungsart
0458 00238a 02            .byt  2             befehlslaenge
0459 00238b c0            .byt $c0            cpu-typ
0460 00238c 00            .byt $00            besonderheit, wenn <>0
0461
0462 00238d 05            .byt $05            hexwert des opcodes
0463 00238e 4f524120      .byt 'ora  '        mnemonischer opcode
0464 002393 0000          .wor  0             text der adressierungsart
0465 002395 02            .byt  2             befehlslaenge
0466 002396 00            .byt $00            cpu-typ
0467 002397 00            .byt $00            besonderheit, wenn <>0
0468
0469 002398 06            .byt $06            hexwert des opcodes
0470 002399 41534c20      .byt 'asl  '        mnemonischer opcode
0471 00239e 0000          .wor  0             text der adressierungsart
0472 0023a0 02            .byt  2             befehlslaenge
0473 0023a1 00            .byt $00            cpu-typ
0474 0023a2 00            .byt $00            besonderheit, wenn <>0
0475
0476 0023a3 07            .byt $07            hexwert des opcodes
0477 0023a4 4f524120      .byt 'ora ('        mnemonischer opcode
0478 0023a9 2c23          .wor  indl          text der adressierungsart
0479 0023ab 02            .byt  2             befehlslaenge
0480 0023ac 80            .byt $80            cpu-typ
0481 0023ad 00            .byt $00            besonderheit, wenn <>0
0482
0483 0023ae 08            .byt $08            hexwert des opcodes
0484 0023af 50485020      .byt 'php  '        mnemonischer opcode
0485 0023b4 0000          .wor  0             text der adressierungsart
0486 0023b6 01            .byt  1             befehlslaenge
0487 0023b7 00            .byt $00            cpu-typ
0488 0023b8 00            .byt $00            besonderheit, wenn <>0
0489
0490 0023b9 09            .byt $09            hexwert des opcodes
0491 0023ba 4f524120      .byt 'ora #'        mnemonischer opcode
0492 0023bf 0000          .wor  0             text der adressierungsart
0493 0023c1 02            .byt  2             befehlslaenge
0494 0023c2 00            .byt $00            cpu-typ
```

```
0495 0023c3 01                .byt $01          besonderheit, wenn <>0
0496
0497 0023c4 0a                .byt $0a          hexwert des opcodes
0498 0023c5 41534c20          .byt 'asl '       mnemonischer opcode
0499 0023ca 3823              .wor  akku        text der adressierungsart
0500 0023cc 01                .byt  1           befehlslaenge
0501 0023cd 00                .byt $00          cpu-typ
0502 0023ce 00                .byt $00          besonderheit, wenn <>0
0503
0504 0023cf 0b                .byt $0b          hexwert des opcodes
0505 0023d0 50484420          .byt 'phd '       mnemonischer opcode
0506 0023d5 0000              .wor  0           text der adressierungsart
0507 0023d7 01                .byt  1           befehlslaenge
0508 0023d8 80                .byt $80          cpu-typ
0509 0023d9 00                .byt $00          besonderheit, wenn <>0
0510
0511 0023da 0c                .byt $0c          hexwert des opcodes
0512 0023db 54534220          .byt 'tsb '       mnemonischer opcode
0513 0023e0 0000              .wor  0           text der adressierungsart
0514 0023e2 03                .byt  3           befehlslaenge
0515 0023e3 c0                .byt $c0          cpu-typ
0516 0023e4 00                .byt $00          besonderheit, wenn <>0
0517
0518 0023e5 0d                .byt $0d          hexwert des opcodes
0519 0023e6 4f524120          .byt 'ora '       mnemonischer opcode
0520 0023eb 0000              .wor  0           text der adressierungsart
0521 0023ed 03                .byt  3           befehlslaenge
0522 0023ee 00                .byt $00          cpu-typ
0523 0023ef 00                .byt $00          besonderheit, wenn <>0
0524
0525 0023f0 0e                .byt $0e          hexwert des opcodes
0526 0023f1 41534c20          .byt 'asl '       mnemonischer opcode
0527 0023f6 0000              .wor  0           text der adressierungsart
0528 0023f8 03                .byt  3           befehlslaenge
0529 0023f9 00                .byt $00          cpu-typ
0530 0023fa 00                .byt $00          besonderheit, wenn <>0
0531
0532 0023fb 0f                .byt $0f          hexwert des opcodes
0533 0023fc 4f524120          .byt 'ora '       mnemonischer opcode
0534 002401 4e23              .wor  absl        text der adressierungsart
0535 002403 04                .byt  4           befehlslaenge
0536 002404 80                .byt $80          cpu-typ
0537 002405 00                .byt $00          besonderheit, wenn <>0
0987 0026c6                   .opt lis
0988 0026c6           liste5
0989 0026c6 50                .byt $50          hexwert des opcodes
0990 0026c7 42564320          .byt 'bvc '       menomonischer opcode
0991 0026cc 0000              .wor  0           text der adressierungsart
0992 0026ce 02                .byt  2           befehlslaenge
0993 0026cf 00                .byt  0           cpu-typ
0994 0026d0 80                .byt $80          besonderheit, wenn <>0
0995
0996 0026d1 51                .byt $51          hexwert des opcodes
0997 0026d2 454f5220          .byt 'eor ('      mnemonischer opcode
0998 0026d7 3b23              .wor  indy        text der adressierungsart
0999 0026d9 02                .byt  2           befehlslaenge
1000 0026da 00                .byt $00          cpu-typ
1001 0026db 00                .byt $00          besonderheit, wenn <>0
1002
```

```
1003 0026dc 52          .byt $52          hexwert des opcodes
1004 0026dd 454f5220    .byt 'eor ('      mnemonischer opcode
1005 0026e2 3623        .wor  indir       text der adressierungsart
1006 0026e4 02          .byt  2           befehlslaenge
1007 0026e5 c0          .byt $c0          cpu-typ
1008 0026e6 00          .byt $00          besonderheit, wenn <>0
1009
1010 0026e7 53          .byt $53          hexwert des opcodes
1011 0026e8 454f5220    .byt 'eor ('      mnemonischer opcode
1012 0026ed 4823        .wor  srely       text der adressierungsart
1013 0026ef 02          .byt  2           befehlslaenge
1014 0026f0 80          .byt $80          cpu-typ
1015 0026f1 00          .byt $00          besonderheit, wenn <>0
1016
1017 0026f2 54          .byt $54          hexwert des opcodes
1018 0026f3 4d564e20    .byt 'mvn  '      mnemonischer opcode
1019 0026f8 0000        .wor  0           text der adressierungsart
1020 0026fa 03          .byt  3           befehlslaenge
1021 0026fb 80          .byt $80          cpu-typ
1022 0026fc 00          .byt $00          besonderheit, wenn <>0
1023
1024 0026fd 55          .byt $55          hexwert des opcodes
1025 0026fe 454f5220    .byt 'eor  '      mnemonischer opcode
1026 002703 2923        .wor  iks         text der adressierungsart
1027 002705 02          .byt  2           befehlslaenge
1028 002706 00          .byt $00          cpu-typ
1029 002707 00          .byt $00          besonderheit, wenn <>0
1030
1031 002708 56          .byt $56          hexwert des opcodes
1032 002709 4c535220    .byt 'lsr  '      mnemonischer opcode
1033 00270e 0000        .wor  0           text der adressierungsart
1034 002710 02          .byt  2           befehlslaenge
1035 002711 00          .byt $00          cpu-typ
1036 002712 00          .byt $00          besonderheit, wenn <>0
1037
1038 002713 57          .byt $57          hexwert des opcodes
1039 002714 454f5220    .byt 'eor ('      mnemonischer opcode
1040 002719 3023        .wor  indly       text der adressierungsart
1041 00271b 02          .byt  2           befehlslaenge
1042 00271c 80          .byt $80          cpu-typ
1043 00271d 00          .byt $00          besonderheit, wenn <>0
1044
1045 00271e 58          .byt $58          hexwert des opcodes
1046 00271f 434c4920    .byt 'cli  '      mnemonischer opcode
1047 002724 0000        .wor  0           text der adressierungsart
1048 002726 01          .byt  1           befehlslaenge
1049 002727 00          .byt $00          cpu-typ
1050 002728 00          .byt $00          besonderheit, wenn <>0
1051
1052 002729 59          .byt $59          hexwert des opcodes
1053 00272a 454f5220    .byt 'eor  '      mnemonischer opcode
1054 00272f 4223        .wor  yps         text der adressierungsart
1055 002731 03          .byt  3           befehlslaenge
1056 002732 00          .byt $00          cpu-typ
1057 002733 00          .byt $00          besonderheit, wenn <>0
1058
1059 002734 5a          .byt $5a          hexwert des opcodes
1060 002735 50485920    .byt 'phy  '      mnemonischer opcode
1061 00273a 0000        .wor  0           text der adressierungsart
```

```
1062 00273c 01              .byt  1           befehlslaenge
1063 00273d c0              .byt $c0          cpu-typ
1064 00273e 00              .byt $00          besonderheit, wenn <>0
1065
1066 00273f 5b              .byt $5b          hexwert des opcodes
1067 002740 54434420        .byt 'tcd  '      mnemonischer opcode
1068 002745 0000            .wor  0           text der adressierungsart
1069 002747 01              .byt  1           befehlslaenge
1070 002748 80              .byt $80          cpu-typ
1071 002749 00              .byt $00          besonderheit, wenn <>0
1072
1073 00274a 5c              .byt $5c          hexwert des opcodes
1074 00274b 4a4d5020        .byt 'jmp  '      mnemonischer opcode
1075 002750 4e23            .wor  absl        text der adressierungsart
1076 002752 04              .byt  4           befehlslaenge
1077 002753 80              .byt $80          cpu-typ
1078 002754 00              .byt $00          besonderheit, wenn <>0
1079
1080 002755 5d              .byt $5d          hexwert des opcodes
1081 002756 454f5220        .byt 'eor  '      mnemonischer opcode
1082 00275b 2923            .wor  iks         text der adressierungsart
1083 00275d 03              .byt  3           befehlslaenge
1084 00275e 00              .byt $00          cpu-typ
1085 00275f 00              .byt $00          besonderheit, wenn <>0
1086
1087 002760 5e              .byt $5e          hexwert des opcodes
1088 002761 4c535220        .byt 'lsr  '      mnemonischer opcode
1089 002766 2923            .wor  iks         text der adressierungsart
1090 002768 03              .byt  3           befehlslaenge
1091 002769 00              .byt $00          cpu-typ
1092 00276a 00              .byt $00          besonderheit, wenn <>0
1093
1094 00276b 5f              .byt $5f          hexwert des opcodes
1095 00276c 454f5220        .byt 'eor  '      mnemonischer opcode
1096 002771 5123            .wor  absix       text der adressierungsart
1097 002773 04              .byt  4           befehlslaenge
1098 002774 80              .byt $80          cpu-typ
1099 002775 00              .byt $00          besonderheit, wenn <>0
1102 002776                 .fil f0:struct-disass1
1779 002b96                 .fil f0:struct-disass2
2457 002fb6                 .end
```

Martin Albrecht, 4350 Recklinghausen

C-Bibliotheksverwaltung

Die Entwicklungssprache der Wahl auf den Atari ST-Rechnern ist eindeutig C. Die hohe Produktivität wird aber besonders beim C-Compiler des Entwicklungspaketes durch zahlreiche Fehler in den Bibliotheken zunichte gemacht. Einige Probleme lassen sich umgehen, wenn auf die Benutzung der Standard-Bilbliothek verzichtet wird und stattdessen direkte Aufrufe des Betriebssystems eingesetzt werden. Das ist auch beim nachstehenden Programm der Fall. Damit ist aber die Übertragbarkeit auf andere Systeme in Frage gestellt.

Bedingt durch die wechselvolle Geschichte dieses C-Compilers und seiner Bibliotheken (Alcyon, Digital Research, Atari, Fließkommapaket von Motorola) ist auf schnelle Beseitigung der Fehler durch den Hersteller nicht zu hoffen. Die Patchanleitungen in /1/ und /2/ bieten deshalb Hilfe zur Selbsthilfe.

Um die Reparaturen auszuführen, benötigt man ein Verwaltungsprogramm für Bibliotheken. Im Entwicklungspaket soll das CP/M-68K Bibliotheksprogramm AR68 diese Aufgabe erfüllen. Die in /1/ vorgestellten Patches konnten mit der vorliegenden Version (Stand März 1986) problemlos vollzogen werden. AR68 weigerte sich aber strikt, das nach /2/ korrigierte Modul auszuwechseln.

Aufgrund dieser Weigerung und weiterer Ungereimtheiten in der Bedienung und der Dokumentation von AR68 wurde hier kurzerhand ein neues Bilbliotheksprogramm aufgelegt. Weil keinerlein Information über den Aufbau der Bibliotheken greifbar war, mußte die Struktur mit dem Debugger SID erforscht werden. Natürlich kann auf diese Weise keine letztgültige Sicherheit über die Bedeutung der verschiedenen Informationen gewonnen werden. Die Ergebnisse reichen aber aus, um ein funktionierendes Verwaltungsprogramm aufzubauen. Hier der Aufbau der zugrundeliegenden Strukturen:

Aufbau einer Bibliothek:

Bytes	Inhalt	Funktion
2	$FF65	Kennung: Bibliothek
28	s.u.	Modulheader
n		Objektcode
28	s.u.	nächster Modulheader
m		Objektcode
...		
...		
2	$0000	Bibliotheksende

Aufbau des Modulheaders:

Bytes	Inhalt	Funktion
12		Filename: xxxxxxxx.yyy
8	$0000000000000000	$00 als Ende des Filenamens, sonst ?
2	$01B6	immer $01B6, vermutlich Unix Zugriffsinfo
2	$0000	immer $0000, Bedeutung ?
2		Objektcodelänge (im Programm unsigned int)
2	$0000	immer $0000, Bedeutung ?

Aufbau des Objektcodes:

Bytes	Inhalt	Funktion
2	$601A	Kennung: Objektfile
4		Größe des Text-Segments
4		Größe des Data-Segments
4		Größe des BSS-Segments
4		Größe der Symboltafel
4	$00000000	reserviert, immer 0
4	$00000000	reserviert, immer 0
2	$0000	reserviert, immer 0
i		Text-Segment
k		Data-Segment
l		BSS-Segment
m		Symboltabelle
n		Verschiebeinformation

Die genaue Bedeutung der Konstanten $01B6 im Modulheader bleibt unklar. Sie rührt wohl aus der Unix-Vergangenheit der Bibliotheken her. Es konnte auch nicht festgestellt werden, ob die Objektcodelänge als 16-Bit oder 32-Bit Zahl einzutragen ist. Im Programm ist eine vorzeichenfreie 16-Bit Zahl vorgesehen. Die Objektcodelänge ist dadurch auf 65536 Bytes beschränkt. Ebenfalls unklar ist, wie eine Bibliothek enden muß. Experimente haben gezeigt, daß der Linker das

Ende daran erkennt, daß der Modulheader mit $00 beginnt. Die Originalbibliotheken führen hinter dem letzten Modul zwischen vier und mehreren Dutzend Null-Bytes. Im nachstehenden Programm wird ein Null-Wort ($0000) verwendet. Die Strktur des Objectcodes entspricht dem Aufbau, den der Assembler erzeugt. Objektfiles können deshalb ohne Umwandlung in Bibliotheken übernommen werden.

Nachdem sich der Aufbau einer Bibliothek als eine einfache Liste der Objectcodes, versehen mit einem Vorspann von Zusatzinformation, herausgestellt hat, gestaltet sich das Verwaltungsprogramm sehr einfach. Um ein Modul einzufügen oder zu löschen, wird der Inhalt der Bibliothek in ein Hilfsfile umgeschaufelt. Dabei wird das entsprechende Modul ausgelassen bzw. ein neuer Modulheader erzeugt und das Modul zusätzlich eingelesen. Ebenso einfach wird die Kopie eines Moduls erzeugt. Es muß nur der Objektcode ohne Modulheader in einem File abgelegt werden. Um den Inhalt einer Bibliothek anzuzeigen, wird die Objektcode-Länge und der Modulname jedes Moduls ausgegeben.

Bei dem nachstehenden Programm handelt es sich um eine verkürzte Version, die nur die unbedingt erforderlichen Funktionen bietet (siehe Kurzanleitung im Programmkopf). Weitere Möglichkeiten, wie Ersetzen und Anfügen eines Moduls und Erzeugen einer Bibliothek können ebenso wie eine detaillierte Fehlerbehandlung leicht selbst programmiert werden. Um das Programm kurz zu halten, wurde konsequent auf die Funktion printf() verzichtet. Außerdem wurde das Startfile nach /3/ verwendet, was nochmals einige KBytes einspart.

Das Programm ist zum Einsatz mit dem Kommandointerpreter COMMAND.TOS gedacht. Soll es als .TTP-Anwendung benutzt werden, dann muß vor dem Programmende (vor exit() in ferror() und in main()) ein Aufruf von Cconin() (entspricht gemdos(0x1)) eingefügt werden. Dadurch bleibt der Textbildschirm erhalten, bis eine beliebige Taste gedrückt wird.

Daß nur mit einer Arbeitskopie und nicht mit den Original-Bibliotheken gearbeitet werden sollte, braucht wohl nicht betont zu werden. Weil keine Pfadnamen zugelassen sind, müssen sich Verwaltungsprogramm und Bibliotheken in demselben Directory befinden. Die Benutzung einer Ramdisk erscheint mit Blick auf die vielen Diskettenzugriffe sinnvoll.

Literatur:

/1/ Martin Albrecht: Patches für die C-Bibliothek des Atari ST, MICRO MAG Nr. 47, September 1986.
/2/ Matthias Köfferlein: Ein Fehler in der LIBF.A-Bibliothek, mc Nr. 9, September 1986
/3/ Peter Glasmacher: Keine Pilze mehr, Eigene TOS-Applikation auf dem Atari 520ST, c't Nr. 11, november 1985.

```
/*********************************************************************/
/* Archivprogramm für CM/M 68K Bibliotheken                          */
/* für das ATARI ST C-Entwicklungspaket von Digital Research         */
/*********************************************************************/
/* Copyright September 1986:  Martin Albrecht, Am Bärenbach 18,      */
/* Alle Rechte vorbehalten.   D-4350 Recklinghausen.                 */
/* verkürzte Version für das Micro Mag, 24.10.1986                   */
/*********************************************************************/
/* Befehlsvorrat:                                                    */
/*                                                                   */
/* delete:  lösche ein Objektmodul                                   */
/* Syntax:  lib d Bibliothek Objektname                              */
/*                                                                   */
/* insert:  füge ein Objektmodul hinter Objektname ein               */
/* Syntax:  lib i Bibliothek Objektfile Objektname                   */
/*                                                                   */
/* view:    zeige den Inhalt einer Bibliothek an                     */
/* Syntax:  lib v Bibliothek                                         */
/*                                                                   */
/* extract: lege eine Kopie eines Objektmoduls als Objektfile an     */
/* Syntax:  lib x Bibliothek Objektfile                              */
/*                                                                   */
/* Bei falscher Syntax oder fehlenden Argumenten erfolgt ein Hinweis */
/* auf die korrekte Benutzung des Befehls.  Die Angabe eines         */
/* Pfadnamens ist nur bei Files erlaubt, die ausschließlich gelesen  */
/* werden. Fehler bein Zugriff auf die Dateien führen zu einer       */
/* Fehlermeldung und zur Beendigung des Programms. Im Fehlerfall     */
```

```
/* bleibt in manchen Fällen das File LIB.TMP erhalten, das als */
/* Arbeitsfile benutzt wird. Alle Parameter in der Kommandozeile */
/* werden immer in Kleinschrift umgewandelt. Modulnamen werden daher */
/* immer in Kleinschrift eingetragen, unabhängig von der */
/* Großschreibung in der Kommandozeile. */
/* Alle Befehle quittieren die korrekte Ausführung mit einer */
/* Meldung. Falls keine Meldung erfolgt konnte der Befehl nicht */
/* ausgeführt werden. */
/* Kommt ein Modul in einer Bibliothek mehrfach vor, dann wird der */
/* Befehl auf jedes Modul angewandt. */
/* Die Modulgröße ist auf 65536 Bytes beschränkt */
/* */
/*****************************************************************************/

/*****************************************************************************/
/* allgemeine Definitionen */
/*****************************************************************************/
#define TRUE        1
#define FALSE       0

#define LIB         0xFF65    /* Filebeginn Bibliothek */
#define OBJ         0x601A    /* Filebeginn Objektfile */
#define END         0         /* Bibliotheksende */

/*****************************************************************************/
/* externe Funktionen */
/*****************************************************************************/
extern long gemdos();         /* gemdos() gibt long-Werte zurück */

/*****************************************************************************/
/* Definitionen zum Zugriff auf TOS Funktionen */
/*****************************************************************************/
#define Fsetdta(a)       gemdos(0x1A,a)           /* DTA setzen */
#define Fsfirst(a,b)     (int)gemdos(0x4E,a,b)    /* 1. Fileeintrag suchen */
#define Fcreate(a,b)     (int)gemdos(0x3C,a,b)    /* File anlegen */
#define Fopen(a,b)       (int)gemdos(0x3D,a,b)    /* File öffnen */
#define Fread(a,b,c)     gemdos(0x3F,a,b,c)       /* File lesen */
#define Fwrite(a,b,c)    gemdos(0x40,a,b,c)       /* File schreiben */
#define Fseek(a,b,c)     gemdos(0x42,a,b,c)       /* Filepointer pos. */
#define Fclose(a)        (int)gemdos(0x3E,a)      /* File schließen */
#define Fdelete(a)       (int)gemdos(0x41,a)      /* File löschen */
#define Frename(a,b,c)   (int)gemdos(0x56,a,b,c)  /* File umbenennen */
#define Cconws(a)        gemdos(0x09,a)           /* String ausgeben */

/*****************************************************************************/

/* globale Variablen */
/*****************************************************************************/
struct                           /* Disk Transfer Adress (Fileinfo) */
{
     char f_osdata[21];          /* OS-Daten */
     char f_attr;                /* Fileattribut */
     int  f_time;                /* Zeitstempel */
     int  f_date;                /* Datumsstempel */
     long f_size;                /* Filegröße */
     char f_name[13];            /* Filename */
} dta;

struct                           /* Bibliothekskopf */
{
     char o_name[20];            /* Objektfilename, mit '\0' aufgefüllt */
     int  o_type;                /* immer 0x01B6 */
     int  o_junk1;               /* immer 0 */
     unsigned int  o_length;     /* Objektfilelänge */
     int  o_junk2;               /* immer 0 */
} kopf;

int  lhandle;        /* Bibliotheksfile Filehandle */
int  thandle;        /* Zwischenfile Filehandle */
int  ohandle;        /* Objektfile Filehandle */
```

```
long length;          /* Länge */
long fres;            /* Resultat Fileoperation/Fehlernummer */
int  typ;             /* Filetyp (Bibl. oder Objektfile, Bibl.-Ende) */
int  btyp;            /* Filetyp */
int  objects;         /* Flag: sind keine/sind Objekte vorhanden */
int  gef;             /* Flag für Stringvergleich */
char numstr[10];      /* für Zahlenumwandlung */
int  i;               /* Index */
int  x;               /* Laufvariable */
char buffer[2000];    /* Zwischenspeicher zum Kopieren */

/*****************************************************************************/
/* int Variable dezimal in String umwandeln                                  */
/*****************************************************************************/
dstring(n)
unsigned int n;
{
     unsigned int a;

     if((a=n/10) != 0)         /* durch 10 teilen */
          dstring(a);          /* weiter wenn != 0 */
     numstr[i++] = n%10+'0';   /* als Zeichen in den Zahlstring */
}

/*****************************************************************************/
/* int Variable dezimal ausgeben                                             */
/*****************************************************************************/
numout(z)
unsigned int z;
{
     i=0;                      /* Index in den Zahlstring */
     dstring(z);               /* umwandeln */
     numstr[i]='\0';           /* und korrekt beenden */
     i=7-strlen(numstr);       /* rechtsbündig in ein 7er-Feld */

     for(; i>0; i--)           /* führende Leerzeichen ausgeben */
          Cconws(" ");

     Cconws(numstr);           /* Zahlstring ausgeben */
     Cconws(" ");
}

/*****************************************************************************/
/* String in Kleinbuchstaben umwandeln                                       */
/*****************************************************************************/
in_klein(c)
char *c;  /* Stringbeginn */
{

     for(; *c; c++)
          if(*c >= 'A' && *c <= 'Z')     /* Großbuchstabe ? */
               *c = *c - 'A' + 'a';      /* in Kleinbuchstaben wandeln */
}

/*****************************************************************************/
/* Floppyfehler                                                              */
/*****************************************************************************/
ferror(file,erno)
char *file;     /* Filename */
long erno;      /* Fehlernummer */
{
     Cconws("\n\rFile ");      /* Filenamen ausgeben */
     Cconws(file);
     Cconws(": ");

     if(erno >= 0)
          /* Schreib/Lese-Fehler */
          Cconws("zu wenig/keine Daten gelesen/geschrieben\n\r");

     else
     {
```

```
          /* TOS-Fehler */
          Cconws("TOS-Fehler Nr. ");
          erno = -erno;
          i=0;                              /* Index in den Zahlstring */
          dstring((unsigned int)erno);      /* umwandeln */
          numstr[i]='\0';                   /* und korrekt beenden */
          Cconws(numstr);                   /* Zahlstring ausgeben */
          Cconws("\n\r");
     }

     exit(0);  /* Files schließen, Programm beenden */
}

/*************************************************************************/
/* Fileinhalt übertagen                                                  */
/*************************************************************************/
transfer(num,h1,h2,n1,n2)
long num;      /* Anzahl Bytes */
int  h1,h2;    /* Filehandles */
char *n1,*n2;  /* Filenamen */
{
     long now;                  /* gerade übertragene Anzahl */
     long fres;

     do
     {
          now = (num < 2000L) ? num :2000L;       /* Anzahl festlegen */
          num=num-now;           /* und von der Gesamtzahl abziehen */

          fres=Fread(h1,now,buffer);     /* einlesen */
          if(fres < 0 || fres != now)
               ferror(n1,fres);

          fres=Fwrite(h2,now,buffer);    /* schreiben */
          if(fres < 0 || fres != now)
               ferror(n2,fres);
     }
     while(num > 0);            /* solange noch welche übrig sind */
}

/*************************************************************************/
/* File öffnen                                                           */
/* gibt die ersten beiden Bytes eines Files als int zurück               */
/*************************************************************************/
int
fileo(fname,handle,length)
char *fname;   /* Filename */
int  *handle;  /* Filehandle */
long *length;  /* Filelänge */
{

     long fres;          /* Fehlerart */
     int  filehandle;    /* Filenummer */
     int  typ;           /* Filetyp */

     Fsetdta(&dta);           /* DTA setzen */

     fres=Fsfirst(fname,0);   /* File suchen */
     if(fres != 0)
          ferror(fname,fres); /* Fehler */

     fres=Fopen(fname,2);     /* File öffnen */
     if(fres < 0)
          ferror(fname,fres); /* Fehler */
     else
          filehandle=fres;    /* Filehandle übernehmen */

     *length=dta.f_size;      /* Filelänge */
     *handle=filehandle;      /* Filehandle */
```

```
     fres=Fread(filehandle,2L,&typ);     /* Filetyp einlesen */
     if(fres < 0 || fres != 2L)
          ferror(fname,fres);

     return(typ);              /* Filetyp zurückgeben */
}

/*************************************************************************/
/* delete: lösche Bibliotheksmodul                                       */
/*************************************************************************/
delete(argc,argv)
int  argc;     /* Anzahl Argumente in der Kommandozeile */
char *argv[];  /* Zeiger auf Argumentstrings */
{
     if(argc < 4)
     {
          /* zu wenig Argumente, Meldung */
          Cconws("Syntax für delete: d Bibliothek Objektname\n\r");
          exit(0);
     }

     /* Bibliothek öffnen */
     typ=fileo(argv[2],&lhandle,&length);
     if(typ == LIB)
     {
          /* Zwischenfile eröffnen */
          fres=Fcreate("LIB.TMP",0);
          if(fres < 0)
               ferror("LIB.TMP",fres);
          else
               thandle=fres;

          /* und Bibliothekskennung schreiben */
          fres=Fwrite(thandle,2L,&typ);
          if(fres < 0 || fres != 2L)
               ferror("LIB.TMP",fres);

          /* bis auf das gesuchte Modul alle ins Zwischenfile kopieren */
          do
          {
               /* Objektmodul-Kopf einlesen */
               fres=Fread(lhandle,28L,&kopf);
               if(fres < 0)
                    ferror(argv[2],fres);

               objects=FALSE;

               /* Modulkopf analysieren */
               if(fres == 28L && kopf.o_length > 0)
               {
                    /* Vergleich mit dem gesuchten Modulnamen */
                    in_klein(kopf.o_name);   /* erzwinge Kleinbuchstaben */
                    gef=strcmp(&kopf,argv[3]);

                    if(gef == 0)
                    {
                         /* gefunden, Vollzugsmeldung ausgeben */
                         Cconws("Modul ");
                         Cconws(argv[3]);
                         Cconws(" in Bibliothek ");
                         Cconws(argv[2]);
                         Cconws(" gelöscht\n\r");

                         /* und Modul auslassen */
                         fres=Fseek((long)kopf.o_length,lhandle,1);
                         if(fres < 0)
                              ferror(argv[2],fres);
                    }

                    else
                    {
```

```
                    /* nicht gefunden, Modulkopf kopieren */
                    fres=Fwrite(thandle,28L,&kopf);
                    if(fres < 0 || fres != 28L)
                         ferror("LIB.TMP",fres);

                    /* Objektmodul kopieren */
                    transfer((long)kopf.o_length,
                              lhandle,thandle,argv[2],"LIB.TMP");
               }

               objects=TRUE;
          }
     }
     while(objects);

     /* Bibliotheksende schreiben */
     typ=END;
     fres=Fwrite(thandle,2L,&typ);
     if(fres < 0 || fres != 2L)
          ferror("LIB.TMP",fres);

     /* Zwischenfile schließen */
     fres=Fclose(thandle);
     if(fres < 0)
          ferror("LIB.TMP",fres);

     /* Bibliothek schließen */
     fres=Fclose(lhandle);
     if(fres < 0)
          ferror(argv[2],fres);

     /* alte Bibliothek löschen */
     fres=Fdelete(argv[2]);
     if(fres < 0)
          ferror(argv[2],fres);

     /* Zwischenfile wird neue Bibliothek */
     fres=Frename(0,"LIB.TMP",argv[2]);
     if(fres < 0)
          ferror("LIB.TMP",fres);
}

else
{
     /* Fehler, File ist keine Bibliothek */
     Cconws(argv[2]);
     Cconws(" ist keine Bibliothek\n\r");

     /* angebliche Bibliothek schließen */
     fres=Fclose(lhandle);
     if(fres < 0)
          ferror(argv[2],fres);
}
}

/*************************************************************************/
/* insert: füge Bibliotheksmodul ein                                     */
/*************************************************************************/
insert(argc,argv)
int  argc;     /* Anzahl Argumente in der Kommandozeile */
char *argv[];  /* Zeiger auf Argumentstrings */
{
     if(argc < 5)
     {
          /* zu wenig Argumente, Meldung */
          Cconws("Syntax für insert: i Bibliothek Objektfile Objektname\n\r");
          exit(0);
     }

     /* Bibliothek öffnen */
     typ=fileo(argv[2],&lhandle,&length);
```

```
if(typ == LIB)
{
     /* Zwischenfile eröffnen */
     fres=Fcreate("LIB.TMP",0);
     if(fres < 0)
          ferror("LIB.TMP",fres);
     else
          thandle=fres;

     /* und Bibliothekskennung schreiben */
     fres=Fwrite(thandle,2L,&typ);
     if(fres < 0 || fres != 2L)
          ferror("LIB.TMP",fres);

     do
     {
          /* Objektmodul-Kopf einlesen */
          fres=Fread(lhandle,28L,&kopf);
          if(fres < 0)
               ferror(argv[2],fres);

          objects=FALSE;

          if(fres == 28L && kopf.o_length > 0)
          {
               /* Vergleich mit dem gesuchten Modulnamen */
               in_klein(kopf.o_name);    /* erzwinge Kleinbuchstaben */
               gef=strcmp(&kopf,argv[4]);
               if(gef == 0)
               {
                    /* gefunden, erst Bibliotheksmodul-Kopf schreiben */
                    fres=Fwrite(thandle,28L,&kopf);
                    if(fres < 0 || fres != 28L)
                         ferror("LIB.TMP",fres);

                    /* Objektmodul kopieren */
                    transfer((long)kopf.o_length,
                              lhandle,thandle,argv[2],"LIB.TMP");

                    /* Objektfile öffnen */
                    typ=fileo(argv[3],&ohandle,&length);
                    if(typ == OBJ)
                    {
                         /* im Modulkopf Namen löschen */
                         for(x=0; x<=19; x++)
                              kopf.o_name[x]='\0';

                         /* und neuen Namen eintragen */
                         strcpy(&kopf,argv[3]);
                         kopf.o_length=length;

                         /* Modulkopf kopieren */
                         fres=Fwrite(thandle,28L,&kopf);
                         if(fres < 0 || fres != 28L)
                              ferror("LIB.TMP",fres);

                         /* Objektbeginn schreiben */
                         fres=Fwrite(thandle,2L,&typ);
                         if(fres < 0 || fres != 2L)
                              ferror("LIB.TMP",fres);

                         /* Objektmodul kopieren */
                         transfer((long)kopf.o_length-2,
                                   ohandle,thandle,argv[3],"LIB.TMP");

                         /* Objektfile schließen */
                         fres=Fclose(ohandle);
                         if(fres < 0)
                              ferror(argv[3],fres);
```

```
                    /* Vollzugsmeldung ausgeben */
                    Cconws("Modul ");
                    Cconws(argv[3]);
                    Cconws(" in Bibliothek ");
                    Cconws(argv[2]);
                    Cconws(" hinter Modul ");
                    Cconws(argv[4]);
                    Cconws(" eingefügt\n\r");
               }

               else
               {
                    /* Fehler, File ist kein Objektfile */
                    Cconws(argv[3]);
                    Cconws(" ist kein Objektfile\n\r");

                    /* angebliches Objektfile schließen */
                    fres=Fclose(ohandle);
                    if(fres < 0)
                         ferror(argv[3],fres);
               }
          }

          else
          {
               /* nicht gefunden, Modulkopf kopieren */
               fres=Fwrite(thandle,28L,&kopf);
               if(fres < 0 || fres != 28L)
                    ferror("LIB.TMP",fres);

               /* Objektmodul kopieren */
               transfer((long)kopf.o_length,
                         lhandle,thandle,argv[2],"LIB.TMP");
          }

          objects=TRUE;
     }
}
while(objects);

/* Bibliotheksende schreiben */
typ=END;
fres=Fwrite(thandle,2L,&typ);
if(fres < 0 || fres != 2L)
     ferror("LIB.TMP",fres);

/* Zwischenfile schließen */
fres=Fclose(thandle);
if(fres < 0)
     ferror("LIB.TMP",fres);

/* Bibliothek schließen */
fres=Fclose(lhandle);
if(fres < 0)
     ferror(argv[2],fres);

/* alte Bibliothek löschen */
fres=Fdelete(argv[2]);
if(fres < 0)
     ferror(argv[2],fres);

/* Zwischenfile wird neue Bibliothek */
fres=Frename(0,"LIB.TMP",argv[2]);
if(fres < 0)
     ferror("LIB.TMP",fres);
}

else
{
     /* Fehler, File ist keine Bibliothek */
     Cconws(argv[2]);
```

```
          Cconws(" ist keine Bibliothek\n\r");

          /* angebliche Bibliothek schließen */
          fres=Fclose(lhandle);
          if(fres < 0)
               ferror(argv[2],fres);
     }
}

/*********************************************************************/
/* view: zeige Bibliotheksinhalt an                                  */
/*********************************************************************/
view(argc,argv)
int  argc;     /* Anzahl Argumente in der Kommandozeile */
char *argv[];  /* Zeiger auf Argumentstrings */
{
     if(argc < 3)
     {
          /* zu wenig Argumente, Meldung */
          Cconws("Syntax für view: v Bibliothek\n\r");
          exit(0);
     }

     /* Bibliothek öffnen */
     typ=fileo(argv[2],&lhandle,&length);
     if(typ == LIB)
     {
          do
          {
               /* Objektmodul-Kopf einlesen */
               fres=Fread(lhandle,28L,&kopf);
               if(fres < 0)
                    ferror(argv[2],fres);

               objects=FALSE;

               /* Modulkopf analysieren */
               if(fres == 28L && kopf.o_length > 0)
               {
                    numout(kopf.o_length);   /* Objektlänge ausgeben */
                    Cconws(&kopf);           /* Objektnamen ausgeben */
                    Cconws("\n\r");

                    /* zum nächsten Objektmodul */
                    fres=Fseek((long)kopf.o_length,lhandle,1);
                    if(fres < 0)
                         ferror(argv[2],fres);

                    objects=TRUE;
               }
          }
          while(objects);

          /* Bibliothek schließen */
          fres=Fclose(lhandle);
          if(fres < 0)
               ferror(argv[2],fres);
     }

     else
     {
          /* Fehler, File ist keine Bibliothek */
          Cconws(argv[2]);
          Cconws(" ist keine Bibliothek\n\r");

          /* angebliche Bibliothek schließen */
          fres=Fclose(lhandle);
          if(fres < 0)
               ferror(argv[2],fres);
     }
}
```

```
/*********************************************************************/
/* extract: extrahiere ein Bibliotheksmodul                          */
/*********************************************************************/
extract(argc,argv)
int  argc;     /* Anzahl Argumente in der Kommandozeile */
char *argv[];  /* Zeiger auf Argumentstrings */
{
     if(argc < 4)
     {
          /* zu wenig Argumente, Meldung */
          Cconws("Syntax für extract: x Bibliothek Objektfile\n\r");
          exit(0);
     }

     /* Bibliothek öffnen */
     typ=fileo(argv[2],&lhandle,&length);
     if(typ == LIB)
     {
          do
          {
               /* Objektmodul-Kopf einlesen */
               fres=Fread(lhandle,28L,&kopf);
               if(fres < 0)
                    ferror(argv[2],fres);

               objects=FALSE;

               /* Modulkopf analysieren */
               if(fres == 28L && kopf.o_length > 0)
               {
                    /* vergleiche mit dem gesuchten Modulnamen */
                    in_klein(kopf.o_name);   /* erzwinge Kleinbuchstaben */
                    gef=strcmp(&kopf,argv[3]);
                    if(gef == 0)
                    {
                         /* gefunden, eröffne Objektfile */
                         fres=Fcreate(argv[3],0);
                         if(fres < 0)
                              ferror(argv[3],fres);
                         else
                              ohandle=fres;

                         /* Objektmodul kopieren */
                         transfer((long)kopf.o_length,
                                   lhandle,ohandle,argv[2],argv[3]);

                         /* und Objektfile schließen */
                         fres=Fclose(ohandle);
                         if(fres < 0)
                              ferror(argv[3],fres);

                         /* Vollzugsmeldung ausgeben */
                         Cconws("Objektfile ");
                         Cconws(argv[3]);
                         Cconws(" angelegt\n\r");
                    }

                    else
                    {

                         /* zum nächsten Objektmodul */
                         fres=Fseek((long)kopf.o_length,lhandle,1);
                         if(fres < 0)
                              ferror(argv[2],fres);
                    }
                    objects=TRUE;
               }
          }
          while(objects);

          /* Bibliothek schließen */
```

```
          fres=Fclose(lhandle);
          if(fres < 0)
               ferror(argv[2],fres);
     }

     else
     {
          /* Fehler, File ist keine Bibliothek */
          Cconws(argv[2]);
          Cconws(" ist keine Bibliothek\n\r");

          /* angebliche Bibliothek schließen */
          fres=Fclose(lhandle);
          if(fres < 0)
               ferror(argv[2],fres);
     }
}

/*********************************************************************/
/* Hauptprogramm                                                     */
/*********************************************************************/
main(argc,argv)
int  argc;     /* Anzahl Argumente in der Kommandozeile */
char *argv[];  /* Zeiger auf Argumentstrings */
{
     if(argc > 1)
     {
          switch(*argv[1])
          {
               /* erlaubte Kommandos ausführen */
               case 'd':
                    delete(argc,argv);
                    break;

               case 'i':
                    insert(argc,argv);
                    break;

               case 'v':
                    view(argc,argv);
                    break;

               case 'x':
                    extract(argc,argv);
                    break;

               default:
               /* kein erlaubtes Kommando, Syntax ausgeben */
               Cconws("Syntax für LIB:\n\r");
               Cconws("LIB divx Bibliothek [Objektfile/name] [Objektname]\n\r");
               break;
          }
     }

     else
     {
          /* kein Kommando, Syntax ausgeben */
          Cconws("Syntax für LIB:\n\r");
          Cconws("LIB divx Bibliothek [Objektfile/name] [Objektname]\n\r");
     }
}
```

Bücher

Viele Neuerscheinungen des Herbstes 1986 sind in anderen Zeitschriften noch gar nicht erwähnt worden. Zur Orientierung der Leser stellen wir die hier vorliegenden Titel daher etwas ausführlicher vor, um die Auswahl geeigneter Literatur zu erleichtern.

Berry, John Thomas: Inside the Amiga. Verlag Howard W. Sams & Co., Indianapolis 1986, ISBN 0-672-22486-2, 426 Seiten, $22,95. Dieses amerikanische Buch geht ausführlich auf die Betriebssysteme Intuition, AmigaDOS und auf den Kernel des Amiga ein. Es ist ein Leitfaden für die Programmierung durch den Anwender, wobei die Leistungen des Systems einbezogen werden, um die für Amiga typischen Anwenderoberflächen zu schaffen. Die zahlreichen voll ausgedrucken Programme sind in C formuliert und schreiten didaktisch von einfachen Aufgabestellungen zu hochentwickelten Leistungen fort. Es wird er Einsatz und Gebrauch des Lattice C-Compilers vorausgesetzt und genau beschrieben und welche der zu ihm gehörende Hilfdateien (die ebenfalls in C formuliert sind) mit INCLUDE jeweils einzubeziehen sind.

Es werden alle wesentlichen Leistungen, die das System bietet, mit Programmen belegt, wobei die Parameter in den C-Strukturen und ihre Wirkung im einzelnen erklärt werden. Die wichtigen abgehandelten Themen sind: Architektur der Hard- und Software, die Umgebung für den Programmierer, Fenster und Windows in Intuition, ferner Gadgets und Menüs, Dialogboxen, die Ein- und Ausgabe in Fenstern und das System der Message Ports (Kommunikation zwischen den Prozessen auf dem Rechner). Zu AmigaDOS wird das Multiprocessing beschrieben, der Umgang mit Dateien auf Disketten und das Starten eines Prozesses auf der Workbench. Etwa die Hälfte des Buches beschäftigt sich mit folgenden Themen, die für die Leistungsfähigkeit des Amiga typisch sind: Zeichnen mit Intuition, Sprites, Klangerzeugung und Erzeugung künstlicher Sprache. Zahlreiche Illustrationen begleiten dabei den verständlich und kompetent geschriebenen Text. - Wer das Innenleben des Amiga kennenlernen möchte, sollte sich dieses Buch beschaffen. Es steht in Aussicht, daß es im Frühjahr 1987 eine deutsche Übersetzung im te-wi Verlag geben wird.

Herold,Helmut; Unger,Werner: Das C-Buch. Verlag te-wi, München 1986, 576 Seiten Softcover, DM 79,-, ISBN 3-921803-62-4. Heft 43 enthielt eine Besprechung der bis Frühjahr 1985 erschienenen Bücher zu C. Im Besitze nun der Arbeit von Herold und Unger darf man wohl sagen, daß sie das beste bisher vorliegende Lehrbuch ist. Zunächst ist der ungeheure Fleiß zu erwähnen, der in die grafische Gestaltung des Inhaltes investiert wurde. Wichtige Stichworte sind auf jeder Seite fett hervorgehoben, ebenso die Zwischenüberschriften, wichtige Aussagen wurden in Kästen gestellt, weniger wichtige kleingedruckt. Zahlreiche Abbildungen erklären den Text, was insbesondere bei Zeiger-Variablen und Strukturen hilfreich ist. Insgesamt mögen über 100 lauffähige Musterprogramme im Buch enthalten sein, jeweils mit Aufgabenstellung, Struktogramm, kommentiertem Quelltext und Beispielen zum Programmablauf. So ergibt sich insgesamt ein gut überschaubarer und lesbarer Text. Die Aufgabenstellungen haben einen deutlich praxisnahen Bezug, so daß man auch die Bedeutung der Programmierung leicht versteht. - Die großen Themen des Buches sind Datentypen, Konstanten und Variablen, Ausdrücke und Operatoren, symbolische Konstanten, Zeiger, Ein-und Ausgabe, Verzweigungen, Schleifenkonstrukte, Datentyp-Umwandlungen, Funktionen und Programmstrukturen, Zeiger und Vektoren, Strukturen, Umgang mit Dateien, Beschreibung der wichtigsten Bibliotheksfunktionen. Die einzelnen Kapitel sind dabei noch vielfältig untergliedert. Im Abschnitt zu Dateien wird (wie an anderen Stellen auch) auf die Unterschiede bei den einzelnen Betriebssystemen eingegangen: MS-DOS, Unix, CP/M-86 und Isis-II. In der Sicht des Rezensenten sind die Abschnitte zu Operatoren, Zeigern, Strukturen und Unions sowie zu den Bibliotheksfunktionen besonders gut gelungen. - Das Lernen wird mit diesem Buch nie langweilig. Die hervorragende grafische Aufbereitung mit dem leichten Zurechtfinden macht es aber auch zum Nachschlagen geeignet.

Jamsa, Chris: Bibliothek der C-Routinen. Bei McGraw-Hill, Hamburg 1986, 316 Seiten, ca. DM 51,-, ISBN 3-89028-078-1. Das amerikanische Original dieses nunmehr übersetzten Buches wurde bereits in Heft 45 besprochen. Es enthält über 125 gut strukturierte und voll ausprogrammierte Beispiele (deren Quelltext ebenfalls ins Deutsche übertragen wurde), die man als Tools benutzen kann. Wichtige Themen sind Zeichenketten, Zeiger, E/A, Felder, Rekursion, Sortieren und Dateibearbeitung. Das Buch ist damit eine Anleitung, wie man zu Lösungen kommt.

Breuer, Markus: Das Amiga-Handbuch. Bei Markt & Technik, Haar 1986, 461 Seiten, DM 49,-, ISBN 3-89090-228-6. Dieses Handbuch gehört zumeist zum Lieferumfang, wenn man einen Amiga kauft. Es ist beschreibt vorwiegend die Hantierung des Computers und beginnt mit der Beschreibung der Hardware und der Fähigkeiten des Rechners. Kapitel 2 und 3 zeigen die Hantierung der Benutzeroberfläche Werkbank und beschreiben ihre Objekte. Der Autor geht weiter auf Anwenderprogramme (Werkzeuge), Projekte, Multitasking und Intuition ein. Kapitel 6 befaßt sich mit der Hantierung der Programme Grahicraft und Textcraft. Auch der zweiten Benutzeroberfläche wird viel Aufmerksamkeit gewidmet, dem CLI (Command Line Interpreter): Grundlagen, Dateien und Dateiverzeichnisse, Kommandos des CLI, Kommandofolgen, Text-Editor ed, Tips für das CLI. Schließlich werden die Spezialchips des Computers und ihre prinzipielle Arbeitsweise erläutert, besonders hinsichtlich Grafik und Klangerzeugung. Die Schnittstellen werden dokumentiert. Am Schluß finden sich zwei allgemeinere Artikel zur Programmierung des Amiga und zum BASIC. - Mit diesen Abschnitten gibt das Buch dem Anwender einen guten Überblick zur Leistungsfähigkeit des Systems und zum Zurechtfinden bei der Hantierung. Man sollte es jedoch auch zur Hand nehmen, wenn man die Anschaffung eines Amiga in Betracht zieht und ausreichende Orientierung wünscht.

Dr. Dobb's Journal: C Tools. Verlag Markt & Technik, Haar 1986, 690 S., DM 78,-, ISBN 3-89090-190-5. Die in den vergangenen Jahren im angesehenen Dr. Dobb's Journal veröffentlichten wichtigen Aufsätze zu C sind in diesem Buche zusammengefaßt, einige Abschnitte wurden für dieses Buch neu geschrieben. Der dargebotene Stoff deckt soviele Gebiete ab, daß hier nur wenige Einzelheiten erwähnt werden können. Zahlreiche Aufsätze befassen sich mit der Systemprogrammierung, mit der Implementierung von C und mit dem Aufbau von Bibliotheken. Dazu gehören folgende Kernabschnitte: Small-C Version 2 (Quelltext eines C-Compilers), Eine neue Bibliothek für Small C, Ein Assembler für Small C, Ein Präprozessor. Weiterhin werden folgende Tools vorgestellt: Getargs (Verarbeitung von Argumenten in dr Kommandozeile), Cross-Referenzen, cc-Kommandodatei, CP/M BIOS und BDOS-Aufrufe, Programme für die Textverarbeitung (dem Buch von Kernighan und Plauger folgend: Programmier-Werkzeuge), Grep (ein Parser nach Unix-Art). Weitere Kapitel befassen sich mit der Arbeitsweise von Compilern und der Optimierung von C-Strings. Interessant ist auch der 1980 erschienene Artikel von Ron Cain über seine Implementierung eines ersten C-Compilers, der dann offensichtlich das Werkzeug vieler anderer Autoren wurde. Viel praktisches findet man auch im "Notizbuch eines C-Programmierers". - Die Beurteilung des Buches fällt nicht leicht. Es ist einerseits Dokument eines zähen Ringens der Autoren um die Implementierung eines leistungsfähigen Sprachkompilers in einer von 8080 bis 8086 reichenden Maschinenumgebung mit allen Beschränkungen durch deren Architektur. Fast alle Module sind in C formuliert, auch die Quelltexte des Compilers, ein Zeichen dafür, daß die Compiler sich von Stufe zu Stufe 'hochgezogen' haben. - Als normaler Sterblicher wird man diese Anstrengung nicht nachvollziehen, wenn man C heute zu jedem aktuellen Personal Computer zu einem vernünftigen Preis erhält. Wer sich jedoch mit Compilerbau beschäftigt und den Innereien der Computersprachen

sowie mit deren Bibliotheken, der findet hier eine reiche Ernte und viele Anregungen. - Dem Übersetzer Peter Rosenbeck muß man für seinen klaren Ausdruck und für die eingestreuten Kommentare danken, besonders aber auch für die mit der Eindeutschung der Quelltexte verbundene Mühe.

Niemann, Peter: Commodore 128 Anwenderhandbuch. Bei McGraw-Hill, Hamburg 1986, 301 S. DM 40,-, ISBN 3-89028-046-3. Das Buch hat folgende Gliederung: Bedienung des 128, Die drei Betriebsarten C-64/128/CP/M, BASIC-Programmierung unter Einbeziehung der neuen Befehle in BASIC 7, Externe Geräte und Dateien, Grafik, Klänge und Spielsteuerungen, Maschinen-Schnittstelle und spezielle Diskettenbefehle und schließlich das Arbeiten mit dem CP/M-Betriebssystem. Es folgen 8 dokumentierende Anhänge und ein Stichwortverzeichnis. Das Buch enthält damit in einer erklärenden und zugleich dokumentierenden Art die Information, die man sonst in drei oder vier ziemlich nackten Handbüchern des Herstellers zusammensuchen muß, besonders, wenn man Anfänger ist. Es ist damit wirklich ein Handbuch für den Anwender. Hinter dieser Funktion und Absicht steht das Thema der Programmierung und ihrer Techniken zurück.

Lien, David A.: Amiga Programmier-Praxis mit MS BASIC. te-wi Verlag, München 1986, 394 Seiten, DM 59,-, ISBN 3-921803-69-1. Der amerikanische Professor legt ein weiteres frisch geschriebenes Buch vor. Wenn nicht etwa vier Kapitel hinzugefügt und der Name Amiga benutzt worden wären, so wäre dieses Buch die praktisch wortwörtliche Wiedergabe des in Heft 47 kurz besprochenen entsprechenden Buches für den MAC oder des in Heft 46 besprochenen Buches für den Commodore PC-10. Als Rezensent kritisiere ich weiterhin die verwirrende fette Hervorhebung belangloser Sprüche wie z.B. "Etwas langsamer, bitte!", "Zeit zum Verschnaufen", "Wieder zurück" und "Zu guter Letzt". Sie stehen auf den willkürlich aufgeschlagenen Seiten 72 und 73. Der nächste Spruch steht dann auf Seite 74: "Mit brutaler Gewalt". Weil da sonst nichts hervorgehoben ist, bemerke ich gar nicht, wovon eigentlich die Rede ist. - Die Beispiele des Buches sind jeweils kurz gehalten und zu einfach, um die Kunst der Programmierung in BASIC zu vermitteln. Das BASIC von Microsoft hat viele Befehle, die auf die besondere Leistungsfähigkeit des Amiga zugeschnitten sind und die insbesondere das grafische System des Betriebssystemes Intuition einbeziehen und nutzen. Lien hat viele Befehle einfach ausgelassen, obwohl sie auch für den Einsteiger von Interesse sein dürften. Diese Befehle verlangen oft zahlreiche Parameter. Wo sie kurz angesprochen sind, da vermißt man die vollständigen Parameterlisten für die alternativen Einstellungen.

Thies, Klaus-Dieter: Die Makroassembler ASM86 und ASM286. te-wi Verlag, München 1986, 312 S. DM 69,-, ISBN 3-921803-60-8. Nach den von ihm schon früher vorgelegten Büchern ist der Autor in Sachen "Maschine" ein alter Hase. Er teilt sein neues Buch in folgende Themengruppen: Die Architektur 8086/80186, Segmentierung aus der Sicht des ASM86-Programmierers, Definition und Initialisierung von Variablen, Adressierung von Variablen, Modulare Programmierung, Prozeduren, JMP- und CALL-Befehle, Kombination mit Modulen aus PL/M-86, Absoluter Code, Records, Makrosprache und Unterschiede zwischen ASM286 und ASM86. Der Text ist wiederum mit zahlreichen erklärenden Grafiken, eingerahmten Zusammenfassungen und ausgelisteten Beispielen in Assemblersprache versehen, die jeweils die Lösung typischer Aufgaben in kurzen Programmsequenzen enthalten. Der Text ist damit gut verständlich und dürfte dem Programmierer eines 8086 oder 80286 sehr hilfreich sein.

Jones, Edward: dBASE III Anwenderhandbuch. Bei McGraw-Hill, Hamburg 1986, 247 S., DM 43,-, ISBN 3-89028-044-7. Dieses aus dem Amerikanischen

übertragene Buch zeichnet sich durch knappe und klare Formulierungen aus, in denen die Möglichkeiten, die Syntax und die Programmierung dieses Datenbanksystemes auch mit zahlreichen Grafiken und Beispielen abgehandelt werden. Wir finden folgende Kapitel: Einführung, Datentypen, Einrichten einer Datei, Änderung, Sortieren und Indizieren, Berichte schreiben. Es folgen: Programmierung der dBASE III, Schleifen/Verzweigungen, Dateiverwaltung, Bildschirmausgaben, fortgeschrittenes Programmieren, Verbesserung von Programmen und Fehlersuche, Datenaustausch mit anderen Programmen, Beispielprogramme, Verbindung von dBASE II und III nebst Überführung, Hilfsprogramme zu dBASE III und ein Anhang mit der Syntax und Beschreibung der Befehle in dieser Datenbanksprache. Der Autor weist an zahlreichen Stellen auch darauf hin, wo Schwachpunkte des Systemes sind, interessant auch seine Darstellungen zum Werdegang der Datenbank. - Das Buch ist in erster Linie natürlich eine konzize Arbeitsunterlage. Wenn man jedoch an den enormen Arbeitsaufwand denkt, der mit der Dateneingabe oder mit der entsprechend formatierten Datenübernahme verbunden sind, die mit einem Datenbanksystem verbunden sind, so ist es auch ein Orientierungsbuch für diejenigen, die sich mit solchen Projekten tragen.

Bücher ohne detaillierte Besprechung:

Byers, T. J.: IBM PC AT. Bei McGraw-Hill, Hamburg 1986, 310 S. DM 48,-, ISBN 3-89028-070-6. Wesentlich abgehandelte Themen sind: Die neue Tastatur, DOS 3.0, Diskettenlaufwerke, Festplatte, Bildschirm und Grafik, Top View mit Multitasking, Schnittstellen, XENIX auf dem AT, IBM-Netzwerk, Kompatibilität, Netzteil/Uhr, BASIC 3.0.

Kelley, James E.: PC-Geheimnisse, IBM-Insider-Tips. Bei McGraw-Hill, Hamburg 1986, 245 S., Buch und Diskette, DM 69,80, ISBN 3-89028-072-2. Die Erfahrungen vieler Computeranwender wurden zusammengetragen. Themen: So arbeitet, die Hardware, Anpassung der Tastatur, Verbesserung der Bildschirmanzeige, Anpassung des Druckers, Mehr über DOS, PC von innen, Einsatz der Festplatte, Mehr über Bytes, Textverarbeitung mit Stil, Mehr über BASIC. Es sind etwa 120 Hilfsprogramme als Kochbuch beschrieben und auf der Diskette enthalten.

Kater, David A.; Kater, Richard L.:Ihr Epson-Drucker. Bei McGraw-Hill, Hamburg 1986, 216 S. DM 48,-, ISBN 3-89028-050-1. Der Schwerpunkt liegt bei den Matrix-Druckern, vom FX 85/105 bis hin zur LQ-Serie. Themen: Einführende Kapitel, Punktmatrixdruck, Druckerstandards, Zeichensätze, Variationen, Papiereinzug, Formulareinstellung und Druckkkopf, Nutzugsmöglichkeiten, Schalterstellungen, Grafik, eigene Zeichensätze, Textverarbeitung, Programme in BASIC, Kommunikation mit dem Drucker.

Schmuck, Anton: Programmiersprache PASCAL - Schritt für Schritt. Humboldt Taschenbuchverlag, München 1986, 159 Seiten, DM 8,80, ISBN 3-581-66551-4. Taschenbuch mit einer verständlichen Einführung in die Sprache.

Dahmke, Mark: Concurrent PC-DOS, Anwendungsmöglichkeiten. Bei McGraw-Hill, Hamburg 1986, 153 Seiten, DM 35,-, ISBN 3-89028-071-4. Darstellung des DRI-Betriebssystems für PC's.

Baras, Edward, M.: Symphony, Das Symphony-Handbuch. Bei McGraw-Hill, Hamburg 1986, 266 Seiten, DM 39,80, ISBN 3-89028-032-3. Anleitung für die Arbeit mit dem integrierten Programmpaket.

Baras, Edward, M.: Symphony Master, Symphony für Fortgeschrittene. Bei McGraw-Hill, Hamburg 1986, 357 Seiten, DM 59,-, ISBN 3-89028-042-0.

Makros, fortgeschrittene Spreadsheeet-Applikationen, Fenster, statistische Funktionen, Datenkommunikation.

Nicholas, Don; Quick, Judy: Bibliothek der Symphony Makros. Bei McGraw-Hill, Hamburg 1986, 137 Seiten, DM 48,-, ISBN 3-89028-049-8.

□○□

Roland Löhr

CBM-Banking

auf 6xx/7xx

Übersicht

Die genannten CBM-Rechner haben drei Jahre nach ihrer Markteinführung noch eine beachtliche Verbreitung gefunden, und zwar wegen des preiswerten Ausverkaufs-Angebotes der Firma Völkner. Wir nehmen das zum Anlaß, den neuen und noch hinzukommenden Betreibern Informationen zur Programmierung zu geben, die eine volle Nutzung der Systeme zulassen. Eine Kernfrage ist dabei die etwas komplizierte Bankumschaltung. Sie hat früher sicher viele potentielle Käufer zurückschrecken lassen, ist gleichwohl so schlimm auch wieder nicht:

Der CPU-Chip 6509 in den Commodore-Computern 6xx/7xx hat außer den üblichen Adreßbus-Pins A0...A15 vier weitere Pins, die der Adressierung dienen, nämlich P0...P3. Die Adressierungsarten der CPU und ihr internes Adreßregister lassen jedoch wie beim 6502 nur eine Ansprache von Befehlen und Operanden in einem 16 Bit-Adreßraum zu. Die vier zusätzlichen Pins P0...P3 werden daher nicht für die Befehls-Dekodierung benutzt, sie dienen vielmehr der Kulissenumschaltung, d.h. der Auswahl der mit hex 00...0f bezeichneten 16 Speicherbänke (von jeweils 64 KB) im Computer. Bank $0f ist dabei die Systembank mit den ROMs des Kernels, des BASIC 4 und mit den Interface-Bausteinen, mit dem Bildschirm-RAM und mit etwas RAM (2 KB) im Bereich $0002...$07ff. Je nach Computertyp und evtl. eigener Erweiterung sind dann die Bänke 1 und 2 bzw. auch 3 und 4 reine RAM-Bänke.

Die Computer sind darauf vorbereitet, am Stecker der Cartridge (auf der Rückseite) eine Zusatzplatine aufzunehmen, die RAM oder EPROM im Adreßbereich $0f2000-$0f7fff enthält. Die Systembank kann damit um 24 KB erweitert werden. Es gibt jedoch einen alternativen Umbau zu eigenen Zwecken: Wenn man die beiden BASIC-ROMs entfernt, meldet sich der Rechner beim Einschalten mit dem Kernel (dekodiert ab $0fe000) und ist von dessen Ebene aus voll hantierbar. Der dann bei $0f8000...$0fbfff freiwerdende Adreßraum kann für eigene Programme benutzt werden.

Die Ausführung von Programmen, die in der Systembank 15 stehen, fällt dem Programmierer am leichtesten, weil kaum sonstige Vorbereitungsarbeit zu leisten ist. Er schreibt und lädt seinen Code und ruft ihn von BASIC her mit dem Befehl SYS Adresse auf. Vom Monitorprogramm her sagt man für einen Start einfach auch nur g Adresse (g=GO). Auch das BASIC führt seine maschinensprachlichen Routinen nur in Bank 15 aus. Die RAM-Bänke werden als reine Parameter-Bänke benutzt. Bank 1 nimmt den Quelltext des BASIC auf (der von Bank 15 her interpretiert wird), die übrigen Bänke enthalten die einfachen Variablen und Arrays sowie die Texte der Funktionstasten. Auch das sind alles Parameter. - Es gibt zwar von BASIC her die Möglichkeit, ein Maschinenprogramm auch in einer anderen Bank aufzurufen, und zwar mit der Anweisungsfolge BANKx:SYS Adresse. Dieser Aufruf setzt jedoch voraus, daß in der Zielbank x systemgerechte Vorbereitungen getroffen worden sind, die den Darstellungen in Heft 33 entsprechen (CBM 710 Spezial-1).

P0...P3 und die CPU-Register e6509 und i6509

Zunächst ganz knapp gesagt: Die CPU enthält zwei Bank- oder Segmentregister,

die wie Portregister eines Einchippers angesprochen werden. Per Adresse $0000 erreicht man e6509, das Execution-Register, und per Adresse $0001 das Indirection-Register. Beide Register "scheinen durch jede Speicherbank hindurch", sind also immer erreichbar. Das Execution-Register e6509 bestimmt dabei, aus welcher Speicherbank Maschinenbefehle und die Operanden geholt werden mit der folgenden Ausnahme für Operanden: Das Register i6502 bestimmt, in welcher Bank die Operanden für die zwei Befehle LDA (Zeiger),y und STA (Zeiger),y liegen sollen. Außer diesen beiden Befehlen arbeiten keine weiteren indirekten Adressierungen mit dem i6509 zusammen. Das Indirection-Register ist damit ein durch die Bänke hindurchgreifender "Bagger" für Parameter. - Physikalisch: Was in e6509 steht (Werte zwischen $00 und $0f) wird auf den Pins P0...P3 ausgesandt, wenn Befehle geholt oder wenn Operanden angesprochen werden. Lediglich für die Befehle LDA (Zeiger),y und STA (Zeiger),y wird an den Pins P0...P3 ausgesandt, was im Register i6509 steht.

Die Bankumschaltung für Programme geschieht damit im Execution-Register e6509, und zwar typisch durch eine Befehlsfolge

```
   LDA #Zielbank
   STA e6509
TRENN xxx
```

Programmfortsetzung in einer anderen Bank
Nach Ausführung des Befehles STA e6509 hat der Programmzähler der CPU dabei den Wert von TRENN. Der nächste Befehl wird dann nahtlos bereits aus der Zielbank geholt, und zwar aus einer Adresse, die dem Wert von TRENN entpricht. Damit die CPU bei der Umschaltung nicht abstürzt, ist also eine Synchronisation des Maschinencodes in den benutzten Bänken notwendig. In der nachfolgenden Liste findet de Leser solche Synchronisationsstellen bei den Labeln, deren Namen mit "trenn" beginnen. - Im Kernel des CBM finden wir eine solche Synchronisationsstelle in Adresse $0ffff6. Dort steht: STA e6509. Der dahinter stehende Befehl RTS wird nicht mehr in der aufrufenden Bank ausgeführt. Er kann benutzt werden, wenn das Programm in der aufgerufenen Bank dieselbe Trennstelle zur Rückgabe benutzt. Ihr Gebrauch setzt allerdings Vorbereitungen in der Zielbank voraus.

Das Umschalten der Programmbank hat folgende Wirkungen: Das Programm wird mit dem laufenden Stand des Programmzählers und unveränderten Maschinenregistern in der Zielbank fortgesetzt. Die CPU schließt von sich aus keine weiteren Aktionen daran an. Zugleich sieht sie sich in einer Umgebung, in der ihre bisherige Zero Page, ihr bisheriger Stack und ihre bisherigen Daten nicht mehr in der Zielbank zur Verfügung stehen. Bei der Umschaltung in eine RAM-Bank verliert sie zugleich den direkten Zugriff auf das Ein- und Ausgabesystem und auf die ROM-Routinen, die es verwalten. Gewissermaßen schlimmer noch: Bei der Architektur der CPU 6509 "hagelt" der 60mal in der Sekunde eintretende Systeminterrupt in die jetzt eingestellte Programmbank (es sei denn, er ist mit dem Befehl SEI ausgeschaltet, was wiederum Systemeinschränkungen bedeuten kann, z.B. bei der Tastaturabfrage). Der Interrupt verlangt seine Vektoren in den obersten Zellen der benutzten Programmbank und seine Handler-Routinen, die normalerweise in Bank 15 stehen.

Die meisten Programme wollen vom Ein- und Ausgabesystem Gebrauch machen, das im Kernel der Bank 15 enthalten ist. Das möchte man nicht kunstvoll nachcodieren. Man möchte weiterhin die bekannten Einsprungssymbole in der Sprungleiste am Ende des F-ROMs benutzen können, so daß es eigentlich egal wird, für welche ausführende Bank man ein Assemblerprogramm schreibt. Das setzt erstens voraus, daß man auch in der Zielbank eine gleichwertige Sprungleiste hat. Da die Zielbank jedoch keine eigenen E/A-Routinen enthalten soll, muß zweitens ein Auffangsystem geschaffen werden, das gleichwohl die

erwünschten Leistungen erbringt, indem es eine Überleitung in die Bank 15 bewirkt und eine Rückführung an die aufrufende Programmstelle in der Anwenderbank.

Die Kernfrage des Bankings ist damit nicht die Abfolge der beiden oben genannten Befehle LDA #Zielbank und STA e6509, sondern es ist die noch recht einfache Frage, wie man eine Sprungleiste und die Vektoren in der Zielbank einrichtet und die etwas kompliziertere Frage, wie das Auffangsystem zu programmieren ist. Dabei muß man beachten, daß die benutzten Bänke eigene Stacks und Parameter haben und daß ein Datentransport zwischen Bänken nur mit den Befehlen LDA (Zeiger),y und STA (Zeiger),y stattfinden kann, wobei dann das Register i6509 eine Rolle spielt.

Ein Banking-Programm

Die nachfolgende Liste zeigt eine von mehreren Möglichkeiten, das Banking zu programmieren. Der Autor hofft, daß mit ihr der notwendige Mechanismus besonders anschaulich wird. Zunächst ist zu berichten, daß der Parameteraustausch zwischen Bänken in Briefkästen (Semaphoren) stattfindet, die in Bank 15 und (hier) in Bank 1 gleichsinnig in den Adressen $0100...0107 und $0108...010f liegen (Zeilen 18-32). Es wurden zwei Satz Briefkästen aus folgender Überlegung gewählt: Wenn im Anwenderprogramm eine E/A-Routine aufgerufen worden ist, dann erfolgt ein Parametertransport nach Bank 15. Wenn vor deren Abschluß und vor deren Bankumschaltung ein Interrupt noch in der Anwenderbank eintritt, dann kann dieser nicht auch noch die eben benutzten Briefkästen benutzen. (Man kommt mit einem Satz Briefkästen aus, wenn während der Überleitungsphase ein Interrupt ausgeschlossen wird.)

Zur Liste: Der hier benutzte CBM-Assembler kann erzeugten Code in jeweils zwei Bänken ablegen. Code unter Programmzähler 0fxxxx wird nach Bank 15 gebracht, solcher mit 01xxxx nach Bank 1. In den Zeilen 37-52 liegen die Trennstellen der Bank 15, in den Zeilen 136-151 sind die Trennstellen der Bank1. Ihre Adressen sind, wie man sieht, mit denen der jeweils anderen Bank synchronisiert.

In der Systembank 15 und in der Anwenderbank 1 wurden je drei Trennstellen (Übergabepunkte) programmiert: TRENNS1...3 und TRENNA1...3. Die jeweils ersten dienen dem Ein- und Ausstieg in die Anwenderbank, die zweiten der Interruptbearbeitung und die dritten den normalen E/A-Funktionen des Betriebssystems.

Zu den ersten Schnittstellen USERLINKF und USERLINK1: Das Programm wird in Adresse $0f76a gestartet (Zeile 96). Die erste Aufgabe besteht darin, eine geeignete Sprungleiste in die Zielbank zu bringen. Dazu wird der Befehl JSR BANKING 45mal dorthin kopiert, es folgen die nicht benutzten Befehle bei GBYE und die Interrupt- und Reset-Vektoren. Sie weisen auf die Routinen NMIROUT, USERLINK1 und IRQROUT. Es folgt der Sprung auf USERLINKF Zeile 37) mit Umschaltung in die Zielbank. Die Programmfortsetzung erfolgt ab der Stelle TRENNA1 in der Zielbank (Zeile 140). Im Weitersprung nach USERINIT werden die beiden Pointer APOINT und IPOINT für den Parametertransport eingerichtet. Danach kann das eigentliche Anwenderprogramm in Adresse $010800 begonnen werden (Zeile 262). Es wartet auf einen Tastendruck und gibt das empfangene Zeichen auf den Bildschirm aus. Es benutzt also in einer sehr gewohnten Weise Routinen des E/A-Systems - dieses spezielle Anwenderprogramm kann nur mit Reset verlassen werden.

Trennstellen bei OSLINKF und OSLINK1: Warum nun wurde die Sprungleiste 45mal mit JSR BANKING initialisiert? Dieser Befehl dient der Identifizierung, welche E/A-Routine des Kernels eigentlich zur Ausführung gelangen soll. Das JSR legt ja eine Rückkehradresse-1 auf dem Stack der Anwenderbank ab. Wenn wir in den Zeilen ab 215 auf dem Stack nachschauen und von dieser Adresse 2 subtrahieren,

dann wissen wir, welche Adresse in der Sprungleiste de Bank 15 benutzt werden soll. Sie wird nach SAV_RET für den Parametertransport übertragen. Wir halten uns vor Augen, daß auf dem Stack außerdem noch eine zweite Rückkehradresse liegt, nämlich die des Aufrufers im Anwenderprogramm. Sie wird später nach Ausführung der E/A-Routine benutzt. - Der Abschnitt BANKING bewirkt ab Zeile 222 den Transport der Parameter (vorwiegend gerettete Registerinhalte) in die Briefkästen der Systembank und führt mit JMP OSLINK1 weiter zur Trennstelle mit Bankumschaltung.

Die Programmfortsetzung erfolgt am Label TRENNS3 (in Zeile 52) in der Systembank. Weil Routinen des E/A normalerweise mit einem JSR aufgerufen werden, müssen wir vor dem Ansprung der Sprungleiste hinten im ROM eine Returnadresse auf den Systemstack bringen (nämlich von OSRETURN), die später eine geordnete Rückführung in die Anwenderbank gewährleistet. Die Register werden dann zurückgeladen und es erfolgt in Zeile 63 ein indirekter Sprung zur bereits berechneten Adresse der Betriebsroutine: JMP (SAV_RET). Nach deren Ausführung werden bei OSRETURN die Register in den Briefkästen gesichert, und es geht weiter zur Trennstelle OSLINKF mit Bankumschaltung.

Wir sind jetzt wieder in der Anwenderbank am Label TRENNA3 (Zeile 151). Die Parameter werden aus den Briefkästen der Systembank eingebaggert, die Register wieder hergestellt, der Stack bereinigt, und mit RTS kehrt man zu der Stelle im Anwenderprogramm zurück, die das Betriebssystem aufrief (genauer gesagt: zum Folgebefehl des JSR E/A). Die Welt ist nun wieder in Ordnung.

Die genannten Überleitungsfunktionen kann man noch etwas kürzen. Gleichwohl sieht man folgende Grundstruktur der Überleitung und Ausführung: 1.) Register der Anwenderbank retten, Funktionsaufruf erkennen, Parameter in die Briefkästen der Systembank bringen. 2.) In der Systembank die Rückführung vorbereiten, die gewünschte Routine im indirekten Sprung ausführen, Register in die Briefkästen retten, Umschalten in die Anwenderbank. 3.) Briefkästen entleeren, Stack berichtigen, Return zu aufrufenden Stelle.

Behandlung der Interrupts an den Trennstellen bei IRQLINKF und IRQLINK1: Ein in der Anwenderbank eintreffender Interrupt wird auf das Label IRQROUT in Zeile 231 bzw. auf NMIROUT vektoriert. Der weitere Ablauf ist ähnlich wie bei den E/A-Routinen: Der Interrupt wirkt etwas anders als ein Aufruf JSR. Auf dem Anwenderstack findet sich die Adresse des nächst auszuführenden Befehles (also nicht Adresse-1 !), außerdem der vor dem Interrupt geltende Status, ggfs. mit gesetztem Break-Flag. Je nach IRQ oder NMI wissen wir mit dem Ereignis schon definitiv, wohin in Bank 15 weiterzuspringen ist. Ab Zeile 231 sichern wir die Register und laden den Vektor. Es wird weiter auf das IRQLINK1 zugeleitet. Über TRENNS2 in Zeile 47 geht es im Sprung weiter nach Zeile 73. Es wird die Rückkehradresse von IRQRETURN auf den Stack gebracht. Ab Zeile 83 wird die IRQ-Routine in Bank 15 ausgeführt. Man kommt wieder an bei IRQRETURN, es folgt das Banking bei IRQLINKF. In der Anwenderbank geht es weiter bei TRENNA2 und schließlich in IRQREST (Zeile 172): Abschluß des IRQ mit dem Befehl RTI.

Die Anwendung des vorstehenden Mechanismus wird es erlauben, für sehr große Programme noch von einer ersten zu einer zweiten Anwenderbank weiterzuschalten. In Bank 15 muß dann nichts mehr hinzugefügt werden, die Übergabeprozedur ist allgemeingültig. Es werden jedoch synchrone Trennstellen zwischen z.B. Bank 1 und Bank 2 notwendig, an denen auch mitgeteilt wird, wo es in der eingeschalteten Bank weitergehen soll.

Aus der eigenen Beobachtung ist noch ein Hinweis zu geben: Wenn das Warten auf ein peripheres Ereignis in die Anwenderbank gelegt wird, wie z.B. das Warten auf einen Tastendruck mit JSR KGETIN, dann trifft auch der Systeminterrupt immer in der Anwenderbank ein. Die notwendigen Umladevorgänge bedingen eine kleine Verzögerung, die sich in einer geringfügigen Unruhe der

Bildschirmausgabe bemerkbar macht. Solche Wartevorgänge sollte man stattdessen mit einer eigenen Trennstelle versehen und in der Systembank ausführen, in die man gewissermaßen zum Warten "auftaucht".

```
0002 routinen zur bankumschaltung auf dem cbm 6xx/7xx
0003 */
0004                                ;symbolerklaerungen
0005 000000          e6509     =$0000          execution register der cpu
0006 000000          i6509     =$0001          indirection register der cpu
0007 000000          sysbank   =$0f            systembank des cbm
0008 000000          zielbank  =01             bank fuer das anwenderprogram
                                               m
0009 000000          apoint    =$ac            ram indirect pointer
0010 000000          ipoint    =$ae            pointer fuer irq
0011 000000          userstart =$0800          start des anwenderprogrammes
0012 000000          nnmi      =$fb31          nmi-routine des systems
0013 000000          nirq      =$fbd6          irq-routine des systems
0014
0015 000000                    .opt nof        assembliere ohne offset nach
                                               bank1
0016 000000                    *=$100
0017                           ;briefkaesten zur parameteruebergabe im stackbe
0017                           reich
0018 000100          asemaphor *=*+1           ablage des akkus etc. normale
                                                routinen
0019 000101          sav_x     *=*+1
0020 000102          sav_y     *=*+1
0021 000103          sav_ps    *=*+1           status
0022 000104          sav_sp    *=*+1           stackpointer
0023 000105          sav_bank  *=*+1           bank, von der man kommt
0024 000106          sav_ret   *=*+2           returnadresse
0025
0026 000108          isemaphor *=*+1           ablage des akkus etc. irq-rou
                                               tinen
0027 000109          isav_x    *=*+1
0028 00010a          isav_y    *=*+1
0029 00010b          isav_ps   *=*+1           status
0030 00010c          isav_sp   *=*+1           stackpointer
0031 00010d          isav_bank *=*+1           bank, von der man kommt
0032 00010e          isav_ret  *=*+2
0033 000110                    .opt ofs        assembliere mit offset nach b
                                               ank15
0034
0035 000110                    *=$0f0700       setze auf die systembank
0036                           ;definition von 3 programm-schnittstellen
0037 0f0700 a901     userlinkf lda #zielbank
0038 0f0702 ea                 nop             ausgleichsbyte
0039 0f0703 8501               sta i6509
0040 0f0705 8500               sta e6509       bankumschaltung beim start
0041 0f0707 00       trenns1   brk             break beim return ausfuehren
0042 0f0708 ea                 nop             fueller
0043 0f0709 ea                 nop
0044 0f070a ad0d01   irqlinkf  lda isav_bank    gib irq an aufrufer zurueck
0045 0f070d 8501               sta i6509
0046 0f070f 8500               sta e6509       zurueck vom irq in bank15 nac
                                               h bank1
0047 0f0711 4c4307   trenns2   jmp irqhandl     irq-handler in bank15
0048
0049 0f0714 ad0501   oslinkf   lda sav_bank     gib nach os-routine an aufruf
                                               er zurueck
```

```
0050 0f0717 8501            sta i6509
0051 0f0719 8500            sta e6509           zurueck von bank15
0052 0f071b         trenns3                     ;handler in bank15 f. sprungl
                                                eiste
0053 0f071b a907            lda #>osreturn      baue returnadresse auf
0054 0f071d 48              pha                 high
0055 0f071e a931            lda #<osreturn-1
0056 0f0720 48              pha                 low address
0057 0f0721 ad0301          lda sav_ps          alter status, register ueberg
                                                eben
0058 0f0724 48              pha
0059 0f0725 ad0001          lda asemaphor
0060 0f0728 ae0101          ldx sav_x
0061 0f072b ac0201          ldy sav_y
0062 0f072e 28              plp
0063 0f072f 6c0601  oslinkz jmp (sav_ret)       routine anwender ausfuehren
0064                        ;osreturn ist wiederansprungspunkt
0065 0f0732 08      osreturn php                rette status
0066 0f0733 8d0001          sta asemaphor       register zur uebergabe retten
0067 0f0736 8e0101          stx sav_x
0068 0f0739 8c0201          sty sav_y
0069 0f073c 68              pla                 geretteter status
0070 0f073d 8d0301          sta sav_ps
0071 0f0740 4c1407          jmp oslinkf         bankumschaltung
0072
0073 0f0743         irqhandl                    ;handler in bank15 fuer irq
0074 0f0743 a907            lda #>irqreturn     baue returnadresse auf
0075 0f0745 48              pha                 high
0076 0f0746 a959            lda #<irqreturn
0077 0f0748 48              pha                 low address
0078 0f0749 ad0b01          lda isav_ps         alter status, register ueberg
                                                eben
0079 0f074c 48              pha
0080 0f074d ad0801          lda isemaphor
0081 0f0750 ae0901          ldx isav_x
0082 0f0753 ac0a01          ldy isav_y
0083 0f0756 6c0e01          jmp (isav_ret)      interrupt-routine ausfuehren
0084
0085 0f0759 08      irqreturn php               rette status
0086 0f075a 8d0801          sta isemaphor       register zur uebergabe retten
0087 0f075d 8e0901          stx isav_x
0088 0f0760 8c0a01          sty isav_y
0089 0f0763 68              pla                 geretteter status
0090 0f0764 8d0b01          sta isav_ps
0091 0f0767 4c0a07          jmp irqlinkf        bankumschaltung
0092
0093                        ;dieser programmteil initialisiert die notwendi
0093                        gen informationen
0094                        ;in der bank des anwenders, hier bank1
0095
0096 0f076a a96f    startinf lda #$6f           unterer beginn der sprungleis
                                                te
0097 0f076c 85ac            sta apoint
0098 0f076e a9ff            lda #$ff            adresse high
0099 0f0770 85ad            sta apoint+1        zielpointer fertig
0100 0f0772 a000            ldy #0              zunaechst sprungleiste in die
                                                 zielbank kopieren
0101 0f0774 a901            lda #zielbank
0102 0f0776 8501            sta i6509
```

```
0103 0f0778 a200      trloop   ldx #0              innenschleife
0104 0f077a bd9707    trloop1  lda transtab,x      den befehl jsr banking
0105 0f077d 91ac               sta (apoint),y     45x einkopieren
0106 0f077f c8                 iny                aussenschleife
0107 0f0780 e8                 inx
0108 0f0781 e003               cpx #3
0109 0f0783 90f5               bcc trloop1        innenschleife
0110 0f0785 c087               cpy #$87           menge erfuellt?
0111 0f0787 90ef               bcc trloop         nein
0112 0f0789 bd9707    trloop2  lda transtab,x      nun den rest der parameter
0113 0f078c 91ac               sta (apoint),y     ab gbye
0114 0f078e c8                 iny                aussenschleife
0115 0f078f e8                 inx
0116 0f0790 c091               cpy #$87+10        plus 10 parameter
0117 0f0792 90f5               bcc trloop2
0118 0f0794 4c0007             jmp userlinkf      beginn programmausfuehrung
0119
0120                           ; die nachfolgenden befehle werden hinten in da
0120                           s
0121                           ;programmsegment des anwenders einkopiert
0122 0f0797 208107    transtab jsr banking        befehle zum einkopieren in di
                                                  e zielbank
0123                           ;good bye
0124 0f079a 8500      gbye     sta e6509          execution reg
0125 0f079c 60                 rts
0126 0f079d ea                 nop
0127                           ;hardware-vektoren
0128 0f079e df07               .wor nmirout       nmiroutine in anwenderbank
0129 0f07a0 0007               .wor userlink1     reset wird break
0130 0f07a2 b607               .wor irqrout       irqroutine in anwenderbank
0131
0132 0f07a4                    .opt nof           assembliere ohne offset nach
                                                  bank1
0133 0f07a4                    *=$010700          bank des anwenderprogrammes
0134
0135                           ;definition von 3 programm-schnittstellen
0136 010700 a90f      userlink1 lda #sysbank
0137 010702 ea                 nop                ausgleichsbyte
0138 010703 8501               sta i6509
0139 010705 8500               sta e6509          bankumschaltung bei programme
                                                  nde
0140 010707 4c6407    trenna1  jmp userinit       initialisiere anwenderprogram
                                                  m
0141 01070a a90f      irqlink1 lda #sysbank       bearbeite irq in bank15
0142 01070c ea                 nop                ausleichsbyte
0143 01070d 8501               sta i6509
0144 01070f 8500               sta e6509          irq in bank15 ausfuehren
0145 010711 4c4307    trenna2  jmp irqrest        schliesse irq in bank1 ab
0146
0147 010714 a90f      oslink1  lda #sysbank       fuehre routine lt. sprungleis
                                                  te aus
0148 010716 ea                 nop
0149 010717 8501               sta i6509
0150 010719 8500               sta e6509          benutze sprungleiste in bank1
                                                  5
0151 01071b a004      trenna3  ldy #4             parametertransport, wieder in
                                                   bank1
0152 01071d a90f               lda #sysbank       systembank ansteuern
0153 01071f 8501               sta i6509          daten
```

```
0154 010721 b1ac      tloopr   lda (apoint),y
0155 010723 990001             sta asemaphor,y    bankdurchgriff
0156 010726 88                 dey
0157 010727 10f8               bpl tloopr
0158 010729 ad0501             lda sav_bank       anwenderbank
0159 01072c 8501               sta i6509          datenbank umschalten
0160 01072e ae0401             ldx sav_sp         stackpointer und register zur
                                                  ueck
0161 010731 9a                 txs                to stackpointer
0162 010732 68                 pla                jsr von sprungleiste aufheben
0163 010733 68                 pla
0164 010734 ad0301             lda sav_ps
0165 010737 48                 pha
0166 010738 ad0001    irqy     lda asemaphor
0167 01073b ae0101             ldx sav_x
0168 01073e ac0201             ldy sav_y
0169 010741 28                 plp                alter status
0170 010742 60                 rts                abschluss
0171
0172 010743 a004      irqrest  ldy #4             parametertransport, abschluss
                                                   des irq
0173 010745 a90f               lda #sysbank       systembank ansteuern
0174 010747 8501               sta i6509          daten
0175 010749 b1ae      itloopr  lda (ipoint),y
0176 01074b 990801             sta isemaphor,y    bankdurchgriff
0177 01074e 88                 dey
0178 01074f 10f8               bpl itloopr
0179 010751 ad0d01             lda isav_bank      anwenderbank
0180 010754 8501               sta i6509          datenbank umschalten
0181 010756 ae0c01             ldx isav sp        stackpointer und register zur
                                                  ueck
0182 010759 9a                 txs                to stackpointer
0183 01075a ad0801             lda isemaphor
0184 01075d ae0901             ldx isav_x
0185 010760 ac0a01             ldy isav_y
0186 010763 40        irqz     rti
0187
0188                           ;initialisierung in der anwenderbank und weiter
0188                           prung
0189                           ;zum eigentlichen programmstart
0190 010764 78        userinit sei                register retten, irq abschalt
                                                  en
0191 010765 a900               lda #<asemaphor    pointer einrichten
0192 010767 85ac               sta apoint
0193 010769 a901               lda #>asemaphor
0194 01076b 85ad               sta apoint+1
0195 01076d a908               lda #<isemaphor    pointer einrichten
0196 01076f 85ae               sta ipoint
0197 010771 a901               lda #>isemaphor
0198 010773 85af               sta ipoint+1
0199 010775 a501               lda i6509          merke die aktive bank
0200 010777 8d0501             sta sav_bank
0201 01077a 8d0d01             sta isav_bank
0202 01077d 58                 cli                interrupt wieder zulassen
0203 01077e 4c0008             jmp userstart      normal nach $800
0204
0205 010781 08        banking  php                 rette status, interrupt absch
                                                  alten
0206 010782 78                 sei                solange wir noch in der anwen
                                                  derbank sind
```

```
0207 010783 8d0001            sta asemaphor
0208 010786 8e0101            stx sav_x
0209 010789 8c0201            sty sav_y
0210 01078c 68                pla                 alter status
0211 01078d 8d0301            sta sav_ps
0212 010790 ba                tsx                 stackpointer
0213 010791 8e0401            stx sav_sp
0214 010794 38                sec                 returnadresse-2 bestimmen
0215 010795 bd0101            lda $0101,x         low, um den aufruf
0216 010798 e902              sbc #2              zu erkennen
0217 01079a 8d0601            sta sav ret
0218 01079d bd0201            lda $0102,x
0219 0107a0 e900              sbc #0
0220 0107a2 8d0701            sta sav_ret+1       das sind jetzt 8 items
0221
0222 0107a5 a007      banking1 ldy #7              parametertransport
0223 0107a7 a90f               lda #sysbank        systembank ansteuern
0224 0107a9 8501               sta i6509           daten
0225 0107ab b90001    tloop    lda asemaphor,y
0226 0107ae 91ac               sta (apoint),y      bankdurchgriff
0227 0107b0 88                 dey
0228 0107b1 10f8               bpl tloop
0229 0107b3 4c1407             jmp oslink1         bankumschaltung
0230
0231 0107b6 20e907    irqrout  jsr rettung         irq-routine
0232 0107b9 a9d6               lda #<nirq
0233 0107bb a0fb               ldy #>nirq          vektoriere interruptberabeitu
                                                   ng
0234 0107bd 8d0e01    irqrout1 sta isav_ret
0235 0107c0 8c0f01             sty isav_ret+1
0236 0107c3 68                 pla                 status mit interruptflag
0237 0107c4 48                 pha
0238 0107c5 0904               ora #$04            i-flag dazu
0239 0107c7 8d0b01             sta isav_ps
0240 0107ca ba                 tsx                 stackpointer
0241 0107cb 8e0c01             stx isav_sp
0242 0107ce a007               ldy #7              parametertransport
0243 0107d0 a90f               lda #sysbank        systembank ansteuern
0244 0107d2 8501               sta i6509           daten
0245 0107d4 b90801    itloop   lda isemaphor,y
0246 0107d7 91ae               sta (ipoint),y      bankdurchgriff
0247 0107d9 88                 dey
0248 0107da 10f8               bpl itloop
0249 0107dc 4c0a07             jmp irqlink1        bankumschaltung
0250
0251 0107df 20e907    nmirout  jsr rettung         nmi-routine
0252 0107e2 a931               lda #<nnmi
0253 0107e4 a0fb               ldy #>nnmi
0254 0107e6 4cbd07             jmp irqrout1
0255
0256 0107e9 8d0801    rettung  sta isemaphor       interrupt-routine in anwender
                                                   bank
0257 0107ec 8e0901             stx isav_x
0258 0107ef 8c0a01             sty isav_y
0259 0107f2 60                 rts
0260
0261 0107f3                    *=$010800           ausfuehrendes programm start
0262 010800 20e4ff    astart   jsr kgetin
0263 010803 20d2ff             jsr kbsout
0264 010806 4c0008             jmp astart
```

Roland Löhr

Datenbanken

H&DBASE - Superbase

Zu den immer häufiger benutzten Anwendungen auf den Personal Computern gehören außer leistungsfähigen Text- und Kalkulationsprogrammen jetzt auch Datenbanken. Von solchen Anwenderprogrammen ist zu fordern, daß sie in den Betrieben in der Hand von Laien problemlos laufen und daß sie pflegefreundlich sind. Es zeigt sich immer wieder, daß aus den Erfahrungen nach der Inbetriebnahme erweiterte oder veränderte Wünsche abgeleitet werden. Bei Datenbanken ist die Datenerfassung, der Aufbau des Informationsbestandes, oft eine Aufgabe von Wochen und Monaten. Hier ist besonders zu fordern, daß schon erledigte Arbeit durch eine später evtl. notwendige Umstellung nicht verloren geht.

Der Autor möchte seine Beobachtungen aus der Arbeit mit zwei Datenbanksystemen hier zur Orientierung der Leser mitteilen.

Datenbanken sollen gespeicherte Information schnell wiederaufschlagbar machen. Jeder kennt das einleitende Gespräch bei der Krankenkasse (Angabe des Namens und des Geburtsdatums) oder beim Energieversorgungsunternehmen (Angabe der Kundennummer oder, falls unbekannt, der Straße und Hausnummer). Mit diesen Basisangaben kann ein kompletter Datensatz zur Anzeige gebracht werden, der weitere benötigte Information enthält. Datenbanken erlauben daneben das Vorwärts- und Rückwärtsblättern in Datensätzen, wenn es z.B. in der Krankenkasse mehrere Datensätze mit gleichem Namen und gleichem Geburtsdatum geben sollte.

Datenbankprogramme sind damit einerseits natürlich Ein- und Ausgabeprogramme für die Erfassung und Wiedergabe von Datensätzen. Insofern sind sie Textprogrammen verwandt, in denen meistens auch eine Suche von Textketten ermöglicht ist (Instring-Suche). Datenbankprogramme sollen aber vor allem eine gezielte Suche in einem oder mehreren Datenfeldern schnellstmöglich erlauben. Ein reines Textprogramm wird bei der Suche nach dem String "6900" nicht nur Kundennummern mit den enthaltenen Zeichen "6900" aufschlagen, sondern möglicherwese auch Telefonnummern und Postleitzahlen in "6900 Heidelberg". Die gezielte Suche in Feldern ist also ein wesentlicher Gesichtspunkt.

Nun kan man weitere Anforderungen stellen: Zu suchen ist "Meier in Heidelberg" (Postleitzahl fällt im Moment nicht ein). Möglicherweise schreibt sich Herr Meier auch als "Meyer" oder auch mit einem "a", also z.B. "Mayer". In diesem Falle soll auch eine Suche mit Stellvertreter-Zeichen ("§") möglich sein. Man suche im Namensfeld nach "M§§er" und natürlich auch nach "Heidelberg". Das wäre eine logische UND-Verknüpfung. Oder Herr M§§er sagt, daß er nicht mit dem dem "M§§er" in Heidelberg vervechselt werden dürfe. In diesen Falle ist eine NOT-Verknüpfung vorzusehen. Oder man sucht in Feldern, deren Inhalt kleiner oder größer als eine Vorgabe ist, z.B. Rechnungsdatum kleiner 01.01.87.

Diese ähnlich auftauchenden Fragen zeigen besser als eine Theorie-Darlegung, daß Datenbankprogramme sich wechselnden Abfrage-Pfaden stellen müssen. Außer der Suche nach "identisch" sollte auch eine Suche nach "ähnlich" mit logisch UND, ODER oder NICHT möglich sein. Und die Ordnung der Datensätze in Suchfeldern wie z.B. Name, Geburtsdatum und Ort spielt auch eine große Rolle. Datenbankprogramme sind vor allem also Suchprogramme.

Die Anordnung von Information - Masken

Man möchte aufgeschlagene Datensätze in einer strukturierten Form auf dem Bildschirm sehen, den Daten der einzelnen Felder sollen die Feldbezeichnungen zur Orientierung des Betrachters vorangestellt sein, also z.B. "Name: Meier" usw. Die Forderung nach Einbettung in eine Maske gilt nicht nur für für die Wiedergabe vorhandener Information, sondern auch für die erstmalige Dateneingabe und für die Datenpflege bei Veränderungen. Bei den letztgenannten Arbeiten möchte man auch erkennen können, welche Schreibbreite für die einzelnen Felder zugelassen ist.

Zur Anordnung der Information bedient man sich also der sog. Masken. Sie enthalten zunächst einmal die Rahmeninformation mit den Feldbezeichnungen. Sie kennzeichnen weiterhin, an welchen Stellen und in welcher Breite Eintragungen möglich sind. Text kann also nicht an beliebiger Stelle des Bildschirmes eingegeben werden, sondern nur an den Schreibstellen der Felder. Zum Komfort bei der Eingabe und bei der Datenpflege gehört es weiterhin, daß man z.B. mit den Cursortasten von Feld zu Feld schnell vorwärts und rückwärts springen kann. Auch die Taste Return wird sinnvoll zum Weiterspringen benutzt.

Zur Erzeugung der Maske gehören aber auch Regieanweisungen für die Bedienung, daß z.B. Felder zwangsweise ausgefüllt sein müssen (wie der Hauptsuchbegriff, Schlüssel) oder daß Felder alternativ ausgefüllt sein müssen: Wenn kein Eintrag für Straße, dann aber Eintrag für Postfach erforderlich und umgekehrt. Und es werden die Eigenschaften der Felder festgelegt: Schlüsselfeld, Textfeld, numerisches Feld, mit dessen Inhalt gerechnet werden kann, und ggfs. Ergebnisfeld, das den Saldo numerischer Felder z.B. in der Buchhaltung aufnimmt.

Datenbanksprachen

Es ist dem Anwender natürlich unbenommen, seine Datenbank und die Maskenerzeugung selbst zu programmieren, sei es nun in Assembler oder in einer höheren Sprache. Im allgemeinen wird er auf ein fertiges Datenbankprogramm für seinen Computer zurückgreifen und dabei feststellen, daß es da nicht nur unterschiedliche Leistungsmerkmale gibt, sondern daß man die speziellen Anwendungen mit unterschiedlichem Komfort noch unter Benutzung von Anweisungen einer Datenbanksprache programmieren muß. Im Vergleich gehen wir hier besonders auf dBASE und auf Superbase ein.

dBASE nebst Abkömmlingen gehört zu den auf IBM-PCs und kompatiblen Rechnern sehr verbreiteten Datenbankprogrammen, besonders jetzt in der Version dBASE III. Eine Implementierung ist z.B. auch Practibase auf den Multitech-Rechnern. Der Autor benutzte auf dem Atari ST die Version H&DBASE mit einem Befehlsumfang entsprechend dBASE II. Es ist für den Atari ist FORTH geschrieben worden und hat gleichwohl ein Programmfile von etwa 200 KB, in dem auch die Sprache FORTH aktivierbar ist. dBASE ist nicht nur Programm, sondern auch eine Datenbanksprache, die in zahlreichen Büchern dokumentiert ist, s.a. die Buchbesprechung in Heft 45.

Superbase ist ein englisches Programm von Precision Software. Es wurde zunächst für Commodore-Rechner der Linien 610-720 angeboten (in dieser Form wurde es hier zum Einsatz gebracht). Man kann es auch für den PC-128 beziehen, dem Vernehmen nach mit erweiterter Leistungsfähigkeit. Bei Drucklegung dieses Heftes gibt es möglicherweise schon eine Freigabe für Commodore AMIGA. Angekündigt ist das Programm auch für Atari ST. - Der Autor verhehlt nicht, daß er Superbase sogar auf dem 8-Bit CBM gegenüber dBASE bevorzugt. Superbase ist mit allen Grundleistungen einer Datenbank bedienbar, ohne daß man eine

einzige Zeile Programm schreiben muß. Gleichwohl kann man auch in Superbase programmieren, um den Anwender in einer eigenen Befehlsebene zu halten oder um gezielte Verarbeitungen vorzunehmen. Dabei kann man auf weite Strecken das BASIC des CBM mit seinen Stringfunktionen u.a.m.benutzen sowie etwa 50 zusätzliche Datenbank-Befehle.

Start mit H&DBASE auf Atari

Für einen Gewerbebetrieb, in dem Laien am Computer arbeiten, wurden inzwischen mehrere Datenbanken implementiert. Da der Erstanwender eines PC seine Wünsche häufig erst im Laufe der Zeit immer deutlicher formuliert, wurde zunächst ein Atari ST1040 mit günstigem Preis-/Leistungsverhältnis zum Einsatz gebracht. Es wurde anfangs in H&DBASE programmiert, und zwar zunächst für die Funktionen Maske aufbauen, laufende Eingabe von Datensätzen (Kundendatei), Korrigieren, Aufschlagen und Blättern in Datensätzen, Suchen in beliebigen Feldern und Zählen. Es stand kein bildschirm-orientierter Maskengenerator zur Verfügung. Aus diesem Grunde waren die Feldbezeichungen der Maske durch häufigen Gebrauch des Befehles §x,y SAY "text" auf den Schirm zu schreiben (x=Zeile, y=Spalte). Das Einholen der Eingaben aus den Feldern erfolgte mit Befehlen wie §x,y GET Variable.

Die wenigen Funktionen füllten bereits eine Programm-Liste von 5 Seiten. Es erwies sich als lästig, daß man je Programmzeile nur einen Befehl schreiben kann. dBASE ist durchaus eine strukturierte Sprache. Bedauerlich gleichwohl, daß man CASE (Fallentscheidungen) nicht ineinander schachteln kann. Bei der Programmierung flossen durchaus Tränen, denn man wurde schon bei geringfügigen Fehlern absolut aus dem Programm herausgeworfen, ohne einen Hinweis zu erhalten, woran es denn gelegen hatte. Und es tauchte häufig die Fehlermeldung auf: System Error (Bus). Das System war damit meistens nicht mehr weiter hantiertbar. Es gibt die Vermutung, daß die Systemprogrammierer z.B. bei Wortzugriffen auf eine ungerade Adresse aufsetzen, was beim 68000 bekanntlich zu einer Exception führt. Der Autor war weiterhin mit der Hantierung des doch sehr einfachen Editors wenig einverstanden, den man zum Schreiben von Kommandodateien braucht. In vielen Fällen war es übersichtlicher, diese mit 1rst Word zu schreiben, umzuladen und sie dann in der Datenbank zu erproben.

Es kam der Einsatz beim Kunden. Die Benutzerführung klappte. Es fiel auf, daß das Kopieren von etwa 12 vorläufigen Variablen der Eingabe in die endgültigen Feldvariablen einige Sekunden dauerte. Nach etwa 100 eingegebenen Datensätzen wurde das System auffällig langsamer, es ergaben sich Wartezeiten in der Größenordnung von schon einer Minute, ehe man den folgenden Satz eingeben konnte. Aus Gründen der Sicherheit (Stromausfall etc.) wurde dabei jeder Datensatz sofort auf Floppy abgespeichert. Mit einer RAM-Floppy hätte man natürlich bessere Reaktionszeiten erzielt, allerdings mit dem zusätzlichen Risiko, daß einmal das Absichern der geleisteten Arbeit vergessen wird. Risiken dieser Art muß man vermeiden.

Nach einem Arbeitstag und einem halben stürzte das System dann endgültig ab. Es waren keinerlei Daten mehr zu retten. In dieser Stunde fiel die Entscheidung, mit H&DBASE weiter nicht mehr zu arbeiten.

Aus dem Vorgang läßt sich für eigene Planungen folgendes ableiten: Datensicherheit muß in einem Gewerbebetrieb Vorrang haben. Dazu gehört natürlich auch das Ziehen von Sicherheitskopien (Backup). Zweitens: Die Datenerfassung darf den Bediener nicht bei der Arbeit behindern. Dem Autor ist früher schon häufiger berichtet worden, daß einige Datenbanksysteme mit dem Einbau weiterer Information immer langsamer werden. Ein wichtiger Gesichtspunkt ist daneben die Reaktionszeit bei Datenbankabfragen. Es geht

nicht an, daß man lange warten muß, ehe man den richtigen Datensatz aufgeschlagen hat. Für die Datei- und Programmpflege ist ferner die Anpassungsfähigkeit von großer Bedeutung.

An dieser Stelle noch einige Anmerkungen zur Programmierung in dBASE-Systemen: Der obenstehende Bericht betrifft eine spezielle Implementierung. Es ist in jedem Fall gut, wenn ein Maskengenerator zur Verfügung steht, mit dem man interaktiv am Bildschirm bestimmen kann, welche Texte wo erscheinen und wo und in welcher Breite und Art Datenfelder/Eingabefelder sein sollen. Wenn ein solcher Generator nicht benutzt werden kann, ist man gezwungen, eine recht umfangreiche Kommandodatei zu schreiben, für die nachstehend ein kleiner Programmausschnitt folgt. Mit § Zeile/Spalte SAY "Textkette" werden Texte auf den Bildschirm plaziert. Typisch folgen Befehle wie § Zeile/Spalte GET Variable und auch READ. GET (in verschiedenen Spielarten) bringt den Inhalt der Variablen an der spezifizierten Stelle zur Anzeige. Bei der erstmaligen Dateneingabe wird man meistens ohne einen Vortrag aus dem vorhergehenden Datensatz arbeiten wollen. Der Inhalt der Variablen wird daher vor GET in voller Feldbreite auf einen String zu setzen sein, der zwischen den Hochkommata nur Zwischenraumzeichen enthält. READ setzt dann den Cursor auf das Feld und holt die Eingabe.

Bei der Anlage einer Datenbank hat man in einem unabhängigen Vorgang zunächst mit CREATE den Namen, den Typ und die Breite der Datenfelder festzulegen, ehe man in der Maske mit GET und READ arbeiten kann. - Die Suche in einer Datenbank erfolgt mit den Befehlsworten LOCATE oder FIND in ihren Spielarten, für die ebenfalls eine Befehlsfolge kurz abgedruckt wird.

Abfragen an den Bediener erfolgen typisch in Programmschleifen. Weil dBASE dabei von Strukturen Gebrauch macht, ist man oft gezwungen, ähnliche Konstrukte wiederholt einzutippen, wenn man nicht die PROCEDURE von dBASE III hat. Einen Ausweg gibt es allerdings mit DO Kommandodatei. Eine Befehlsdatei kann andere mit diesem Befehl als Unterprogramm aufrufen um z.B. die Maske auf den Schirm zu schreiben. Das Schreiben strukturierter Befehlsdateien wird dadurch vereinfacht. Wenn sich aber ein solches Unterprogramm auf Diskette befindet, so treten gewisse Wartezeiten ein.

Aus diesen Hinweisen ist für den Systemprogrammierer zu entnehmen: Datenbanken vom Typ dBASE arbeiten mit Befehlsdateien, die schnell eine längere Liste bilden und für die man einige Tipparbeit investieren muß. Der Berichterstatter meint daher, daß sie für den durchschnittlichen Anwender zu schwierig sind, wenn er maßgeschneiderte Lösungen anstrebt. Für Standard-Lösungen kann man ihm natürlich eine Umgebung verschaffen, in der er seine Daten in einer befriedigenden Weise herein- und auch wieder herausbekommt.

Befehlsfolgen in dBASE II zur Dateneingabe

```
* Maske schreiben
@ 0,6  SAY 'Kunden-Nr.'
@ 1,6  SAY 'Anrede Nr.'
@ 1,39 SAY 'Anrede'
@ 3,10 SAY 'Name 1'
@ 4,10 SAY 'Name 2'
@ 6,10 SAY 'Strae'
@ 7,8  SAY 'Postfach'
@ 9,9  SAY 'PLZ Ort'
@ 11,7 SAY 'Kundenart'
? revers
* Kunden-Nummer eingeben
DO WHILE e:kdnr = '     '
     @ 0,17 GET e:kdnr
     READ
ENDDO
```

```
* Sehen, ob es die Nummer schon gibt:
STORE T TO nrloop
DO WHILE nrloop
     STORE '0'+e:kdnr+'0' TO suchnr
     FIND &suchnr
     IF # <>0
          @ 20,0 SAY 'Die Kunden-Nummer '+e:kdnr+'  gibt es schon, bitte Neu-E
          STORE '     ' TO e:kdnr
          @ 0,17 GET e:kdnr
          READ
     ELSE
          STORE F TO nrloop
     ENDIF
ENDDO
@ 20,0 SAY 'Anredeschluessel: 1=RA-, 2=PA-Buero, 3=Firma, sonst. Texteingabe'
* Merkmal der Anrede eingeben
DO WHILE e:merkmal = ' '
@ 1,17 GET e:merkmal
READ
ENDDO
* Entschluesselung des Merkmales Anrede
DO CASE
     CASE e:merkmal='1'
          STORE 'Anwaltsbuero' TO e:anrede
          @ 1,47 SAY e:anrede
     CASE e:merkmal='2'
          STORE 'Anwaltsbuero' TO e:anrede
          @ 1,47 SAY e:anrede
     CASE e:merkmal='3'
          STORE 'Firma' TO e:anrede
          @ 1,47 SAY e:anrede
     OTHERWISE
          DO WHILE e:anrede='
               @ 1,50 GET e:anrede
               READ
          ENDDO
ENDCASE
* Zeile loeschen
@ 20,0 SAY chr(27)+chr(75)
* Namen 1 zwingend eingeben
DO WHILE e:name1='
     @ 3,17 GET e:name1
     READ
ENDDO
```

Superbase 700

Im Sommer standen für den Atari noch keine alternativen Datenbankprogramme zur Verfügung, über deren Bewährung in der Praxis man schon etwas gewußt hätte. Sofort greifbar waren jedoch ein CBM 710 (8 Bit), eine Floppy SFD1001 dazu und vor allem das Programm Superbase mit etwa 180-seitiger Dokumentation. Die Eingabe einer Maske geschieht hier am Bildschirm innerhalb der Funktion Format. Diese Arbeit inklusive Verschönerung des Aussehens war bei der erstmaligen ernsthaften Inbetriebnahme des Programmes in gut zwei Stunden erledigt. Es wurden zwar Wochen später noch zusätzliche Felder in der Maske angelegt (was möglich ist), es konnte jedoch ab sofort mit der Datenerfassung und -pflege zuverlässig fortgefahren werden, die nach wie vor unter Superbase auf dem CBM erfolgt. Der Atari konnte im weiteren Verlauf in Assemblerprogrammierung als sehr schnelles Suchsystem implementiert werden.

Nach den positiven Erfahrungen mit Superbase möchte ich näher auf dieses Datenbanksystem eingehen, zumal es jetzt auch für Amiga und Atari ST angekündigt ist und dort hoffentlich die Erwartungen gleichermaßen erfüllt. Die Maskenerstellung erfolgt also komfortabel am Bildschirm (full screen editor). Feldnamen werden normal als Text getippt. Datenfelder werden mit Escape-Sequenzen definiert und bei der Formatierung symbolisch angezeigt. Es gibt Schlüsselfelder, Textfelder, numerische Felder, Konstanten-Felder, Ergebnis-, Datums- und Kalenderfelder. Felder können als forciert erklärt werden (sie müssen immer einen Eintrag haben).

Eine zusammengehörige Maske kann bis zu 4 Bildschirmseiten füllen und bis zu 127 Felder haben. - Innerhalb einer Datenbank können bis zu 15 Dateien enthalten sein. Dabei ist es mit Link-Befehlen möglich, Daten aus anderen Dateien zu entnehmen. Die Zahl der Datensätze je Datei ist logisch unbeschränkt. Mit dem Aufbau einer Datei und ihrer automatischen Indizierung (alphabetische Ordnung nach bis zu 30 Zeichen) tritt keinerlei Verlangsamung beim Abspeichern ein. Der Rechner ist immer wieder nach 1 Sekunde aufnahmebereit. Ebenso schlägt er nach Schüssel gesuchte Datensätze in etwa 1 Sekunde von der Floppy her auf. Beim Blättern zum folgenden oder vorhergehenden Satz ist er noch schneller.

Es ist normal, daß das Format einer Datei einmal geändert werden soll, weil man bei ihrer Anlage nicht alles bedachte. In Superbase darf man ungestraft die Feldbreiten reformatieren, solange man bei einer Kürzung nicht in vorhandenen Datensätzen enthaltene Nutzinformation abhackt. Felder können am Schluß hinzugefügt oder fortgelassen werden, ohne daß Daten verloren gehen. Wenn diese Möglichkeiten des Reformatierens noch nicht ausreichen, dann kann man mit dem Export- oder dem Output-Befehl eine sequentielle Datei als Abzug der Datenbank erzeugen, in der auch die Feldinhalte in eine neue Abfolge gesetzt, verändert oder mit weiterer Information durchsetzt werden. Der Befehl Import gestattet es, eine solche sequentielle Datei wieder zu einer Datenbank zusammenzusetzen. Für 2000 Datensätze dauert ein solcher Import mit Verkettung etwa 1 Stunde. Export und Import bieten weiterhin die Möglichkeit, mit den Informationen der einen Datenbank eine zweite zu gewinnen. Wenn man in der ersten z.B. die Kundennummer als Schlüsselfeld führt, so können in der Tochterdatei z.B. die Namen als Schlüssel zum schnellen Aufschlagen benutzt werden.

Damit sind die Fragen des Suchens angesprochen. Ein Datensatz wird am schnellsten dann aufgeschlagen, wenn man seinen Schlüssel kennt. Das Suchen in Superbase ist jedoch nicht auf das Schlüsselfeld beschränkt. Man kann in jedem beliebigen Feld suchen und zugleich in mehreren Feldern als verknüpfte Suchbedingung. Man kann also nach "Meier" und zugleich "Heidelberg" und zugleich "männlich" suchen oder nach "Meier" in "Heidelberg" oder möglicherweise auch in "Mannheim". Oder in der Buchhaltung läßt man sich alle Kunden aufschlagen, deren Rechnung 4 Wochen alt ist und deren Betrag größer als DM 50 ist. Ein zusätzliches Merkmal wäre "für den Bezirk des Reisenden Meier".

Für solche einfachen oder zusammengesetzten Absuchungen bedient man sich des Befehles Select Match. Als Bediener erhält man dann eine leere Maske auf dem Bildschirm angezeigt, in die man in die zutreffenden Felder hineinschreibt, wonach identisch, ähnlich oder mit Oder-Bedingung zu suchen ist. Nach dem Auslösen der Funktion sucht Superbase dann etwa 220 Datensätze (bei Adressen) je Minute auf der Floppy und prüft sie auf die Merkmale. - Der Vorgang läßt sich natürlich natürlich durch ein Programm automatisieren, wobei entweder

eine Liste mit den Schlüsseln geschrieben oder gleich Etiketten, Mahnungen etc. auf dem Drucker erzeugt werden.

Dieses Suchsystem ist zwar gut und zuverlässig und mag für die Verwaltung eines kleinen Vereines oder einer Schallplattensammlung ausreichen. Im professionellen Einsatz sind 9 Minuten Wartezeit (ungünstigster Fall) bei etwa 2000 Adressen zuviel Zeitaufwand, zumal das Risiko besteht, daß man die Suchvorgabe falsch wählte "der Kunde schreibt sich ja gar nicht so!". Dann muß der Suchlauf mit verbesserten Vorgaben wiederholt werden. Bei größeren Datenbeständen muß man dann in Superbase lieber doch Dateien parallel führen, die nach den verschiedenen häufig vorkommenden Suchgesichtspunkten geordnet sind. Das "Umgießen" einer größeren Datei per Export und Import in eine anders verschlüsselte mit dem o.e. Zeitaufwand von 1 std. ist dabei eine überlegenswerte Aktion. Der Autor ist sicher, daß man mit Rumpfdateien, die nur wenige Felder der Zuordnung haben (Name zu Kundennummer oder Geburtsdatum zu Name) schnellere Umformungen errreicht. Es kommt ja nur auf die Stichworte an, wo man sinnvollerweise weitersuchen sollte. Und schließlich kann man Verbindungen (Links) zu solchen Rumpfdateien aufbauen, die eine schnelle Orientierung des Benutzers ermöglichen, wie er die Suche zeitsparend fortsetzt.

Ein schnelles Suchen kann aber auch auf anderem Weg erreicht werden, der etwas unkonventionell ist: Für den CBM 720 und den PC-128 wurden in den Heften 45 und 47 Text-Editoren veröffentlicht, die den sehr schnellen Suchalgorithmus nach Boyer und Moore enthalten: Jeder Instring in 64 KB Speicher wird in schneller als 1 Sek. gefunden. Wenn man also per Export- oder Outputbefehl eine sequentielle Datei abzieht, so kann man diese Suche auf dem gleichen Rechner erheblich beschleunigen, wenn man die Daten speicherresident hält. Wir gingen noch einen Schritt weiter: Der Atari mit seinem großen RAMspeicher erhält per Schnittstelle RS232 einen seqentiellen Abzug von Superbase und speichert ihn auf seine eigene Floppy. Die Daten stehen ihm weiterhin zur Verfügung, er kann sie am folgenden Tag jedoch wieder von der Floppy in 20 Sekunden laden. Mit dem genannten Suchalgorithmus finden wir auf dem Atari nun jeden Instring (auch mit enthaltenen Stellvertreter-Zeichen) im ungünstigsten Fall blitzartig in 1 Sekunde (2000 Adressen, ca. 260 KB).

Das Gespann Superbase auf Commodore, Schnittstelle RS232 und schnelles Suchen in der RAM-residenten Datei des Atari ist sicher ungewöhnlich. Nach Ansicht des Autors kommt es vor allem im professionellen Einsatz auf Leistung und Schnelligkeit an. Information muß schnell aufschlagbar sein. Wenn für den einen Computer die Dateneingabe und Datenpflege zuverlässig gelöst ist, dann soll man ihn wegen des vorhandenen Programmes dafür einsetzen. Wenn der andere Computer eine wichtige Dienstleistung z.B. im genannten Zeitverhältnis von 9 Minuten zu 1 Sekunde besser ausführen kann (Faktor 540), dann sollte man solche Gespanne benutzen. Wir leiden derzeit doch darunter, daß wir zwar gute Hardware haben, jedoch nur wenige Programme, die die ihre spezifische Leistungsfähigkeit voll ausnutzen können.

Die weitere Beschreibung von Superbase wird etwas kürzer gefaßt, weil es zwar auf Qualitäten ankommt, nicht aber so sehr auf den Aspekt der Geschwindigkeit.

Die wesentlichen Funktionen der Anpassungsfähigkeit in der Formatierung und der Zuweisung von Eigenschaften zu Datenfeldern wurden bereits genannt. Bei Ergebnisfeldern ist erwähnenswert, daß man ihnen ein Berechnungsschema mit einer Formel wie unter BASIC zuweisen kann. Eine wichtige Funktion ist auch SORT: Man kann ein Ausgabefile erzeugen, das auf die Merkmale eines beliebigen Datenfeldes auf- oder absteigend sortiert wird. Das Ausgabefile enthält dann die aus der Sortierordnung entstehende Schlüsselfolge. Dieser Vorgang dauert

für 2000 Adressen auch nur 1/2 Stunde und ist damit im Aufwand vertretbar.

Mit CALC, REPORT und BATCH kann man Berechnungen vornehmem, Berichte mit kalkulierten Ergebnissen schreiben oder Verarbeitungen im Job vornehmen. Weitere wichtige Funktionen sind HELP und MEMO. Mit HELP kann man dem Benutzer Tafeln mit Instruktionen zur Benutzerfühung zeigen. Mit MEMO schreibt man sie, ggfs. als auch eine Ergänzung zu Datensätzen, denen man einen Hinweis hinzufügen möchte. MAINTAIN zeigt als weitere Befehlsebene im Status die Eigenschaften und die Breite der einzelnen Datenfelder. Zugleich ist aber auch der Zugriff auf alle Floppy-Befehle des CBM gesichert.

Eine weitere wichtige Leistungssteigerung von Superbase liegt in der Option PROG. Man kann Befehle im Tischrechnermodus per Kommandozeile eingeben und ausführen. Aber man kann auch wie in BASIC ein Programm zur Ausführung mit numerierten Zeilennummern schreiben, das mit EXECUTE Zeilen-Nr. zur Ausführung gebracht wird. Alle wichtigen Befehle des BASIC stehen hier zur Verfügung, einschließlich der Stringfunktionen, auf Datenfelder ausgeübt und des Instrings unter BASIC 4-7. Mit PROG kann man den Anwender in eine Befehlsebene des Systemprogrammierers führen, in der seine Eingaben auf Zulässigkeit geprüft werden. Ebenfalls kann man Suchfunktionen mit Dateiverbindung und Abfragen zur Kontrolle implementieren. Die Möglichkeiten sind ziemlich unbegrenzt (bis 8 KB Quelltext unter PROG, Variablen- und Stringmanipulation).

Zusammenfassung

Superbase hat hervorragende Eigenschaften bei der Datenerfassung und -pflege. Programmentgleisungen wurden nicht festgestellt. Es kann alles umgeformt und wiedergefunden werden. Masken dürfen für eine bereits vorhandene Datei verändert werden, ohne daß Schaden entsteht. Es ist auch eine komlizierte Suche möglich. Man erhält bei optimierten Algorithmen bessere Suchfunktionen in Assemblerprogrmmierung, wenn man nicht nach dem Schlüssel aufschlägt und wenn man große RAM-Speicher benutzen kann.

□○□

Editorial

Am Jahreswechsel ein kurzer Rück- und Ausblick: Das Jahr 1986 mit seinen zahlreichen Umweltkatastrophen und Tschernobyl dürfte weiten Kreisen bewußt gemacht haben, das das Wirtschaften der Menschen diesen blauen Planeten zunehmend mit Giften belastet. Zu den Belastungen gehört auch der Verkehrslärm aus der Luft, von der Straße und von den Bahnen. Die Vermeidung von Müll, Luft- und Gewässerverschmutzung und von Lärm und des Entstehens giftiger und radioaktiver Abfallstoffe werden zunehmend volkswirtschaftliche Kosten verursachen. Im Bereich der Politik sieht man wenig Entschlossenheit, langfristige Konzepte anzufassen, die die Belastungen zurückführen würden. Das betrifft auch die Zusammenarbeit in Europa in Sachen Umweltschutz bei tatsächlichen oder möglichen Beeinträchtigungen, die die nationalen Grenzen überschreiten. So wird man immer wieder mit bösen Überraschungen rechnen müssen, die nach heutiger Technik vermeidbar wären. Vorsorge werden dabei nur die reicheren Nationen treffen können. Schon bei den Nachbarn im Osten und erst recht in den Entwicklungsländern fehlt es an Geld, um die Umwelt ausreichend zu schützen. - Gleichwohl werden Computer in der Überwachung der Bedingungen eine segensreiche Rolle spielen können und auch spielen.

Als mit Computern befaßter Zeitgenosse muß man jedoch der amerikanischen strategischen Verteidigungsinitiative SDI mit großer Besorgnis begegnen. Bisher haben Menschen am Abzugshahn der Nuklearwaffen gesessen. Und man weiß, daß das computergestützte System der NORAD in den USA oft dringenden Alarm gibt, der sich später als Irrtum, als gegenstandlos herausstellt. Beim SDI werden Computer mit nuklear angetriebenen Laserwaffen im Weltraum postiert, wo besondere Strahlungs- und Temperaturbedingungen herrschen. Wir wissen jedoch, daß schon hier auf Erden Computer gelegentlich für Überraschungen sorgen, wenn Induktionen eintreten. Einem Fehlverhalten sehen wir jedoch nicht an, ob es möglicherweise durch Strahlung verursacht wurde, weil das Ereignis nicht reproduzierbar ist. - Mit immer feineren Transistorstrukturen hängt es künftig nur noch von ganz wenigen Dotierungsatomen ab, ob ein Chip funktionsfähig ist. Damit dürften Schaltkreise m.E. auch empfindlicher werden. Es muß daher bezweifelt werden, daß im Weltraum stationierte Computersysteme so sicher gemacht

werden können, daß nicht ein Fehlverhalten eintritt, das die Katastrophe nach sich zieht. Man kann nur hoffen, daß der amerikanische Congress und die Executive den vernünftigeren Weg der kontrollierten Abrüstung beschreiten.

Im Ausblick: 1986 wurde der Computer mit 1 MB RAM besonders durch Atari eine Selbverständlichkeit. Nach den Ankündigungen darf man davon ausgehen, daß es in einigen Monaten bei etwa gleichem Preis Computer geben wird, die 4 MB RAM haben oder dahin aufrüstbar sind. Die Industrie macht weltweit große Investitionen, um noch höher integrierte RAMs zu entwickeln und zu produzieren, wobei die jeweils nächste Generation um den Faktor vier leistungsfähiger ist. Und auch bei den Festwertspeichern nimmt die Integration zu, Festplatten werden leistungsfähiger und preiswerter. - Von Atari wird für die zweite Hälfte 1987 ein Zusatzmodul mit dem Prozessor 68020 erwartet (Bezeichnung TT) für den Anschluß mehrerer Bildschirme und Benutzer, möglicherweise mit UNIX. Von einem solchen System war bereits im Oktober 1985 in München die Rede. Es braucht also auch in der Hardware einige Zeit, bis die neuen Möglichkeiten angefaßt werden. Das betrifft auch die Verwendung der sehr leistungsfähigen Video-/Grafikchips und sicher auch den neu vorgestellten sehr leistungsfähigen 68030 von Motorola.

Es bedarf des großen Nachfragevolumens eines Herstellers wie Apple, Atari, Commodore usw., daß ein Chip wirklich in die Stückzahlen geht und damit billig wird. - Neue Zentraleinheiten werden heute bereits für Taktfrequenzen ab 24 MHz ausgelegt. Von der Technik her dürfen wir damit davon ausgehen, daß wir in etwa drei Jahren Computer vorfinden, die gegenüber den heute gebräuchlichen PC's eine etwa 10mal größere Leistung haben werden. Der derzeit noch verbreitete IBM-PC und seine Klones werden damit weiter in die Preisklasse der Homecomputer zurückfallen.

Man muß weiterhin davon ausgehen, daß die Softwareindustrie möglichst den bequemen und billigeren Weg geht, der in der Übertragung lauffähiger Betriebssysteme in die nächste Hardware-Klasse besteht, ohne Ausnutzung der spezifischen neuen Leistungsmerkmale. Ein Computer wird damit in der Konkurrenz der Anbieter schneller präsentierbar. Über 1 MB mehr oder weniger für Betriebssoftware wird man sich dabei nicht viele Gedanken machen. Wie auch bei früheren Gelegenheiten wird es damit hier in der alten Welt viele Ansatzpunkte geben, die neue Leistungsfähigkeit der Hardware auch in der Software auszunutzen. Wir sind allerdings schon jetzt an einer Stelle, wo man zum neuen Computer bereits eine Datenbank braucht, um ihn leistungsgerecht programmieren zu können.

Der Herausgeber wünscht den Lesern einen guten Start im neuen Jahr und hofft auf weiterhin gute Zusammenarbeit. Verbunden damit ist die Bitte, den Gedankenaustausch wachzuhalten und gelegentlich zu Themen zu berichten, die auch für andere Computerbetreiber interessant sein können.

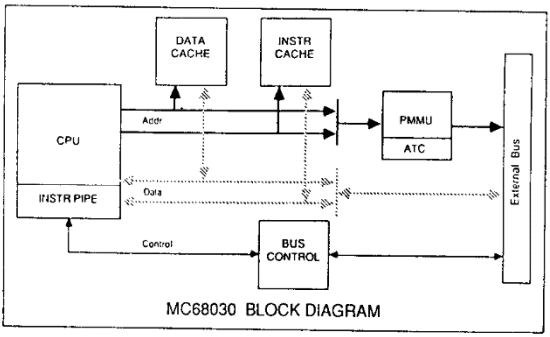
□○

Aus der Branche

Motorola: 32-Bit Prozessor 68030 angekündigt. Zur Bemusterung im 3. Quartal 1987 und zur Lieferung ab 4. Quart. 87 wurde jetzt der neue Microprozessor 68030 angekündigt. Es soll zunächst mit 16,67 MHz betrieben und im 128-poligen Gehäuse Pin Grid Array geliefert werden. Seine wesentlichen Merkmale sind: Harvard-Architektur mit je zwei unabhängigen 32 Bit Adreß- und Datenbussen für Programme und Daten, 2 Cache-Speicher auf dem Chip für Programme und Daten, integrierte Speicherverwaltungseinheit auf dem Chip: Paged Memory Management Unit. Das Design baut auf den Funktionseinheiten des 68020 auf, den man bereits 1987 in verschiedenen Personal Computern vorfinden wird. Neben einem zum 68000 aufwärtscompatiblen Befehlssatz, beschleunigter Multiplikation und Division sowie der Bearbeitung von Bitfeldern zeichnet sich der 68020 bereits durch je einen Daten- und Adreßbus mit 32 Bit aus sowie durch einen Befehls-Cache von 256 Bytes auf dem Chip. Da Programme sehr häufig in Befehlsschleifen verharren, führt dieser Cache bereits zu einer erheblichen Entlastung des externen Busses, weil benötigte Befehle bereits in die CPU geladen sind. Die Einrichtung eines zweiten Caches für Daten führt beim 68030 zu einer weiteren Busentlastung. Die unabhängigen Adreß- und Datenbusse erlauben eine Parallelverarbeitung. Zusätzlich zum bisherigen asynchronen Bus wird eine synchrone Busschnittstelle eingerichtet, die zyklusweise konfiguriert wird. An der synchronen Schnittstelle kann ein schnelles Cache-Subsystem angeschlossen werden, das mit statischen RAMs arbeitet, an die asynchrone Schnittstelle ein langsameres DRAM-System. Auf der synchronen Schnittstelle hat man dann eine minimale Zugriffszeit von 2 Taktzyklen. Cache-Speicher können im Burst Fill–Modus beschleunigt geladen werden. - Auch die auf den Chip integrierte Speicherverwaltungseinheit ist mit den beiden Caches verbunden. Durch die Möglichkeit der parallelen Zugriffe ergibt sich

über alles eine interne Busbandbreite von über 80 MByte/Sek, wenn Daten und Befehle aus den Cache-Speichern und parallel auch aus den externen Speichern abgerufen werden. Insgesamt soll sich damit eine Verarbeitungsleistung erheben, die mit einer VAX 8600 Schritt halten kann, bei einem angenommenen Preis für ein Büroautomatisierungssystem von etwa 2-3000 Dollar.

Zur integrierten PMMU wird angegeben, daß sie einen eigenen chipinternen Cache für die Adreßumsetzung hat. Das interne Pipelining mache es möglich, daß sich die Buszyklen durch die Adreßumsetzung nicht verlängern. Für besonders zeitkritische Anwendungen soll es die Möglichkeit der sog. transparenten Speicherfenster geben. Unter Umgehung der PMMU kann die 68030 dabei bestimmte logische Adreßbereiche direkt auf den physikalischen Bereich abbilden. Die Größe dieser Speicherfenster kann zwischen 16 MByte und 4 GByte betragen, womit z.B. in grafischen Systemen ausreichend Platz im schnellen Zugriff liegt. - Angesagt ist auch ein verbesserter Koprozessor 68882 für Fließpunktaritmetik.



MC68030 BLOCK DIAGRAM

Doppel-UART 65C52 von Rockwell International. Dieser neue Chip enthält für die serielle Datenübertragung zwei von einander unabhängige Schnittstellen mit jeweils eigenen programmierbaren Registersätzen zur Steuerung der Übertragungsparameter. Mit diesem 40-poligen Chip lassen sich Schnittstellen kostengünstiger und platzsparender aufbauen. Mit beiden Schnittstellen kann gleichzeitig mit verschiedenen Übertragungsraten gesendet und empfangen werden. Baudraten von 50 bis 38400, Wortlängen 5, 6, 7 oder 8 Bit, programmierbare Stop-Bits und Parity.

Kölner CBM-Selbsthilfe 600. Späte Erwerber eines 6xx/7xx von Commodore bemühen sich um die Beschaffung und Verbreitung von Information zu den Computern, Dokumentationen werden fotokopiert, eine kleine Zeitschrift ist in Vorbereitung. Kontakt: Gerd Schumacher, Palanterstr. 12 d, 5000 Köln 41, Tel.: 0221-44 83 68.

VMEbus Design Kit von VALVO. Zur Unterstützung von Hardware-Entwicklungen wird ein Kit zu DM 275 mit folgendem Inhalt angeboten: VMEbus-Spezifikation als Dokumentaion (Rev. C.1/IEEE P1014, Draft 1.2), Datenblätter und Applikationsartikel, die den Einstieg in die VMEbus-Entwicklung wesentlich vereinfachen, ferner 6 Chips: Je eine synchrone/synchrone Interface-Schaltung, Interruptgenerator und ein Handler, CPU 68000 mit 10 MHz und ein DUART.

68020-Board von Plessey. Unter der Bezeichnung PME 68-22 wird ein Platinencomputer mit CPU 68020 und VMEbus-Interface angeboten. Takt bis 20 MHz, Co-Prozessoroption 68881, Paged Memory Management Unit 68851, Dualported DRAM in 2 oder 8 MB-Option, SCSI-Interface, 2x RS232, Timer RTC usw.

Dokumentation zum Commodore Amiga im Verlag Addison-Wesley. Die ursprünglich mit dem Entwicklungspaket beziehbaren Handbücher zum Rechner, geschrieben von Entwicklungsmitarbeitern bei Commodore, sind jetzt unabhängig als Buch beziehbar: Hardware Reference Manual (272 S., DM 62,50), ROM Kernel Reference Manual (544 S., DM 88), Intuition Reference Manual (363 S., DM 62,50) und Exec Reference Manual (176 S., DM 62,50).

Platinen-Layouts im Maßstab 2:1 kostengünstig für Kleinserien werden jetzt von der Fa. Uwe Tams in 24 Kiel 1, Westring 273 angeboten, Tel.: 0431-180975. Es kommt das Programm Platine ST auf dem Atari zum Einsatz im Verbund mit dem Drucker star NL-10. Neben dem Layout oben und unten werden erzeugt: Bestückungsplan, Bauteil- und Verbindungsliste.

Inhaltsverzeichnis des MICRO MAG
für die Hefte 41 - 48

□○□

Kleinanzeige

Suche GWK-Floppy Disk-Controller-Karte für AIM 65. Rudolf Boller, Flörsheimer Str. 13, 6094 Bischofsheim, Tel. 06144-87 10.

Neue Assembler

Neu konzipierte Assembler für 6502, R65C02, 65802/65816
für AIM 65, CBM 610/710/720 und jetzt auch PC-128

Deutsche interaktive Bedienerführung, Fehlerhinweise in deutschem Klartext. Einstellbare Befehlssätze für die drei CPU-Typen mit Fehlerhinweis bei nicht implementierten Befehlen/Adressierungen. Schnelldurchgang mit alten Parametern möglich. Pass 2 kann mit neuen Parametern für die Ausgabe zeitsparend wiederholt werden. Symboltafel alphabetisch geordnet. Symbole dürfen 9 Zeichen lang sein, dadurch aussagekräftige Label möglich. Listbilder in diesem Heft. Eingabe von speicherresidentem Quelltext oder von der Commodore-Floppy. Im Texteditor kann ein Kommandotext benutzt werden, der beliebig viele Quelltext-Module von der Floppy aufruft und sie auch mit residentem zusätzlichen Quelltext verbindet. Codeablage an beliebiger Stelle und in beliebiger Bank (bei CBM und PC-128). Codeablage und Ablage mit Offset per Quelltext ein- und ausschaltbar. Erweiterte logische Operandenverknüpfung, Multiplikation/Division von Ausdrücken. - Profi-Werkzeug, seit einem Jahr in vielen Betrieben im Einsatz. Lieferumfang: Dokumentation, 2 EPROMs für AIM 65, 1 EPROM für CBM 610/710/720 + Textprogramm auf Floppy 8250, lauffähig im Cartridge-Port der Bank 15. DM 350,--. Für PC-128 von Commodore: Textprogramm (s. Heft 47) und Assembler auf Diskette 1541, DM 320,--.

6805 und 68705 Cross-Assembler
für CBM 610/710/720 und für PC-128

Ausstattung und Lieferung wie vor. DM 350,--

6800/6802/6803/6303 Cross-Assembler

Für CBM-Systeme und PC-128 in Vorbereitung

6805 und 68705 Cross-Assembler auf ATARI ST

Lieferbar ab Oktober 1986. Erzeugter Maschinencode wird in Form von S-Records auf anderen Rechner oder Eprommer übertragen. DM 350,–.

Mathe-ROM für 6502, Sockel $D000

Fließkommaarithmetik und höhere mathematische Funktionen wie in Microsoft-BASIC, für FORTH des AIM 65 oder Assemblerprogrammierung. 20 Seiten Dokumentation mit Einsprungspunkten und Argumenten. DM 124,30.

Den CBM 610/710/720 eindeutschen!

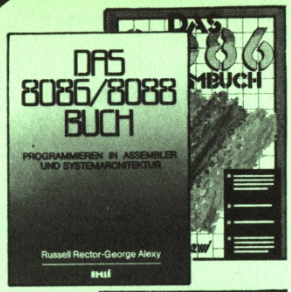
Deutsche Tastaturbelegung DIN-ASCII, deutsche Zeichen auf dem Bildschirm, wahlweise auf ASCII-Sonderzeichen umstellbar (Escape/z). Memory-Befehl mit ASCII-Ausdeutung der angezeigten Hexbytes, dafür Coprozessor-Routinen fortgelassen. 2 Eproms 2764 (Kernel und Zeichengenerator), DM 45.

Die Diskette zum Buch 'Assembler-Praxis auf Atari ST'

Die im Buch abgedruckten Utilities und Programme sind als Assembler-Quelltext enthalten (s.a. Heft 47), DM 42,--.

Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg
Tel.: 04 102 - 55 816

68008 Computer, Entwicklungssystem unter CP/M-68K laufend, 832 KB RAM, Floppy, RAM-Floppy, 2 RS232-Schnittstellen, Centronics Port, Userport (PIA), Echtzeituhr (Batterie) mit C-Compiler und 68000-Assembler (von Digital Research) sowie vielen Utilities, alles laufend, günstig abzugeben. Ferner: FORCE-Computer CPU-1 mit 68000 PIT 68230, 128 KB RAM (aufrüstbar auf 512 KB), Assembler und Screen-Editor, 3 Schnittstellen RS232. - Minibee-Computer mit implementiertem FORTH. Video-Terminalkarte nach mc, Sanyo-Monitor DM5912CX mit angenehmen grünen Phosphor, Power-Netzteil +5 Volt 15 A, +24 Volt 10A, - Roland Löhr.



DAS 8086/8088 BUCH
Rector/Alexy, 560 Seiten
Standardwerk zu Architektur, Funktion und Assemblerprogrammierung. Für Systemprogrammierer und Anwender von CPU-8086/88-PCs.
Softcover, DM 79,–

DAS 8086 SYSTEMBUCH
Thies, 200 Seiten
Einziger Entwicklertext über „redundante Multiprocessing-Systeme mit Busarbiter 8289" und „I/O-Interfacing-Technik" für Basis-Bussysteme.
Softcover, DM 49,– (Mitte 86)

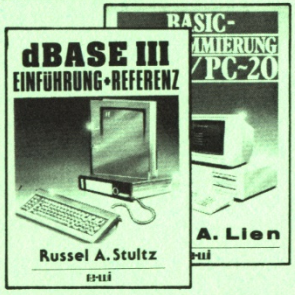
DIE ASM86/ASM286 MAKROASSEMBLER
Thies, 300 Seiten
Assemblerprogrammierung von 8086, 8087, 80186, 80286: ASM-Konzepte wie Segmente, Module, Prozeduren usw. werden über CPU/Speicherbilder funktionell erklärt.
Softcover, DM 69,– (Mitte 86)

DIE 8085/8086 INTERFACES
Thies, 633 Seiten, A4
Funktion und Programmierung der Bausteine von 8085/8086-CPUs anhand umfangreicher Funktionsbilder. Für 8205, 8282, 8255A, 8253, 8251A, 8237, 8259 usw.
Softcover, A4-Format, DM 89,–

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

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

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



NEU
dBASE III – Einführung + Referenz
Die deutsche dBASE III-Version – als dBASE III-Kurstext entwickelt, mit lexikalischer Befehlsdarstellung.
Von Russel A. Stultz. 464 Seiten. Softcover. DM 79,–

BASIC – Programmierung PC-10/PC-20
Eine systematische, kurserprobte BASIC-Spracheinführung, zugeschnitten auf das System PC-10/PC-20.
Von David A. Lien. 500 Seiten. Softcover. DM 59,–

IBM-PC-Handbuch
US-pragmatische, faktenreiche Systemübersicht. Als Textbuch für IBM-PC-BASIC-Kurse beliebt. Beschreibt u. a. auch DFÜ und wichtige Peripherie/Systemerweiterungen.
Von Lyle Graham. 416 Seiten. Softcover. DM 59,–

Neu IBM-PC/XT Assembler-Programmierung, CPU 8088
Besonderheit: Systemnahe Assemblerbeschreibung für direkte Kontrolle der IBM-PC-Komponenten. Detaillierte IBM-PC-Systemfakten durch hervorragendes Bildmaterial auch für Nicht-Professionelle.
Von Willen/Krantz. 416 Seiten. Softcover. DM 66.–

tm 4582

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

Noch im Programm:
Erfahrungen mit Gate Arrays DM 66,–
Einführung in die Mikrocomputertechnik DM 66,–

In Vorbereitung:
Das 80286 Buch

MICRO MAG

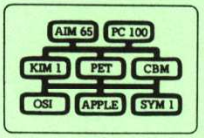
HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☎ (04102) 5 58 16

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- im Inland, bzw. DM 59,-- im Ausland (surface mail). Luftpostzustellung auf Anfrage. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu 4 Wochen vor deren Ablauf. - Nachliefermöglichkeiten siehe unten.

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

Leser-Service des Herausgebers

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM

Jetzt Preisvergünstigung bei Nachlieferungen,
solange der Vorrat reicht:

Buch 7-13 des MICRO MAG DM 25,--
Hefte 14/15 sowie 17-40 DM 3,50/St.
Hefte ab Nr. 41 DM 7,80/St
+DM 2,50/Sendung. Bestellungen im Wert bis zu DM 25,- werden nur bei Vorab-Überweisung oder Scheckeinsendung ausgeführt.

Mathe-ROM nach P. Rix für FORTH und Einbindung in Assembler-Programme, mit Dokumentation DM 124,30.

Assembler für 6502/65C02/65802/65816 (drei Befehlssätze!) sowie für MC6805 und für 6800/6802/6303 (drei Befehlssätze): Siehe im Heftinneren.

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