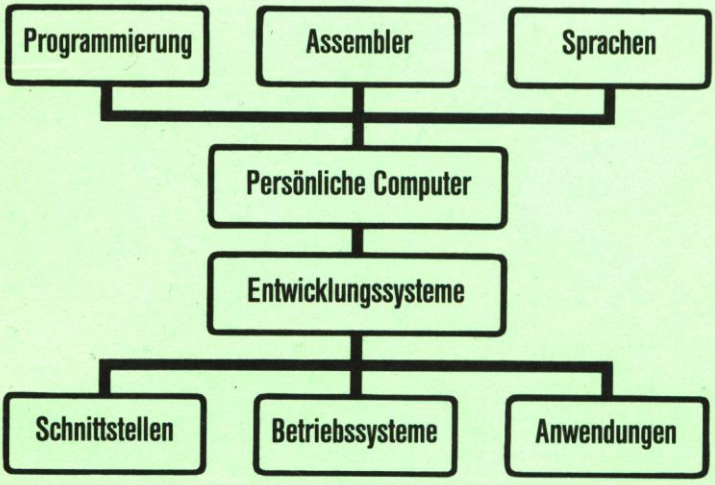
MICRO MAG

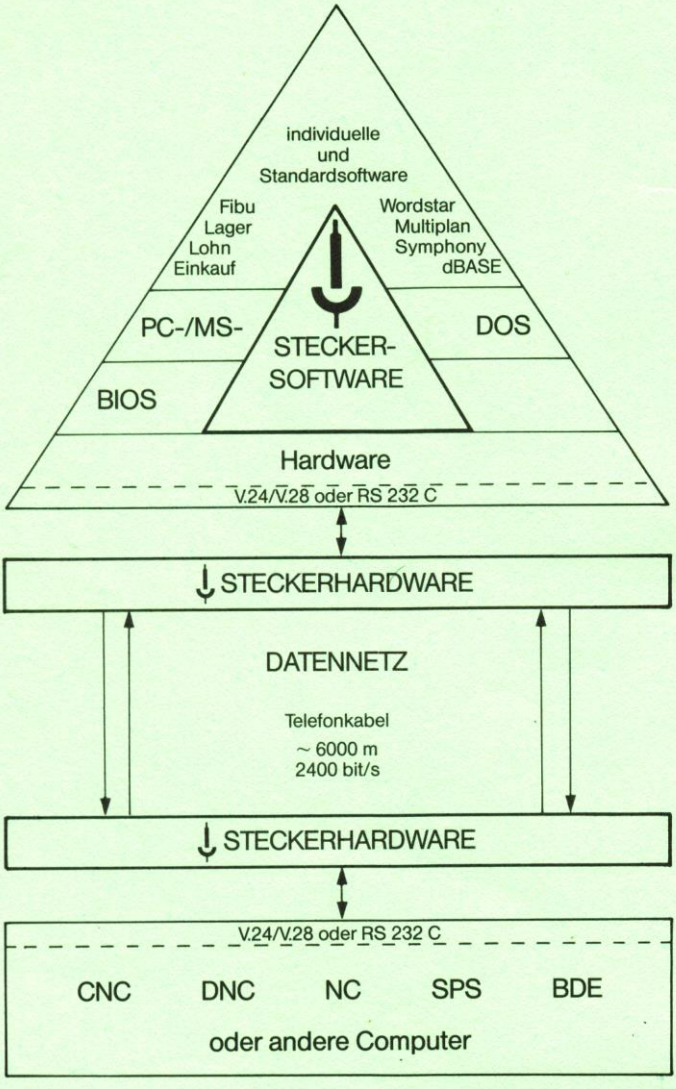
DM 9,50 Nr. 49 Juni 1987



Inhaltsverzeichnis

CIM und MAP

mit IBM und kompatiblen Personal-Computern



INGENIEURBÜRO
STECKER

5000 Köln 60 (Niehl)
Postfach 60 07 66
Delmenhorster Str. 20
Tel. (02 21) 7 12 40 18

Roland Löhr

Zahlenwandlung

Mit dem nachstehenden allgemeinen Programm für jeden 68xxx können sehr große dezimale Zahlenwerte in den Rechner eingegeben und in das binäre Rechenformat umgewandelt werden (Routine DEZHEX). Zur Umwandlung von Binärzahlen in eine BCD-Codierung wird die Routine HEXDEZ benutzt. Mit dem Unterprogramm BCDOUT kann eine solche BCD-Zahl hernach zur Wiederanzeige gebracht werden.

Die Umwandlung von Zahlen in beiden Richtungen ist eine häufig benötigte Dienstleistung. Sie kann selbst auf dem schnellen 68000 je nach Algorithmus viel Zeit in Anspruch nehmen, wenn es sich um eine massenhafte Benutzung handelt, z.B. bei der Ausgabe von Tabellen oder beim Einlesen von Zahlenstrings von der Floppy. Es kam daher auf einen Lösungsweg an, der nicht stur-schematisch abarbeitet, sondern der gegebene Abkürzungsmöglichkeiten ausnutzt, wenn z.B. eine Multiplikation mit einem Faktor Null erfolgen würde. Immerhin kostet eine Multiplikation zwischen Registern 70 Taktzyklen. Ein vorheriger Test des Operanden mit nachfolgendem Verzweigungsbefehl kostet jedoch nur 12-14 Zyklen. Wenn eine Dezimalzahl nach und nach eingelesen und in eine Binärzahl verwandelt wird, dann braucht man anfangs den höheren Teil des Ergebnisses nicht mit 10 zu multiplizieren, weil er noch Null ist.

Bei der Umwandlung einer Binärzahl nach dezimal (BCD) kann man ebenso abkürzen, wenn es sich um kleinere Werte handelt; man überspringt dabei die Umwandlung des höherwertigen Langwortes. Zu dieser Umwandlung ist weiterhin zu bemerken, daß sie nicht stur nach einem Horner-Schema verläuft, sondern eine Tabelle zu Hilfe nimmt, die in binärer Codierung die Zehnerpotenzen enthält. Man prüft den umzuwandelnden Operanden Stufe um Stufe, wieviele Vielfache einer Zehnerpotenz in ihm vorhanden sind. Dabei beginnt man mit der höchsten Zehnerpotenz und bildet Stufe um Stufe den Restwert des Operanden.

Es können sehr große Zahlen in der Eingabe und in der Rückumwandlung behandelt werden, nämlich bis zu 2 Exponent 64. Das entspricht 1,84 E19, also 19-20 gültigen Dezimalziffern. Maschinenintern erfolgt die binäre Darstellung entsprechend in 2 Langworten (64 Bits) und dezimal in einem 10 Byte breiten Feld ERGEBNIS.

Die CPU 68xxx ist für die binären Grundrechenarten recht gut mit Befehlen ausgerüstet. Additionen und Subtraktionen können in verschiedener Breite nicht nur unter Zuhilfenahme von Registern durchgeführt, sondern auch auf Datenfelder beliebiger Breite ausgeübt werden. Bei Multiplikationen und Divisionen ist immer ein Datenregister als Zieladresse zu benutzen. Hier gelten die Beschränkungen 16*16 Bits=32 Bits bzw. 32/16 Bits=16 Bits + Rest 16 Bits. Für diese beiden Rechenarten finden wir erst ab 68020 eine größere Operationsbreite, nämlich z.B. 32*32 Bits=64 Bits bzw. 64/32 Bits=32 Bits + 32 Bits Rest. - Soweit man also keine 68020 zur Verfügung hat, müssen Multiplikation und Division großer Zahlen stufenweise ausgeführt werden.

Die dezimalen Rechenarten sind auf dem 68xxx nur mit ABCD, SBCD und NBCD für Addition, Subtraktion und Komplementbildung vertreten. Sie arbeiten per Befehl nur an 1 BCD-Byte, was bei höheren Werten viele Schleifendurchläufe nach sich zieht. Aus diesem Grunde ist es empfehlenswert, Berechnungen vorwiegend mit den Befehlen für binäre Operanden vorzunehmen und dezimale Zahlen bei der Eingabe umzuwandeln bzw. für die Ausgabe zu erzeugen. Dem dienen hier die Unterprogramme DEZHEX und HEXDEZ.

Zum Programm: Für Experimente wurden die entscheidenden Routinen DEZHEX und HEXDEZ in eine interaktive Umgebung eingebunden. Ein Eingabestrom wird per GETLINE in einen Buffer gebracht. Bei MAIN2 werden führende Spaces übergangen. DEZHEX verwertet dann alle folgenden Dezimalziffern der Eingabe, bis eine Nicht-Ziffer folgt oder bis ein Überlauf eintritt. Erst ab 9. eingegebener Ziffer (in DIGICOUNT) kommt ein Übertrag in das höherwertige Langwort in Betracht, so daß sich davor noch Abkürzungen ergeben. Das trifft auch zu, wenn der Eingabestrom mit führenden Nullen beginnt. - Die zahlreichen Multiplikationen hängen mit der o.a. Beschränkung auf jeweils 16*16 Bits zusammen. Ein Langwortregister wird daher in 2 Teilen multipliziert, die nachher stellenrichtig zu addieren sind (daher SWAP). Auch können sich Überträge in ein höherwertiges Langwort ergeben. - Mit dem Befehl SNE setzen wir wir ein Merk-Byte mit den erreichten Status. Trifft die Abfrage zu (hier: Not Equal), dann wird das Byte zu $FF gesetzt, sonst zu 00. Diese Merker werden später mit dem Befehl TST.B abgefragt.

HEXDEZ bildet aus zwei Langworten ein Ergebnis in BCD-Form. Es wird die Tabelle mit den Zehnerpotenzen ab Label ZP19 bzw. ZP9 benutzt. Wenn das höherwertige Langwort nämlich 00 ist, dann können wir gleich bei HEXDEZ0 anfangen.

Die Befehlsfolgen ab HD0 bzw. ab HEXDEZ1 sind jeweils zweistufig, und zwar mit folgender Überlegung: Je nach Vergleichsoperand ZP19...ZP0 in der Tabelle kann sich jeweils ein BCD-Byte im Wertebereich 00...90 bzw. 00...09 ergeben. Oder wenn man zwei Vergleichsoperanden nacheinander verwertet, dann ergeben sich im zählenden Register D4 Byte-Werte zwischen 00 und 99. Ein Überlauf aus den Ziffernstellen kann nicht erfolgen, weil wir immer von Resten ausgehen. Dieses Sammeln einzelner Bytes für das Ergebnis hat den Vorteil, daß Wertänderungen immer nur in einem 8 Bit breiten Fenster stattfinden, das nach jeder ausgeführten doppelten Stufe in das Ergebnis einkopiert wird.

Der vorgenannte Lösungsweg läßt sich natürlich für andere Zahlensysteme und auch für andere Genauigkeiten anpassen.

```
* WANDLUNG3
* Zahlenwandlung HEXDEZ im Subtraktionsverfahren
* um Zehnerpotenzen mit einer Tabelle
* Umwandlung dezimal zu binär und umgekehrt, interaktiv
* Wertebereich bis 1,84*10^19, entsprechend 64 Bits
* Copyright 1987 by Roland Löhr

* Verkehrszeichen
cr        equ       $0d             Carriage Return
lf        equ       $0a             Linefeed
escape    equ       $1b             Escape-Taste
space     equ       $20             Zwischenraum
GEMDOS    equ       1               TOS-Aufrufe

* Folgende Unterprogramme zur Ansprache des GEMDOS sind an anderer
* Stelle in diesem Heft/ in dieser Zeitschrift dargestellt worden:
* crlf = CR und LF ausgeben
* printline = null-terminierten Textstring ausgeben
* getline = Eingabezeile nach INBUFF holen
* term = Rückkehr zum TOS

 SECTION  ONE

main      movem.l   a3-a5,regsave   Register sichern
main1     bsr       crlf            Neue Zeile
          move.l    #text1,d0       Uberschrift anzeigen
          bsr       printline       und Dezimalzahl anfordern
          bsr       crlf
```

```
            lea.l     inbuff,a5       Zeige auf Tastaturpuffer
            move.b    #0,(a5)         Delimiter vorne in den Puffer
            bsr       getline         1 Zeile holen
            cmp.b     #escape,d0      Will man abbrechen?
            beq       abbruch         ja
            bsr       crlf            Neue Zeile

main1a      lea.l     inbuff,a5       Zeige auf Zeilenbeginn
            clr.l     d0              für spätere Addition
main2       move.b    (a5)+,d0        1. Zeichen holen
            beq.s     main1           Zeile ist leer
            cmpi.b    #space,d0       Führende Leerzeichen?
            beq.s     main2           ja, übergehen

            bsr       dezhex          Ziffern filtern, Eingabe umwandeln
*                                     zu 64 Bit nach D3.L und D3.L

* Binärzahl in D2.L und D3.L zurückverwandeln zu Dezimalzahl
* unter Benutzung einer Umwandlungstabelle für Zehnerpotenzen

            bsr       hexdez          Umwandlung mit Ergebnis
*                                     im Feld ERGEBNIS

main4       lea.l     ergebnis,a5     Ergebnisausgabe
            move.w    #9,d1           Ausgabe von 20 Ergebnis-Ziffern,
            bsr       bcdout          die BCD-codiert sind
            bsr crlf
            bra       main1

abbruch     movem.l   regsave,a3-a5   Register zurückholen
            jmp       term            Programm beenden

* dezhex wandelt string zu binär in den Registern D2.L und D3.L
* d0, d4, d5 sind Hilfsregister der Multiplikation

dezhex      clr.l     d2              Ergebnisfeld = 0, höherwertiger Teil
            clr.l     d3              dito, niederwertige Teile
            clr.l     d4              dito
            move.w    #10,d1          Zahlenbasis, Faktor konstant 10
            clr.w     digicount       Zahl bisheriger Ziffern
            bra       dezhex1         abkürzen, nur für die 1. Ziffer

dezhex0     bsr       dezzif          Erlaubte Eingabe?
            bmi       dezhex3         nicht (mehr) hex, Abbruch
            cmp.w     #8,digicount    Noch unter 10^9 ?
            bcs.s     dezh00          ja, < 8 Dezimalziffern, abkürzen
* Höherwertiges Langwort bearbeiten
            move.l    d2,d4           Multiplikationsvorbereitung
            beq.s     dezh00          abkürzen, wenn 0
            clr.w     d4
            swap      d4              Hoher Teil multiplizierbar gemacht
            sne       d4stat          Merke, ob Faktor 0?
            mulu      d1,d2           Teil Low*10
            tst.b     d4stat          allerhöchstes Wort <>0 ?
            beq.s     dezh            ja, abkürzen
            mulu      d1,d4           Ziffern*10
dezh        swap      d2              hohes Wort der Multiplikation
            add.w     d4,d2           stellenrichtig einziehen
            swap      d2              D2 in normale Lage zurück
            swap      d4              gab es überlauf?
            tst.w     d4
            beq.s     dezh00          nein, weiter
dezhfehl    move.l    #text2,d0       Fehlertext anzeigen
            bra       printline       rts dort
* Niederwertiges Langwort bearbeiten
dezh00      move.l    d3,d5           Kopie machen
            sne       d3stat          merke Status D3
```

```
            clr.w     d5              hohen Teil als Faktor isolieren
            swap      d5              erledigt, jetzt bisheriges Ergebnis
            sne       d5stat          Faktor kann 00 sein
            tst.b     d3stat
            beq.s     dezh01          nicht multiplizieren, abkürzen
            mulu      d1,d3           D3-Low mit Basis 10 multiplizieren
dezh01      tst.b     d5stat          ist was im Teil High?
            beq.s     dezh02          nein, abkürzen
            mulu      d1,d5           *10
            swap      d3              vorderes Wort
            add.w     d5,d3           stellenrichtig einziehen
            swap      d3              D3 wieder in Normalstellung
dezh02      clr.w     d5              der auf Bit 0-15 fallende Teil

            swap      d5              Bit 16-31 isoliert, führende Nullen
            addx      d5,d2           Einziehen eines Übertrages
dezh03      bcs.s     dezhfehl        Überlauf, Fehler

dezhex1     andi.b    #$0f,d0         maskiere vordere Bits, Eingabeziffer
            add.l     d0,d3           Eingabeziffer hinzuaddieren
            bcc.s     dezhex2         Wenn kein Übertrag entstand
            addq.l    #1,d2           Übertrag einaddieren
dezhex2     add.w     #1,digicount    Zahl Eingabeziffern +1
            move.b    (a5)+,d0        Nächstes Zeichen holen
            bne       dezhex0         und umwandeln
dezhex3     rts

* Umwandlung einer Binärzahl in D2.L und D3.L (bis 64 Bits)
* in eine BCD-Zahl, Ablage in ERGEBNIS

hexdez      clr.w     ergebnis        Ergebnisfeld (10 Bytes) löschen
            clr.l     ergebnis+2
            clr.l     ergebnis+6
            lea       ergebnis,a5     Zeiger für Zwischenergebnisse
            lea       zp19,a4         Zeiger auf Umwandlungstafel
            tst.l     d2              Gibt es sehr hohe Werte?
            beq.s     hexdez0         nein, abkürzen
* Höherwertiges Langwort bewerten
hd0         move.l    (a4)+,d1        Hohes Langwort aus der Tabelle und
            move.l    (a4)+,d0        niedriges zum Vergleich laden
            clr.b     d4              Stellenergebnis=0
hd1         cmp.l     d1,d2           Vergleich hohe Langworte
            beq.s     hdequ           bei gleich: Noch Low prüfen
            bcc.s     hdborg          größer, alles subtrahierbar
            bra.s     hd3             kleiner, schalte 1 Stelle weiter
hdequ       cmp.l     d0,d3           bei gleich: wie niedriges Langwort?
            bcs.s     hd3             Man kann nichts borgen
hdborg      add.b     #$10,d4         Ergebnis+10
            sub.l     d0,d3           Tabellenwert Low abziehen
            subx.l    d1,d2           und High
            bra.s     hd1             steckt noch mehr in dieser Potenz?

hd3         move.l    (a4)+,d1        Nächstes Wertepaar aus der Tabelle
            move.l    (a4)+,d0        laden: High, dann Low
hd4         cmp.l     d1,d2           Vergleich hohe Langworte
            beq.s     hdequ1          Bei gleich
            bcc.s     hdborg1         größer, alles subtrahierbar
            bra.s     hd6             kleiner, schalte 1 Stelle weiter
hdequ1      cmp.l     d0,d3           bei gleich: wie niedriges Langwort?
            bcs.s     hd6             Man kann nichts borgen
hdborg1     add.b     #$01,d4         Ergebnis+1
            sub.l     d0,d3           Tabellenwert Low abziehen
            subx.l    d1,d2           und High
            bra.s     hd4             ist noch mehr in dieser Potenz?

hd6         move.b    d4,(a5)+        2 Dezimalziffern aus D4 ablegen
            cmp.l     #$540be400,d0   Abbruch mit 10^10
            bne.s     hd0             Langwort D2 weiter bewerten
```

```
* Wertebereich des unteren D3 Langwortes umwandeln
hexdez0   lea       ergebnis+5,a5  Ablage von Zwischenergebnissen
          lea       zp9,a4         Zeige auf Zehnerpotenzen-Tabelle
hexdez1   move.l    (a4)+,d0       Vergleicher aus Tabelle
          clr.b     d4             Dezimalziffern soweit 00
hexdez2   cmp.l     d0,d3          d3 >= Vergleicher?
          bcs.s     hexdez3        nein
          add.b     #$10,d4        Ergebnis+10
          sub.l     d0,d3          Rest bilden
          bra.s     hexdez2        ist mehr in der Dezimalstelle?

hexdez3   move.l    (a4)+,d0       Folgenden Tabellenwert nehmen
hexdez4   cmp.l     d0,d3          d3 >= Vergleicher?
          bcs.s     hexdez5        nein
          add.b     #$01,d4        Ergebnis+10
          sub.l     d0,d3          Rest bilden
          bra.s     hexdez4        kommt noch etwas hinzu?

hexdez5   move.b    d4,(a5)+       2 Ziffern ins Ergebnisfeld übertragen
          cmp.l     #$1,d0         letzter Vergleichsoperand?
          bne.s     hexdez1        nein, weiter
          rts


* Ausabe eines BCD-Ergebnisses
bcdout    move.b    (a5),d0        A5 zeigt auf Ergebnisfeld
          lsr.b     #4,d0          MSB isolieren
          ori.b     #$30,d0        zu ASCII machen
          bsr       conout         ausgeben
          move.b    (a5)+,d0       LSD
          andi.b    #$0f,d0        BCD-Wert isolieren
          ori.b     #$30,d0        zu ASCII
          bsr       conout         ausgeben
          dbra      d1,bcdout      Ausgabeschleife
          rts


* Tabelle der Zehnerpotenzen zur Umwandlung HEXDEZ
zp19      dc.l      $8ac72304,$89e80000 10^19, entspricht knapp 2^64
zp18      dc.l      $de0b6b3,$a7640000
zp17      dc.l      $1634578,$5d8a0000
zp16      dc.l      $2386f2,$6fc10000
zp15      dc.l      $38d7e,$a4c68000    10^15
zp14      dc.l      $5af3,$107a4000
zp13      dc.l      $918,$4e72a000
zp12      dc.l      $e8,$d4a51000
zp11      dc.l      $17,$4876e800
zp10      dc.l      $2,$540be400

zp9       dc.l      $3b9aca00       10^9 =Grenze 2^32 unteres Langwort
zp8       dc.l      $5f5e100        10^8
zp7       dc.l      $989680
zp6       dc.l      $f4240
zp5       dc.l      $186a0

zp4       dc.l      $2710
zp3       dc.l      $3e8
zp2       dc.l      $64
zp1       dc.l      $a
zp0       dc.l      $1

* Textkonstante
text1     dc.b      'Zahlenwandlung, ESC führt zu TOS zurück ',cr,lf
          dc.b      'Bitte eine Dezimalzahl eingeben  ',cr,lf
          dc.b      'Maximal 20 Ziffern <= 1,844 10^19:  ',cr,lf
          dc.b      'Dez-Hex und dann Hex-Dez',cr,lf,0
text2     dc.b      'Zahl zu gross! ',cr,lf,0
```

```
* Arbeitsspeicher
regsave   ds.l    3       Ablage von Registern
ergebnis  ds.w    1       Ergebnisfeld
          ds.l    4       Ergebnisfeld
inbuff    ds.b    80      Allg. Eingabepuffer
filtflag  ds.w    1       Flag-Variable

d2stat    dc.b    1       Status Register D2
d3stat    dc.b    1       Status Register D3
d4stat    dc.b    1       Status Register D4
d5stat    dc.b    1       Status Register D5
digicount dc.w    1       Zahl ausgewerteter Dezimalziffern
*                         d. Eingabe
```

□○□

Roland Löhr

Omikron-BASIC für Atari ST

Etwa seit Beginn des Jahres 1987 erfreut sich das OMIKRON-BASIC zunehmender Beliebtheit. Es wurde von jungen Leuten (Artur Södler und Carsten Kraus) implementiert, die dem Publikum damit eine Leistung in der Spitzenklasse bei der Ausnutzung des MC68000 präsentieren. Dieses BASIC ist sehr schnell, hat eine hohe Rechengenauigkeit, ist in der Formulierung weitgehend kompatibel zum MBASIC, bietet jedoch mehr. Es tritt später und mit geringeren Werbeetats als das GFA-BASIC an, das hier noch nicht erprobt wurde.

Hardware: OMIKRON-BASIC wird geliefert als ROM-Modul für den Modul-Port des ST, zusammen mit einer Diskette und einem 179-seitigen Handbuch, das mit seinen Beispielen für die Benutzung der Befehle als gut lesbar und informativ bezeichnet werden kann. Allerdings würden sich auch erfahrene Programmierer hier und da ein Mehr an Information wünschen. Auf der Diskette findet man etwa 20 Dateien einerseits zur Dokumentation, andererseits mit Beispielprogrammen. Besonders zu erwähnen sind vor-formulierte Quelltexte zur Benutzung der Bedieneroberfläche GEM (mit VDI und AES) sowie Datenbank-Quelltexte für ISAM.

Beide Themenbereiche sind durch Beispielprogramme in BASIC unterstützt. Während es für GEM mit VDI und AES jedoch bereits Bücher gibt, in denen man nachschlagen kann, muß die Dokumentation zur Benutzung des ISAM jedoch noch als schwach bezeichnet werden. Man darf nicht aus zu knapp formulierten Kommentaren mit zeitaufwendigen eigenen Versuchen erst zu Erfolgen kommen können. Hier müßte nachdokumentiert werden.

Charakteristik: BASIC mit strukturierenden Sprachelementen (REPEAT UNTIL usw.), Prozeduren, ein- und mehrzeilige Funktionen mit Parameterübergabe, Aufruf von Assemblerprogrammen mit USR und CALL mit Parameterübergaben, Bezeichner für Variablen, Prozeduren usw. dürfen 32 Zeichen lang und damit sehr unterscheidungsfähig sein. Programmverzweigungen und Sprünge können zu Labeln oder (herkömmlich) zu Zeilennummern erfolgen. Die Ein- und Ausgabefunktionen sind reichhaltig und unterstützen auch die Massenspeicherung nebst relativen Dateien.

Programme und Befehle können direkt eingetippt werden, wobei wie bei Commodore gilt: What you see is what you get. Man kann also mit dem Cursor in angezeigte Zeilen fahren, sie berichtigen und Befehle in veränderter Form zur Ausführung bringen. Daneben steht ein Fullscreen-Editor zur Formulierung von Programmen zur Verfügung, der auch bequeme Such-, Ersetzungs- und Blockfunktionen hat. Die erweiterten mathematischen Funktionen können mit einer Rechengenauigkeit bis zu 19 Stellen ausgeführt werden. Es stehen leistungsfähige Matrizenfunktionen und Hilfen für die Fehlerbehandlung und die Programmerprobung (TRACE) zur Verfügung.

Für spezielle Anwendungen hat man daneben grafische und Soundfunktionen, Alertboxen, Datei-Auswahlboxen, Mausüberwachung und eine Art Multitasking mit "ON Ereignis GOSUB". Die mitgelieferte GEM-Bibliothek ermöglicht die Einbettung von Programmen in diese Benutzerumgebung. Mit der ISAM-Bibliothek können Datenbanken aufgebaut werden. Erstellte Programme können auch ohne die ROM-Cartridge ausgeführt werden, wenn dem Anwender der frei kopierbare Runtime-Interpreter zur Verfügung gestellt wird. Das ist sicher für Entwickler von Software interessant.

Nach diesem allgemeinen Oberblick, der bereits die besondere Leistungsfähigkeit des OMIKRON-BASIC erkennen läßt, gehen wir wir auf einige erwähnenswerte Einzelheiten ein:

Der Editor. Ein leistungsfähiges Textsystem ist für ein zügiges Formulieren und Bearbeiten noch immer eine Grundvoraussetzung. Sie ist mit dem Fullscreen-Editor EDIT sehr gut erfüllt. Er benutzt die Funktionstasten, CTRL-Zeichen und Escape-Sequenzen wie im Terminal-Modus. Es kann bildschirmweise vor- und zurückgeblättert werden, man setzt mit einem Tastendruck auf den Textanfang oder auf das Textende, geht gezielt zu einer Zeile mit Nummer (man kann auch ohne angezeigte Zeilennummern programmieren), man kann sich alle Vorkommen eines Suchstrings in inverser Schrift anzeigen lassen und man kann Strings automatisch oder nach Rückfrage ersetzen. Mit den Blockfunktionen kann man Textteile entfernen, kopieren, von einer Datei einziehen oder auf eine solche absetzen. Man kann in zwei Bildfenstern arbeiten, um z.B. Textstellen zu vergleichen. Der Cursor ist für Korrekturen frei beweglich, er kann auch gezielt an z.B. an den Zeilenanfang oder das Zeilenende gesetzt werden. Alle Funktionen führen schnell aus. Der Editor nimmt auch eine Vorprüfung des eingegebenen Programmes vor.

Variablen: Mit Bezeichnern von bis zu 32 Zeichen (auch bei Funktionen, Prozeduren und Labeln) lassen sich Programme gut dokumentierend formulieren. Der normale Variablentyp (Name ohne Zusätze) ist eine Zahl vom Typ long integer (31 Bits und Vorzeichen, Wertebereich bis +-2,1 Mrd), während er in den BASIC-Versionen von Commodore und anderen eine Gleitkommazahl war. Insofern muß man bei der Obernahme von Programmen aufmerksam sein. Dieser Variablentyp hat aber erhebliche Vorteile bei vielen Berechnungen, denn er ist der internen 32-Bit Struktur der CPU angepaßt. Mit ihm dauert z.B. das blinde Hochzählen von 1...100.000 in einer Programmschleife nur 2,4 Sek!

Mit entsprechenden Bezeichnern haben wir daneben Flagvariablen (1 Byte), Integer Byte (8 Bits), Integer Word (16 Bits) sowie die einfach- und die doppeltgenauen Gleitkommazahlen (angezeigte Genauigkeit 9 bzw. 19 Stellen. Beide Typen decken den ungeheuren Wertebereich von +-5,11 E+-4931 ab. Daneben gibt es natürlich die Strings mit erweiterten Funktionen. Wie in FORTRAN lassen sich Buchstabenbereiche der Variablennamen einem Variablentyp zuordnen. Zahlen können können auch zur binären, oktalen oder hexadezimalen Interpretation eingegeben werden (Bezeichner %, & und $). Für die Abspeicherung von Zahlen auf externe Speicher gibt es Umwandlungsfunktionen, die ein gleichmäßiges Aufzeichnungsformat und eine entsprechende Rückumwandlung in den Datentyp gewährleisten.

Bei den arithmetischen Funktionen, die auch in doppelter Genauigkeit ausgeführt werden können, sind erwähnenswert: Abschneiden des Nachkommateiles, des Vorkommateiles, Maximum und Minimum zweier Zahlen, Zufallszahlen mit Bereichsfilter, Division mit Modulo, höhere Winkelfunktionen, schnell ausführende Fakultätsberechnung (bis etwa 1750!) und Matrizenbefehle. - Bei den logischen Funktionen findet man auch NAND, NOR, BIT Shift rechts und links um n Bitpositionen, logisches Äquivalent und die Implikation.

Die Stringfunktionen sind gegenüber früheren BASIC-Versionen wie folgt erweitert: Stringmultiplikation, natürlich Instring, Spiegelfunktion, Umwandlung groß/klein, Einsetzen in den Midstring. - Prozeduren sind bereits benutzbar, sobald sie im Programmtext stehen, also noch vor einem RUN. Sie können damit für den Tischrechnermodus wie eine Spracherweiterung benutzt werden. Prozeduren können mit EXIT verlassen werden.

Bei den Programmierhilfen wurde bereits der Trace (Einzelschritt) erwähnt. Es gibt auch ein ON TRACE GOSUB, mit dem man z.B. die Entwicklung von Variablen beobachten kann, ohne immer neu tippen zu müssen. Variablen können auf den Bildschirm und auf den Drucker gedumpt werden. Ebenso gibt es die Fehlerbehandlung ON ERROR GOTO. Man kann auch die Funktionstasten mit Texten belegen oder sagen ON Funktionstaste GOSUB.

Ein- und Ausgabefunktionen: In früheren Dialekten gab es Beschränkungen beim Befehl INPUT, die hier mit LINE INPUT auch beim Lesen von Diskette umgangen werden können. Mit diesem Befehl kann man z.B. Texte von 1st WORD einlesen.

Für den Bildschirm gibt es INPUT AT (Zeile Spalte) und vor allem das INPUT AT USING. Der Kontrollstring USING kann dabei die für ein Feld zulässigen Zeichen (Ziffern, Buchstaben, Sonderzeichen, Einzelzeichen) enthalten und Buchstaben grundsätzlich zu groß oder klein oder Einzelzeichen gezielt umwandeln (z.B. Punkte in Kommas). Daneben kann man die Eingabelänge definieren und das Abbruch-Kriterium für die Eingabe, das auch eine Cursortaste oder ein Scancode sein kann. Die abbrechende Taste erhält man per Variable zurück, so daß man im weiteren Verlauf logische Entscheidungen treffen kann. Damit hat man sehr schnell und einfach eine Eingabemaske formuliert. Am Schluß findet sich ein kleines Programmbeispiel.

Die Ausgabe enthält im wesentlichen Spielarten des Befehles PRINT: PRINT AT, PRINT USING (Formatierung), LPRINT (Drucker), PRINT auf logische Datei, PRINT (und auch INPUT) auf seriellen Kanal. WRITE gibt auch die Kommas in einem String aus und ist damit das Gegenstück zu LINE INPUT. Beim Schreiben auf Dateien wird an den String hex 0d 0a als Zeilenende angehängt, wenn auf PRINT' nichts folgt. Folgt ein Komma, so findet man hex 09 (TAB) auf der Datei. Folgt ein Semikolon, so steht dort kein Trenner.

Dateien werden mit OPEN, CLOSE, PRINT' und INPUT' (in Spielarten) angefaßt. Bis zu 16 logische Dateien können angesprochen werden. Für relative Dateien gibt es FIELD für die Erkärung von Buffern, LSET und RSET für die Obertragung von Feldern in den Buffer sowie PUT log. Datei,Satz-Nr und ebenso GET für das Schreiben aus dem Buffer in die Datei bzw. für das Lesen in den Buffer. - Programmdateien werden mit LOAD und SAVE behandelt, wobei auch ein ASCII-Abzug ohne die Tokens möglich ist (Weiterbearbeitung in einem Textsystem). Per BASIC kann man ferner MERGE und CHAIN zum Zusammenfügen oder für die Ablaufsteuerung von Programmen benutzen.

Maschinennahe Programmierung: Die üblichen Befehle PEEK und POKE gibt es in den Ausführungen für Byte, Wort und Langwort. Die Speicheradresse von Variablen erhält man mit der Funktion VARPTR. Mit CLEAR und Parametern reserviert man einen gegen das BASIC geschützten Speicherbereich. MEMORY reserviert und gibt die Beginnadresse zurück. DEF USR definiert die Startadresse für ein Assemblerprogramm. A=USR(X) ruft ein solches Programm auf, übergibt ihm X als Parameter und liefert in A den Inhalt des CPU-Registers D0 zurück. Der Befehl CALL ruft ein Programm an einer Adresse auf und übergibt ihm Parameter auf dem Stack. Die Befehle BIOS, XBIOS und GEMDOS rufen mit Parameterübergaben hin und zurück die entsprechenden Teile des Betriebssystemes auf. Das gilt entsprechend für die Befehle VDI und AES,

die das GEM ansprechen. Programme und Bilder werden mit BLOAD und BSAVE in Bereiche gezielt geladen bzw. von dort abgespeichert. - Assemblerprogramme sollten für die anderen Erfordernisse geschrieben sein und entweder nur verschieblichen Code enthalten oder für feste Ladeadressen (Direktive ORG) im Zusammenspiel mit BASIC formuliert sein.

Zusammenfassend: OMIKRON-BASIC ist reichhaltig ausgestattet. Es ist nicht nur, wie manche Besprechungen in den Zeitschriften auch wegen der Geschwindigkeit glauben machen wollen, besonders für mathematische Anwendungen mit hoher Genauigkeit zu empfehlen. Seine Stärken liegen ebenso in der leichten interaktiven Steuerung und Prüfung von Eingaben schlechthin, in seinem Interface zur Dateiverwaltung, zu grafischen Funktionen und zu Leistungen des Betriebssystems oder von Assemblerprogrammen. Die Bibliotheken für ISAM und GEM sind eine weitere Bereicherung. - BASIC ist eine leicht erlernbare Sprache, die schnelle Abprüfungen und Erfolge bringt, ohne daß man umständlich kompilieren und linken muß. Das bleibt seine Stärke. Das OMIKRON-BASIC ist dem Atari und seinem 68000 auf den Leib geschrieben und hat Leistungsmerkmale, die den umständlicheren Gebrauch von C oder Assembler nur noch in einer geringeren Zahl von Fällen empfehlenswert machen. Oder: BASIC ist hier als Sprache deutlich aufgewertet.

Zum Schluß ein kurzes Formulierungsbeispiel, das der ISAM-Demo entnommen und vom Autor nachkommentiert wurde. Es soll vor allem die Prozedur Eingabe (ab Zeile 1000) mit INPUT AT USING (in 1050) zeigen. Der USING-String läßt als Eingaben Buchstaben (die mit U immer zu groß gewandelt werden), Ziffern und Sonderzeichen zu. Mit verschiedenen Tasten kann abgebrochen werden. Ihr Scancode wird in TASTE zurückgegeben und ab 1051 entschlüsselt. Mit CR und Cursortasten wird die Eingabe in einer anderen Zeile fortgesetzt. Mit EXIT n (n=Ebenen) oder EXIT TO Label wird die Prozedur verlassen. Label sind daran zu erkennen, daß ihr Name mit einem Bindestrich beginnt wie bei -Schleife, -Eingabe1 oder -Titel. Kommentare beginnen bei einem Hochkomma.

Eine Prozedur wird mit dem Namen aufgerufen und ggfs. dahinter mit den Parametern in Klammern. ISAM-Prozeduren sind z.B. Is_Entry(), Is_Open(), Is_Insert(). Die ISAM-Prozeduren der Bibliothek wurden hier nicht abgedruckt. Eingabe in Zeile 110 ist ein Prozeduraufruf ohne Parameter. Wenn man sich über Programmierstil auch unterhalten kann: Man sieht, mit wie wenig Aufwand man z.B. die Eingabemaske für eine Datenbank o.ä. formulieren kann und wie aussagekräftig sich Bezeichner wählen lassen.

```
0 PRINT ""
1 ' Demo-Programm zur ISAM-Library
2 ' Isam-Library ist eingebunden, Anleitung dazu (IS_LIB.DOC) ebenfalls
3 ' 2 Dateien anlegen: ADRESSEN.DAT (Stammdatei) und ADRESSEN.IO1 (Index)
4 '
5 MODE "D"
10 Is_Entry(0,1,0,30,1)' Name als (einziges) Suchkriterium erklären
12 '
15 ' Handle, Nr. d. Suchkriteriums (1-10), Länge dessen (1-  ), Typ: 0=ASCII,
     1=alfa, 2=numerisch
20 '
30 Is_Open(0,"adressen",120,1,1)'Datei öffnen, dabei sind
32 '
35 ' Handle, Name$, RecLänge, Datei-# (1-15), Anzahl Such/Sortierkriterien
36 '
40 KEY 10="" ' Taste [F10] belegen, damit spätere Reaktion möglich
41 '
100-Schleife' Eingabeschleife
110    Eingabe'Prozedur mit weiterer Befehlsentschlüsselung
120    FOR I=1 TO 4: LSET Buffer$(I)=Eingabe$(I): NEXT 'übernahme in den Buffer
130    Is_Insert(0)'Prozedur mit Handle, Sortiert laufenden Satz ein
140 GOTO Schleife
```

```
150-Suchen' ein Label
160   LSET Buffer$(1)=Eingabe$(1)'Suchbegriff der Eingabe entnehmen
170   Is_Search(0,1,Rec_Nr)' Suche mit Erfolg in Rec_Nr
172 ' Handle, I, R
173 '
180   IF Rec_Nr THEN ' erfolgreich
190     GET Is_Fnr(0),Rec_Nr' Datei- und Satz-Nummer, Laden
200     FOR I=1 TO 4:Eingabe$(I)=Buffer$(I): NEXT ' aus dem Buffer übertragen
210   ENDIF
220 GOTO Schleife
225 '
900-Ende' mit ESCAPE bei Eingabe angesprungen
910 Is_Close(0)' Dateien schließen
990 END
999 '
1000 DEF PROC Eingabe'Eingabe und Befehlsentschlüsselung
1010 LOCAL I,Taste
1015 PRINT "" Bildschirm löschen
1017 PRINT "Suchen nach Name mit [HELP], Abspeichern mit [F10], Ende mit [ESC]"
1020 RESTORE Titel: FOR I=1 TO 4: READ Titel$(I): PRINT @(5+I*2,1);Titel$(I);Ein
gabe$(I): NEXT
1025 'Feldnamen laden und anzeigen sowie Feldinhalte
1030 I=1: REPEAT
1040   -Eingabe1
1045 '
1046 ' INPUT USING Feldname, Eingabe-Variable Buchstaben, Ziffern, Sonderzeichen
 Grossbuchstaben, Abbruchtasten/Scancode: [HELP] [F10] [ESC] Curs. up down
1047 ' dann Rückgabevariable [Taste], Feldlänge 30
1050   INPUT @(5+I*2,1)+Titel$(I);Eingabe$(I) USING "a0%UsbsDss"+ CHR$($48)+"s"+
 CHR$($50),Taste,30
1051   Taste=(Taste AND $FF0000)/$10000'Scancode als Byte isolieren
1052   IF Taste=$44 THEN EXIT 2' Abspeichern bei [F10], EXIT IF und Eingabe
1055   IF Taste=$62 THEN EXIT 2 TO Suchen'bei [HELP]
1057   IF Taste=1 THEN EXIT 2 TO Ende' bei [ESC]
1060   IF Taste=$48 THEN
1070     IF I>1 THEN I=I-1' bei Cursor up zurückgehen
1080     GOTO Eingabe1
1090   ENDIF
1100   IF I<4 THEN I=I+1' Folgezeile einstellen
1110 UNTIL 1=0' endlosschleife
1115 '
1120 RETURN
1125 '
2000-Titel' Bezeichnung der Felder
2010 DATA "Name        : "
2020 DATA "Straße      : "
2030 DATA "PLZ, Ort    : "
2040 DATA "Bemerkungen: "
```

Roland Löhr

Datenübertragung

in S-Records

Datenübertragung zwischen Computern und Untersystemen ist ein aktuelles Thema. Es tritt auf, wenn ein neuer Computer in Betrieb genommen wird, der die Datenträger (Disketten) des Vorgängermodelles nicht lesen kann, oder wenn auf einem Entwicklungssystem ein Programm assembliert worden ist, das ein Zielsystem zur Erprobung in sein RAM laden oder das zu einem Eprommer übertragen werden soll, der Datenspeicher für ein Zielsystem brennt. Es ist das allgemeine Thema, daß ein vorhandener Computer Datenbestände oder Leistungsfähigkeiten besitzt, die für ein anderes System benutzt werden sollen.

Das vorliegende Programm ist vor allem zur Datenübertragung an Computer gedacht, die schon zum Brennen von Eproms vorbereitet sind, oder an entsprechende Eprommer, die S-Records lesen und auswerten können. Der allgemeine meist unproblematische Lösungsweg zur Übertragung besteht in der Geräteverbindung über eine serielle Schnittstelle RS232C, die auch hier benutzt wird. - Das Programm ist in BASIC geschrieben.

Das Programm ist ein Anwendungsbeispiel für das OMIKRON-BASIC auf dem Atari ST. Dieses BASIC wird an anderer Stelle im laufenden Heft besprochen. Es entspricht dem MBASIC und setzt nur voraus, daß in dieser Sprache die serielle Schnittstelle für die Ausgabe eröffnet werden kann. Das geschieht hier in der Programmzeile 340. Diese Möglichkeit sollte man auf allen derzeit in Auslieferung befindlichen modernen Computern vorfinden.

Für Benutzer eines älteren BASIC-Dialektes folgende Anmerkungen: Namen von Variablen, Unterprogrammen und Prozeduren sind nicht auf 2 alfanumerische Zeichen plus Typbezeichner beschränkt, die man früher z.B noch schreiben mußte als AA=.. oder A$="... . Sie nehmen jetzt eine die Tätigkeit ausreichend kennzeichnende Textkette unterschiedlicher Länge ein. Sprünge zu Unterprogrammen bei GOTO und GOSUB dürfen den Namen eines Labels benutzen. Wenn man schreibt Xout(-) oder To_Check, dann wird damit eine Prozedur aufgerufen, die in ihrer Wirkung einem Unterprogramm-Aufruf mit Parameterübergabe vergleichbar ist.

In der nachstehenden Liste finden wir einige Male die Funktion INPUT USING (mit Kontrollstring zum Filtern der Eingabe auf erlaubte Zeichen) und HEX$(). Die zweitgenannte Funktion hat man auch schon auf Heimcomputern, wie dem PC-128 mit BASIC.7 (Umwandlung einer Zahl in einen hexadezimalen String, der dann zur Aussendung kommt). Wenn man die Filterung der Eingabe und die Umwandlung zur Ausgabe beachtet, kann man mit älteren BASIC-Dialekten gleiches wie hier erreichen.

Das Prinzip der S-Records

S-Records sind ein Datenübertragungs-Format auf Motorola-Computern. Man findet es auch auf CP/M-68K-Computern im Programm SENDC68.REL von DRI, das der Autor allerdings nicht im Entwicklungspaket für den Atari ST vorfand (stattdessen ist dort KERMIT mit anderem Protokoll). Man darf wohl behaupten, daß diese Art der Datenübertragung zwar ein Protokoll hat, jedoch keine Vorsehung für eine Korrektur bei falsch empfangenen Daten. Der Falschempfang im seriellen Verkehr wurde hier allerdings noch nicht beobachtet, solange man ohne aktive induktive

Störer in der Umgebung auf dem gleichen Arbeitstisch Übertragungen machte. Einzige Möglichkeit zur Erkennung von Übertragungsfehlern ist die Kontrollsumme, die im Programm Checksum genannnt wurde, z.B. Zeilen 500 und 900. Der Empfänger kann nur an der Kontrollsumme feststellen, daß etwas schief ging, er kann aber nicht zur korrigierenden Wiederholung aufrufen, die man ggfs. manuell auslösen muß.

Das Thema der S-Records wurde für 65xx-Computer bereits in Heft 44 mit einem Assembler-Programm angesprochen. Für sie haben wir das allgemeine Übertragungsformat:

S0-Record - Sx-Record(s) - S(10-x)-Record

Der S0-Record signalisiert dabei, daß die Übertragung beginnt. Er darf optional auch einen Dateinamen enthalten. Man wird sowohl diesen Record-Typ wie auch den Namen ggfs. fortlassen können (abhängig vom Zielsystem). Die Sx-Records enthalten die zu übertragenden Daten. Folgende Typen, die man im

nachfolgenden Programm in Zeile 160 auswählen kann, kommen zum Einsatz: S1-Records mit einer Adresse von 16 Bit, S2-Records und S3-Records mit Adreßvorgaben von 24 bzw. 32 Bits. - Der Record-Typ S(10-x) läßt das Ende einer Übertragung erkennen: Ein S9-Record beschließt eine Kette von S1-Records, S8- und S7-Records beschließen eine Kette von S2- bzw. S3-Records. In Zeile 200 wird errechnet, welcher Typ die Übertragung beenden muß, nachdem die Wahl in 160 erfolgte.

Vor jedem einzelnen Record sendet man zweckmäßig die hexadezimale Folge $0D, $0A, $00, was Carriage Return, Linefeed und Zeilentrenner bedeutet (Variable Linfed$ in Zeile 330). Im Einzelfall wird man darauf verzichten können. Nur: Die Zeilentrennung ist zweckmäßig, wenn man das Empfangene auf dem Zielsystem zur Anzeige bringen möchte. Es hängt weiterhin von der Bearbeitungszeit im Zielsystem ab, ob man zwischen die Records eine kleine Übertragungspause einlegt. Wenn das Empfangene nur decodiert und in den Speicher transportiert wird, kann man darauf verzichten. Wenn es auf dem Bildschirm zur Anzeige gebracht werden soll, dann macht man häufig die Beobachtung, daß der Empfänger wegen des Scrollings in zeitlichen Verzug gerät und die Anzeige verwürfelt. In noch stärkerem Maße wird das bei unmittelbarer Abspeicherung auf Floppy eintreten, weil die Zugriffszeiten noch länger sind.

Zugelassene Zeichen: Wenn man einmal von den möglichen eben erwähnten nicht druckbaren Zeichen für den Zeilenvorschub absieht, werden in den S-Records nur die hexadezimalen Ziffern 0...9 und A...F sowie der Buchstabe S benutzt. Die beiden ersten Zeichen eines S-Records sind der Buchstabe S sowie eine der Ziffern 0...3 oder 9...7. Diese beiden ersten Zeichen werden ganz normal als ASCII übertragen.

Alle darauf folgenden Zeichen sind ASCII-codierte Hexadezimalziffern, wobei jedes Byte in der Reihenfolge High/Low in der Form von 2 Hexadezimalziffern (als ein "Paar") gesendet wird. Eine gleichartige Zerlegung von Bytes findet sich in jedem Monitorprogramm zur Anzeige von Speicherinhalten. Solche Routinen nennen sich NUMA, WROB, HEXOUT etc.

Auf den Header Sx folgen 1 Byte der Länge, 2-4 Bytes mit der Startadresse für die nachfolgenden Daten, n Datenbytes und 1 Byte mit einer Kontrollsumme (Checksum). Jedes Byte wird dabei, wie erwähnt, als 1 Paar von Hexziffern gesendet. Das Byte der Länge kann als Höchstwert 255 enthalten. Wir gehen im normalen S3-Record bis 21 ($15), und zwar für 4 Adreß-Bytes, 16 Datenbytes + 1 Byte der Kontrollsumme. Wenn die Zahl der zu sendenden Datenbytes kein Vielfaches von 16 ist, wird der Überhang in einem entsprechend kürzeren Record des gewählten Typs übertragen.

Im S0-Record wird als Adresse 0000 gewählt, ebenso im abschließenden Record. Die Kontrollsumme bezieht sich auf die Bytes der Felder Länge, Adresse und Daten. Dabei werden die Hexwerte der Halbbytes ohne Übertrag addiert. Verwertet wird das niederwertige Byte der Kontrollsumme, das für die Ausgabe bitweise invertiert wird (XOR).

Es ist darauf aufmerksam zu machen, daß offensichtlich auch andere Formate für S-Records benutzt werden. So mögen die Adressen relativ auf den Start (=00000) berechnet sein. Auch bei der Bildung der Kontrollsumme mag ein anderer Algorithmus benutzt sein. Man vergewissere sich im Einzelfall und schreibe ggfs. das nachfolgende Programm etwas um.

Zum Programm: Es ist typisch für Entwicklungszwecke geschrieben: Ein auf dem sendenden Rechner entwickeltes Programm soll auf einen Eprommer oder auf einen

Zielrechner übertragen werden. Natürlich kann man auch reine Daten übertragen. Dazu muß man wissen, in welchem Bereich Von...Bis die zu sendenden Daten stehen. Das wird in den Zeilen 210 und 250 abgefragt. Da dieser Adreßbereich keineswegs mit dem auf dem Zielsystem angestrebten identisch sein muß, kann in Zeile 240 eine virtuelle Adresse für das Zielsystem bestimmt werden, die in den S-Records im Adreßfeld zur Aussendung kommt. Wenn man hier z.B. 0000 angibt, so beziehen sich alle Angaben, wie oben erwähnt, auf den Start der Übertragung.

In Zeile 320 wird die Zahl der Blocks mit je 16 Datenbytes berechnet und auch der Rest, der in einem letzten Record zu senden ist. - Die serielle Datei wird in Zeile 340 geöffnet. Zeile 370 ruft die Prozedur Send_Blocks (in Zeile 490) entsprechend häufig auf. In Zeile 390 wird ggfs. der Rest mit Aufruf der Prozedur Send_Rest gesendet. - Die Prozeduren ab Zeile 730 unterstützen die beiden vorerwähnten mit wiederkehrenden Diensten für die Ausgabe der Adresse, von Bytes, für die Bildung und Ausgabe der Kontrollsumme.

Ein Hinweis für die Übertragung von Dateien, die sich auf der Floppy befinden: Wenn es sich um ASCII-Dateien handelt, kann man ihren Inhalt mit INPUT' in Variablen einlesen und deren Inhalt absenden. Binärdateien können mit INPUT' nicht gelesen werden. Für Sie empfiehlt es sich, sie mit BLOAD an eine definierte Adresse zu laden und ihre Länge mit der Funktion LOF festzustellen, um die Endadresse der Übertragung bestimmen zu können. Alternativ könnte man natürlich auch in Assembler programmieren.

Das nachstehende Programm läßt sich leicht für einen Memory-Dump umfunktionieren: Man benutzt statt PRINT' (Ausgabe auf logische Datei) einfach den Befehl PRINT für den Bildschirm oder LPRINT für den Drucker. Dabei wird man etwas umformatieren und zwischen die einzelnen Felder Zwischenraumzeichen setzen.

```
100 ' Programm zum Senden von S-records
110 ' by Roland Löhr, Mai 1987
120 '
130 ' Abfrage der Parameter: Typ, Anfang, virtuelle Empfangsadresse, Ende
140-Start_Mes: CLS : PRINT " Bitte wählen Sie den Typ der S-Records"
150         PRINT " S1= Adresse 16 Bit, S2= Adresse 24 Bit, S3= Adresse 32 Bit"
160-Styp: INPUT " Bitte S1 S2 oder S3 eingeben    ";Typ$ USING "+S +1 +2 +3",,2
170         IF LEFT$(Typ$,1)<>"S" THEN PRINT : GOTO Styp
180 Leng$= RIGHT$(Typ$,1)' Typ isoliert
190 Leng=2+ VAL(Leng$)*2' Länge der zu sendenden Adresse
200 End_Typ=10- VAL(Leng$):End_Typ$= RIGHT$( STR$(End_Typ),1)
210         PRINT : INPUT " Ab Startadresse im Atari  $";Start$ USING "0 +a +b +c
 +d +e +f",,8' maximal 8 Hexzeichen zugelassen
220         PRINT : INPUT " Ist die Startadresse auf dem Zielsystem davon verschi
eden? j/n    ";Offs$ USING "+j +n",,1
230         IF Offs$="n" THEN GOTO Send
240         PRINT : INPUT " Was ist die virtuelle Zieladresse? $";Ziel$ USING "0
+a +b +c +d +e +f",,Leng:Ziel= VAL("$"+Ziel$)
250-Send: PRINT : INPUT " Bis einschließlich Endadresse im Atari $";Ende$ USING
"0 +a +b +c +d +e +f",,8
260 ' Übernahme der Parameter
270         Start= VAL("$"+Start$):Ende= VAL("$"+Ende$)'Start und Ende numerisch
machen
280         IF(Start=Ende OR Start>Ende) THEN PRINT : PRINT "Fehler in den Vorgab
en": END
290       PRINT : INPUT " Ist das richtig? j/n ";Antwort$ USING "+j +n",,1: IF A
ntwort$="n" THEN PRINT : GOTO Styp
300 Virt_Pc=Start: IF Offs$="j" THEN Virt_Pc=Ziel
310 '
320 Blocks= INT((Ende+1-Start)/16):Rest=(Ende+1-Start) MOD 16'Blöcke zu 16 Bytes
 + Rest
330 Linfed$= CHR$($D)+ CHR$($A)+ CHR$(0)' benötigt am Start von Records
340-Open_Ser: OPEN "V",1' öffne serielle Schnittstelle"
350 PRINT : PRINT "Blöcke",Blocks,"Rest",Rest
```

```
360 PRINT #1,Linfed$+"S0030000FC";' Sende Record S0
370-Sendb: IF Blocks>=1 THEN Send_Blocks:Blocks=Blocks-1: GOTO Sendb' Blöcke mit
 je 16 Datenbytes senden
380 '
390 IF Rest<>0 THEN Send_Rest' Rest der Daten senden, 1 Rumpfblock
400 '
410-Last_Rec: PRINT #1,Linfed$+"S"+End_Typ$+"030000FC";' Sende Record S-Typ
420 PRINT #1,Linfed$
430 CLOSE 1
440 PRINT : PRINT " Die Übertragung ist beendet"
450 PRINT " Nächste Adresse im Atari ist                $"; HEX$(Start)
460 PRINT " Nächste virtuelle Adresse im Empfänger ist $";Virt_Pc$
470 END
480 '
490 DEF PROC Send_Blocks' Blöcke mit je 16 Datenbytes senden
500 Checksum=0
510 PRINT #1,Linfed$+Typ$;' das ist der Vorspann des Satzes
520 X=16+1+Leng/2:Xout(X)'Länge als 2 Byte ausgeben Daten, Adresse, Checksum
530 To_Check' addiere zur Kontrollsumme
540 Astring: PRINT #1,Virt_Pc$;' Adresse ausgeben
550 Byte$=Virt_Pc$:To_Check
560 FOR I=0 TO 15:X= PEEK(Start+I):Xout(X):To_Check: NEXT
570 Checks_Out' Ausgabe Kontrollsumme für den laufenden Block
580 Start=Start+16:Virt_Pc=Virt_Pc+16:Virt_Pc$= HEX$(Virt_Pc)' Zähler erhöhen
590 RETURN
600 '
610 DEF PROC Send_Rest' Rest der Bytes senden, die keinen Block mehr füllen
620 Checksum=0
630 PRINT #1,Linfed$+Typ$;
640 X=Rest+1+Leng/2:Xout(X)' Länge ausgeben
650 To_Check' zur Kontrollsumme addieren
660 Astring: PRINT #1,Virt_Pc$;' Adresse ausgeben
670 Byte$=Virt_Pc$:To_Check
680 FOR I=0 TO Rest-1:X= PEEK(Virt_Pc+I):Xout(X):To_Check: NEXT
690 Checks_Out
700 Start=Start+Rest:Virt_Pc=Virt_Pc+Rest:Virt_Pc$= HEX$(Virt_Pc)' Zähler erhöhe
n
710 RETURN
720 '
730 DEF PROC Astring:Virt_Pc$= MID$( HEX$(Virt_Pc),2)'Bilde Ausgabestring der Ad
resse
740 IF Leng= LEN(Virt_Pc$) THEN RETURN
750 IF Leng< LEN(Virt_Pc$) THEN Virt_Pc$= RIGHT$(Virt_Pc$,Leng)
760 L_Diff=Leng- LEN(Virt_Pc$):Virt_Pc$="0"*L_Diff+Virt_Pc$' Stringmultiplikatio
n:return
770 RETURN
780 '
790 DEF PROC Xout(X)' 1 Byte als 2 ASCII-codierte Bytes ausgeben
800 Byte$= MID$( HEX$(X),2): IF LEN(Byte$)=1 THEN Byte$="0"+Byte$' bei Werten 0-
15
810 PRINT #1,Byte$;
820 RETURN
830 '
840 DEF PROC To_Check'Addiere zu Checksum
850 FOR J=1 TO LEN(Byte$):Lo$= MID$(Byte$,J,1):Lo= ASC(Lo$) AND $F' Mask off
860 IF Lo$>="A" THEN Lo=Lo+9' Sprung der ASCII-Codierung
870 Checksum=Checksum+Lo: NEXT J
880 RETURN
890 '
900 DEF PROC Checks_Out' Gib Checksummme invertiert aus
910 Checksum=Checksum AND $FF' maskiere low byte
920 X=Checksum XOR $FF:Xout(X)
930 RETURN
```

Hier zunächst ein üblicher Memory-Dump vom Beginn des TOS-ROMs:

```
FC0000 60 1E 01 00 00 FC 00 20 00 FC 00 00 00 00 61 00
FC0010 00 FC 00 20 00 FE FF F4 02 06 19 86 00 03 0C 46
FC0020 46 FC 27 00 4E 70 0C B9 FA 52 23 5F 00 FA 00 00
```

Bytes vom Beginn des TOS-ROMs als S-Records:

```
S0030000FC
S214FC0000601E010000FC002000FC000000061008A
S214FC001000FC002000FEFFF40206198600030C463A
S207FC002046FC27AD
S8030000FC
```

○□○

Roland Löhr

Maskengenerator

Im Vergleich von Datenbanken berichteten wir in Heft 48, wie umständlich in dBase eine Maske mit zahlreichen Programmzeilen generiert werden muß, wenn kein Maskengenerator zur Verfügung steht. Demgegenüber läßt Superbase für CBM den komfortablen Entwurf einer Maske am Bildschirm zu. Superbase war damit Vorbild für den nachfolgenden Maskengenerator auf dem Atari. Er soll dem Leser als ein Systembaustein bei der Erledigung ähnlicher Aufgaben dienen.

Die Grundidee

Unter TOS kann der Rechner wie ein Terminal betrieben werden. Die Position des Cursors und von Schreibstellen wird dabei durch Escape-Sequenzen bestimmt. Man muß also nicht immer den gesamten Bildschirm löschen und zeitraubend neu aufbauen, wenn man bei gleichbleibender Maske z.B. einen neuen Datensatz in einer Datenbank zur Anzeige bringen will. Insofern kann das Programm auch für andere Computer interessant sein, bei denen eine Art Terminalbetrieb möglich ist.

Felder innerhalb einer Maske werden zwischen den Zeichenpaaren spitze Klammer auf und zu definiert. Am Ende einer Editierung merkt sich das System in einem Zeigerarray (FELDPARM), in welcher Zeile und Spalte die Felder beginnen. Diese Parameter können später zum gezielten Schreiben in Felder benutzt werden. Es ist aber auch möglich, vollkommen ohne solche Feldbezeichner zu arbeiten. Mit dem Befehl ´l´ (laden) und einem Dateinamen kann man stattdessen z.B. auch eine Hilfetafel zur Benutzerführung zur Anzeige bringen.

Wenn man bei Editieren mit dem Cursor frei über den Bildschirm fahren will, dann muß man seine Bewegungen verfolgen und in laufende Parameter von Zeile und Spalte umsetzen. Ein guter Teil des Editors ist mit dieser Verfolgung beschäftigt. Was beim Editieren als Text eingetippt wird, gelangt zur Zwischenspeicherung in einen Speicherbereich mit Namen BILD.

Die Funktionen

Das Programm startet mit der Anzeige eines Menüs, von dem aus folgende Hauptfunktionen mit Tastendruck aufgerufen werden können: Erstmaliges Anlegen (Editieren) einer Maske, Wiederanzeige mit Angabe der jeweiligen Feldbreiten zwischen den spitzen Klammern, Reformatieren (neue Feldbestimmung bei vorhandener Maske), Laden einer Maske von Floppy nebst Anzeige und Abspeichern dorthin.

Editieren

Am Label MASKE0 wird der Bildspeicher zunächst mit Spaces gefüllt und der Cursor in die Stellung Home gebracht. Die linke obere Ecke ist also Bezugspunkt. Ab MKEY wird die Betätigung von Tasten beobachtet und verwertet. Dabei ist zu beachten, daß Cursor-Tasten keinen ASCII-Code abgeben, sondern einen Scancode (wie bei IBM), der einer Tastennummer entspricht. Druckbare Zeichen werden in den Speicher BILD gebracht und andere verwertbare Zeichen in

Bewegungen des Cursors innerhalb der Bildgrenzen umgesetzt. Das Editieren wird mit CTRL-c abgeschlossen. Es folgen die Auswertung des Bildes und die Wiederanzeige des Menüs.

Auswertung des Bildes
Im Sinne eines schnellen Bildaufbaues bei der Wiederanzeige wird das Abbild des Bildschirmes in 25 Zeilen-Variablen gebracht (Z0...Z24), wobei unnötige Leerstellen am Ende einer Zeile entfernt werden. Hinter das letzte druckbare Zeichen einer Zeile wird daher in den Variablen eine hexadezimale Null als Zeilenbegrenzer eingeschrieben. Die Zeilen sind also von hinten her zu prüfen. Die entsprechenden Schritte folgen auf das Label STORE. Bei STORE4 wird nun von links oben nach rechts unten nach spitzen Klammern gesucht und in der Variablen FELDDES Buch geführt, wieviele Felder es gibt.

Eine sich öffnende spitze Klammer (Feldbeginn-1) wird wie folgt für die Ablage einer Escape-Sequenz verwertet: Register D2 ist Zeilenzeiger und D1 verwaltet die Spalte. Es hat nun folgende Eigenheit im Terminalbetrieb mit der späteren Escape-Sequenz ESC Y Zeile Spalte: Eine hexadezimale 20 (ein Space) bedeutet Zeile oder Spalte 0, ein $21 dann 1 usw. Daher die entsprechende Vorladung der Register D2 und D1 bei Label FELDSU. Entsprechend codierte Feldbeginne und die Längen werden in das Array FELDPARM übernommen.

Im Programm werden nur Paare spitzer Klammern als Feldbegrenzer benutzt. Es ist dem Leser unbenommen, auch andere Zeichen zu verwenden, um z.B. den Typ eines Feldes zu bestimmen.

Wiederanzeige
Das Unterprogramm ANZEIGE bringt die Maske, die in den Variablen für die Zeilen gespeichert ist, nebst spitzen Klammern zur Anzeige. Interessant wird es dann beim Label SETZFELD: Mit der Ausgabe der gespeicherten Escape-Sequenzen wird die Schreibposition der Felder angesteuert und dann die Länge der Felder dezimal berechnet und zwischen den spitzen Klammern zur Anzeige gebracht. Nach dem gleichen Prinzip steuert man dann später die einzelnen Schreibfelder z.B. einer Datenbank an. Der Parameter der Länge dient dann der Kontrolle, daß man nicht die Feldgrenzen verläßt.

Das gezielte Schreiben in Felder beschleunigt das Blättern in einer Datenbank erheblich. Man muß dann jedoch nach dem gleichen Prinzip dafür sorgen, daß ein Anzeigefeld zunächst zu Spaces gelöscht wird, ehe man es neu beschreibt, es sei denn, man füllt ab erreichter Schreibposition mit Spaces auf oder die Felder eines Datensatzes haben trailing spaces (angehängte Zwischenraumzeichen).

Abspeichern und Laden
Für die beiden bei MASKELAD und SPEICHERN verwirklichten Funktionen sind zahlreiche GEMDOS-Funktionen für die Dateibearbeitung benutzt und in das Programm eingebunden wurden. Für die Ablage einer Maske wird eine Datei angelegt. Für das Lesen wird nach einer Datei gesucht. Wenn gefunden, dann wird sie geöffnet. Zur Abspeicherung gelangen das Bild, die Felddeskriptoren und auch die Zeilen-Variablen. Wenn man nur Hilfetafeln benutzen will, reicht die Abspeicherung des Bildes, auf das eine Null folgt. Man bringt es dann mit einem einzigen Befehl PRINTLINE zur Wiederanzeige.

```
* MASKE1.ASM
* Maskenerzeugung am Bildschirm
* Datenfelder in spitze Klammern einfassen < >
* Copyright by Roland Löhr, 3.8.86

 SECTION  ONE
```

```
* Verkehrszeichen
space     equ       $20             Definition Zwischenraum
cr        equ       $0d
lf        equ       $0a
escape    equ       $1b
GEMDOS    equ       1

* Hauptprogramm, zunächst Löschung des Screen-Abbildes im Speicher

maske     movem.l   a0-a6/d1-d7,-(sp)   Register sichern
menue     move.l    #mtext,d0       Menue anzeigen
          bsr       printline
menue1    bsr       dircon          Befehlstaste holen
          move.b    d0,befehl       Befehl merken
          bsr       clearscr        Clear Screen
          cmpi.b    #escape,befehl  Abbruch, Befehlsverteiler
          beq       schluss         ja
* Befehle decodieren
          cmpi.b    #'e',befehl     Maske erstmalig anlegen?
          beq       maske0          ja
          cmpi.b    #'l',befehl     Maske laden und neu formatieren?
          beq       maskelad        ja
          cmpi.b    #'w',befehl     Wiederanzeige?
          beq       wanzeige
          cmpi.b    #'s',befehl     Speicherung auf Diskette?
          beq       speichern
          cmpi.b    #'r',befehl     reformatieren?
          beq       reform
          bra       menue1          Keine erlaubte Taste

*****************************************************************

* Eine neue Maske am Bildschirm anlegen, Cursor-Tasten aktiv

maske0    clr.b     bildlim         Delimiter hinter das Bild setzen
          lea.l     bild,a1         Setze Zeiger auf Anfang der Maske
          move.w    #499,d1         500 Durchgänge=25 Zeilen
          move.l    #$20202020,d0   das sind 4 Spaces
maske1    move.l    d0,(a1)+        in das Bild einfüllen
          dbra      d1,maske1       Schleife, bis Bild gelöscht

home      lea.l     bild,a1         Setze auf Anfang des Abbildes
          move.b    #escape,d0      Terminalbefehle: Escape-Sequenz
bsr       conout
          move.b    #'H',d0         Cursor Home
          bsr       conout
          move.b    #escape,d0
          bsr       conout
          move.b    #'w',d0         kein fortlaufendes Schreiben
          bsr       conout

* Zeichen holen, in Cursorbewegungen umsetzen bzw. einschreiben
* A1 ist Zeiger auf das entstehende Bild

mkey      bsr       dircon          Zeichen holen per GEMDOS
          cmpi.b    #$03,d0         Ist es CTRL-C ?
          beq       store           ja, dann Abbruch der Eingabe
          swap      d0              Zeichen sichern, Scancode aktivieren
* Bearbeitung der Scancodes
          cmpi.b    #$53,d0         Delete?
          beq       mkey            nicht akzeptieren
* Cursor-Tasten in Aktionen umsetzen
          cmpi.b    #$47,d0         Clear/Home
          beq       home
          cmpi.b    #$4b,d0         Cursor left
          beq       curleft
          cmpi.b    #$4d,d0         Cursor right
          beq       curright
```

```
          cmpi.b    #$50,d0         Cursor down
          beq       curdown
          cmpi.b    #$48,d0         Cursor up
          beq       curup

* Scancodes soweit erledigt, sonstige Sonderzeichen prüfen
          swap      d0              ASCII-Teil wieder aktiv
          cmpi.b    #$1b,d0         Escape
          beq       doescape        noch ohne weitere Funktionen
          cmpi.b    #$0d,d0         Return?
          beq       return
          cmpi.b    #$08,d0         Backspace
          beq       curleft         delete
          cmpi.b    #' ',d0         sonst. Kontrollzeichen?
          bcs       mkey            nicht akzeptieren
          cmpa.l    #bildlim,a1     allerletzte Position?
          bcc       mkey            ja, nichts mehr annehmen

          move.b    d0,(a1)+        Ablage d. empfangenen Zeichens
mkey2     bsr       conout          Ausgabe im Echo
          bra.s     mkey            weiter in der Eingabe-Schleife

store     move.b    #escape,d0      Es lag CTRL-C vor, nun Bild auswerten
          bsr       conout
          move.b    #'E',d0
          bsr       conout          Clear Screen

* Trailing Spaces aus den Zeilen entfernen
* A1 ist Zeiger auf den Bildspeicher bild
* A2 ist Zeiger auf die Zeilen-Variablen z0...z24,
* in die umgespeichert wird
          move.b    #space,d0       Operand zum Vergleich
          lea.l     bildlim-1,a1    Zeiger auf Ende Screen
          lea.l     z25-1,a2        Zeiger auf letzte Zeile, hinten

          move.w    #24,d2          Aussenschleife 25x

* Nun das Bild für Printline in 25 Zeilen aufteilen, arbeite rückwärts
store00   move.w    #79,d1          Zähler Innenschleife
          clr.b     (a2)            angehängte Null, Delimiter in 81. Stelle
store0    cmp.b     (a1),d0         Spaces in laufender Zeile abtrennen
          bne       store1          nicht mehr Space
          clr.b     -(a2)           Space wird zu Delimiter nach hinten
          suba.l    #1,a1           auf das Zeichen davor positionieren
          dbra      d1,store0       Innenschleife fortsetzen
          tst.w     d1
          bmi       store2          alles schon gefüllt, Leerzeile
* Gültige Zeichen der Bildzeile in Variablen übertragen
store1    move.b    (a1),-(a2)      Rest der Zeile übertragen
          suba.l    #1,a1           in die Variablen z0...z24
          dbra      d1,store1       Innenschleife

store2    suba.l    #1,a2           Zeilenzeiger nachziehen
          dbra      d2,store00      Aussenschleife, nächste Zeile

* alle Bildzeilen sind nun übertragen
* nun Deskriptoren für die Felder in spitzen Klammern errechnen

          clr.w     felddes         0 Felder bisher erkannt
          move.w    #74,d1
          lea.l     feldparm,a1     lösche die 75 möglichen Feldparameter
store4    clr.l     (a1)+           je 1 Langwort als Deskriptor
          dbra      d1,store4

* Nun spitze Klammern als Feldbegrenzer suchen ordnen und merken
          move.b    #'<',d0         Zeichen für Feldbeginn zum Vergleich
          lea.l     bild,a1         Bildanfang nach A1 laden
          lea.l     feldparm,a2     Zeiger auf zu füllende Tabelle
```

```
          clr.w     feldflag        0 = ausserhalb eines Feldes
* Feldparameter als Escape-Sequenzen anlegen
* D2 nimmt Zeichen f. Zeile auf, D1 für Spalte
          move.b    #' ',d2         Aussenschleife, Zeile=0
feldsu    move.b    #' ',d1         Innenschleife, Zeichen f. Spalte 0
feldsu1   cmp.b     (a1)+,d0        Feldbeginn rechts davon?
          beq.s     feldsu2         spitze Klammer < angetroffen
          addq.b    #1,d1           Zeichen für nächste Spalte
feldsu1a  cmpi.b    #$70,d1         Zeilenende?
          bcs.s     feldsu1         nein
          addq.b    #1,d2           Zeichen für nächste Zeile
          cmpi.b    #$39,d2         höchste Zeile?
          bcs.s     feldsu          nein, weiter in Aussenschleife
          bra       menue           Bild ist ganz erledigt

* Verarbeitung der spitzen Klammer <
feldsu2   addq.b    #1,d1           erhöhe Spaltenzeichen
          tst.w     feldflag        sind wir im offenen Feld?
          bmi.s     feldsu4         ja
          move.w    #$ffff,feldflag  invertieren = Feld jetzt offen
          move.b    d2,(a2)         Zeilen-Kennzeichen für Escape ...
          move.b    d1,1(a2)        und Spalten-Kennzeichen merken

          addq.w    #1,felddes      Descriptor+1, Menge erhöhen
          move.l    a1,a3           Merke den Feldbeginn in A3
          move.b    #'>',d0         jetzt rechten Begrenzer > suchen
          bra       feldsu1

feldsu4   clr.w     feldflag
          move.l    a1,d0           Zeiger+1 ins Bild
          subq.l    #1,d0           Adresse-1 von '>'
          sub.l     a3,d0           Differenz = Länge
          move.w    d0,2(a2)        in die Tabelle feldparm schreiben
          adda.l    #4,a2           Zeiger auf Deskriptoren +4
          move.b    #'<',d0         nun wieder linken Begrenzer suchen
          bra       feldsu1a

* Wiederanzeige einer Maske mit Angabe der Feldbreite in den Klammern
* Befehlstaste w

wanzeige  bsr       anzeige         25 Zeilen zeigen
          lea       feldparm,a5     setze ins 1. Fenster
wanz1a    tst.l     (a5)
          beq.s     wanz2           Kein weiteres Fenster
          bsr       setzfeld
          bra.s     wanz1a
wanz2     bsr       dircon          warten
          bra       menue

* Upro zur Ansteuerung der Schreibpositionen, Anzeige der Feldbreite

setzfeld  move.b    #escape,d0      setze ins Datenfeld
          bsr       conout          per Escape-Sequenz
          move.b    #'Y',d0         ESC+Y = Cursor setzen
          bsr       conout
          move.b    (a5)+,d0        Parameter der Ansteuerung Zeile,
          bsr       conout
          move.b    (a5)+,d0        Spalte ausgeben
          bsr       conout
          move.w    (a5)+,d1        merke Länge des Feldes
          bne.s     setzf2          Länge ungleich 0
          move.b    #'0',d0
          bsr       conout          Null ausgeben
setzf1    rts

setzf2    clr.w     produkt         Feldbreite dezimal errechnen
          cmpi.w    #1000,d1        mehr als 1000?
          bcs.s     setzf3          nein
```

```
          move.w    #$1000,produkt Ergebnis
          subi.w    #1000,d1       Rest bilden
setzf3    cmpi.w    #100,d1
          bcs.s     setzf5         kleiner 100
          move.w    #$0100,faktor
          lea       produkt+2,a2
          lea       faktor+2,a3    Zeiger vorbereiten
          move.w    #0,ccr         Ohne Uebertrag addieren
          abcd      -(a3),-(a2)
          abcd      -(a3),-(a2)
          subi.w    #100,d1        Rest vermindern
          bra.s     setzf3
setzf5    cmpi.w    #10,d1
          bcs.s     setzf6         kleiner 10
          move.w    #$0010,faktor
          lea       produkt+2,a2
          lea       faktor+2,a3    Zeiger vorbereiten
          move.w    #0,ccr         Ohne Uebertrag addieren
          abcd      -(a3),-(a2)
          abcd      -(a3),-(a2)
          subi.w    #10,d1         Rest vermindern
          bra.s     setzf5
setzf6    move.w    d1,faktor      Jetzt die Einer-Stelle
          beq.s     setzf8         kein Rest
          lea       produkt+2,a2
          lea       faktor+2,a3    Zeiger vorbereiten
          move.w    #0,ccr         Ohne Uebertrag addieren
          abcd      -(a3),-(a2)
          abcd      -(a3),-(a2)

setzf8    lea       produkt,a2
setzf9    move.b    (a2)+,d0       Jetzt Ausgabe ohne führende Nullen
          beq.s     setzf10        führende Nullen
          bsr       hexout         Feldlänge anzeigen
setzf10   move.b    (a2),d0
          cmpi.b    #$10,d0        ggfs. nur 1 Ziffer ausgeben
          bcs.s     setzf11        ja
          bra       hexout         rts dort
setzf11   bra       hexout1        rts dort

* Programm abbrechen, zurück zu TOS
schluss   move.l    #textt,d0      Frage, ob wirklich Abbruch
          bsr       printline
          bsr       dircon         Tastaturabfrage
          cmpi.b    #'j',d0
          bne       menue
          movem.l   (sp)+,a0-a6/d1-d7   Register zurückgeben
term      clr.w     -(sp)          GEMDOS-Funktion 0
          trap      #GEMDOS        Rückkehr zum Betriebssystem

*********************************
doescape  bra       mkey           vorläufig, noch keine Funktion

* Lesen einer maske von Floppy und reformatieren
maskelad  bsr       getfname       Lese Datei mit Maske von der Floppy
          beq       menue          Abbruch mit Escape
          bsr       crlf
          bsr       getdta         alten Zeiger sichern
          bsr       setdta         neuen Zeiger setzen
          lea       filename,a5    Auf den Namen zeigen
          bsr       sfirst         File aufsuchen
          beq.s     lesen1         Datei bekannt
          move.l    #textc,d0      Datei nicht vorhanden
          bsr       printline      Textausgabe
          bsr       dircon         Bild stehen lassen
          bra       menue
```

```
lesen1    move.l    #bild,bufptri  Eingabepuffer=Bildspeicher
          move.l    fsize,bufleni  Dateigrösse
          bsr       fopen
          bpl.s     lesen2
          move.l    #texto,d0      "Kann nicht öffnen"
          bsr       printline      ausgeben
          bsr       dircon         Bild stehen lassen
          bra       menue

lesen2    bsr       fread          D0.L returns # of bytes read or error
          bsr       fclose
          move.l    #olddta,-(sp)  alten Zeiger dta wieder einsetzen
          move.w    #$1a,-(sp)     Funktions-#
          trap      #GEMDOS        Aufruf
          addq.l    #6,sp          Stack berichtigen
reform    bsr       clearscr       Clear Screen
          bsr       anzeige        Bildschirm schreiben
          bra       home

* Umsetzung der Cursor-Bewegungen in die Verwaltung von A1 als Bildzeiger
curup     cmpa.l    #bild+80,a1    1. Zeile?
          bcs       mkey           nicht akzeptieren
          move.b    #escape,d0     ESC+A = Cursor up
          bsr       conout
          move.b    #'A',d0
          bsr       conout
          suba.l    #80,a1         Zeiger nachführen
          bra       mkey
curdown   cmpa.l    #bildlim-80,a1 letzte Zeile
          bcc.s     return0        nicht akzeptieren
          move.b    #escape,d0     ESCAPE+B ausgeben
          bsr       conout
          move.b    #'B',d0
          bsr       conout
          adda.l    #80,a1         Zeiger ins Bild +80
          bra       mkey
return    cmpa.l    #bildlim-80,a1 Return-Taste
          bcc.s     return0
          move.b    #lf,d0         Linefeed
          bsr       conout
return0   move.b    #cr,d0
          bsr       conout
          move.l    a1,d0          Zeiger weiterstellen
          subi.l    #bild,d0
          divu      #80,d0
          cmpi.w    #25,d0         letzte Zeile?
          bcc.s     return1        dann auf deren Anfang setzen
          addq.l    #1,d0

return1   swap      d0
          clr.w     d0
          swap      d0
          mulu      #80,d0
          move.l    d0,a1
          adda.l    #bild,a1       Anfang neue Zeile
          bra       mkey

curleft   cmpa.l    #bild,a1       untere Grenze erreicht?
          beq       mkey           ja
          bsr       moveleft
          bra       mkey
moveleft  move.b    #escape,d0     Escape-Sequenz
          bsr       conout
          move.b    #'D',d0
          bsr       conout
          suba.l    #1,a1
          rts
```

```
curright  cmpa.l    #bildlim-1,a1  rechte untere Ecke
          beq       mkey           ja
          bsr       moveright
          bra       mkey
moveright move.b    #escape,d0
          bsr       conout
          move.b    #'C',d0
          bsr       conout
          adda.l    #1,a1
          rts

************************* Routinen der Eingabe ******************

*     conin wartet und liest dann ein Zeichen von der Tastatur
*     und echot es auf den Bildschirm, das ASCII-Byte ist in D0.B
*     Im Low Byte des höherwertigen Wortes von D0.L ist der Scan-Code

conin     move.w    #1,-(sp)            Funktionsnummer
          trap      #GEMDOS             Aufruf
          addq.l    #2,sp               Stack berichtigen
          rts                           Jetzt D0.L oder D0.B auswerten

*     dircon liest ein Zeichen von der Tastatur
*     ohne Echo-Ausgabe auf den Bildschirm,
*     Control-Zeichen werden durchgelassen

dircon    move.w    #7,-(sp)            Funktionsnummer
dircon1   trap      #GEMDOS
          addq.l    #2,sp
          rts                           Nun Zeichen in D0 auswerten

* Eine Eingabezeile als 00-terminierten String holen
getline   suba.l    a4,a4          Setze A4 zu 00
getline2  bsr       conin
          cmp.b     #$0d,d0        Zeilenabschluss?
          beq.s     getlincr       ja
          cmp.b     #$08,d0        Backspace?
          bne.s     getline3       nein
          cmpa.l    #0,a4          Sind wir in 1. Spalte?
          beq.s     getline2       ja, dann Backspace ignorieren
          move.b    #' ',d0        Altes Zeichen mit Space
          bsr       conout         überschreiben
          move.b    #$08,d0        Cursor wieder auf alte
          bsr       conout         Schreibstelle setzen
          suba.l    #1,a5          Pointer berichtigen
          suba.l    #1,a4          Ebenso Zähler
          bra.s     getline2
getline3  bsr       druckbar       Zeichen Filtern
          bmi.s     getline2       Nicht druckbar
          move.b    d0,(a5)+       Abspeichern in den Buffer
          adda.l    #1,a4          Mitlaufender Zähler +1
          cmpa.l    #80,a4         Zeile voll?
          bcc.s     getlincr       ja, 80 Zeichen, Zwangsabbruch
          bra.s     getline2
getlincr  clr.b     (a5)           00 als Begrenzer in den Buffer
getlinez  rts

* DRUCKBAR ist ein Filter für die Zeichen Space...Tilde ($7e)
druckbar  cmpi.b    #$9f,d0        Rubout oder grösser?
          bcc.s     nothex         ja, nicht druckbar
          cmpi.b    #' ',d0        Space?
          bcs.s     nothex         kleiner, nicht druckbar
          clr.w     filtflag       =00
          rts
nothex    move.w    #$ffff,filtflag
          rts
```

```
********** Routinen der Ausgabe ****************************

*    conaut gibt ein in D0.W, Low Byte übergebenes Zeichen auf
*    die Standardausgabe (Bildschirm) aus, das High Byte sollte 00 sein.
conout    move.w    d0,-(sp)          Zeichen auf den Stack
          move.w    #2,-(sp)          Funktions-Nr.
conout1   trap      #GEMDOS           Aufruf
          addq.l    #4,sp
          rts

* Bildschirm aus den Variablen schreiben
anzeige   lea  z0,a5                Bildschirm schreiben
          move.w    #23,d3          24mal mit crlf
anz1      move.l    a5,d0
          bsr       writeln         Zeile zeigen +CR/LF
          adda.l    #81,a5
          dbra      d3,anz1
          move.l    a5,d0           25. Zeile ohne crlf
          bsr       printline
          rts
*    printline gibt einen String Byte für Byte auf die Standard-Ausgabe
*    aus, bis eine 00 als Begrenzer gefunden wird.
*    In D0.L ist die Adresse des Stringbeginnes zu übergeben

printline move.l    d0,-(sp)          Stringadresse
          move.w    #9,-(sp)     Funktion
          trap      #GEMDOS           Aufruf
          addq.l    #6,sp             Stack berichtigen
          rts                         Gffs. CR/LF nachschieben

*    crlf gibt ein CR und LF auf die Standard-Ausgabe (Bildschirm)
writeln   bsr       printline       Zeile mit crlf ausgeben
crlf      move.w    #$0d,d0         Das CR
          bsr       conout
          move.w    #$0a,d0         Das LF
          bra       conout          rts dort
clearscr  move.b    #escape,d0      Funktion Clear Screen
          bsr       conout
          move.b    #'E',d0
          bra       conout          rts dort

*         hexout gibt 1 Hexbyte als 2 ASCII-Zeichen aus
hexout    move.w    d0,-(sp)        Sichern einer Kopie von d0
          lsr.b     #4,d0           zuerst high byte isolieren
          bsr.s     hexout1         und ausgeben
          move.w    (sp)+,d0        Kopie holen
          andi.b    #$0f,d0         low Byte isolieren
hexout1   ori.w     #$30,d0         ASCII '0' hinzu-odern
          cmpi.b    #$3a,d0         grösser '9'?
          bcs.s     hexout2         nein
          addq.b     #7,d0          Korrektur für A-F
hexout2   bsr conout                Die eigentliche Ausgabe
          rts

********** Routinen für Floppy-Betrieb ******************
speichern bsr       gcreatrw        Lege Datei an
          bsr       crlf
          move.l    #bild,bufptro Zeige auf Pufferbeginn für fwrite
          move.l    #z25,d0         Ende des benutzten Speichers
          addq.l    #1,d0           z25 zeigte auf den Begrenzer
          sub.l     #bild,d0        minus Beginn-Adresse
          move.l    d0,bufleno      = Menge Bytes
          bsr       fwrite          GEMDOS: Schreiben auf Floppy
          bsr       fclose
          bra       menue           Nächster Befehl

* Hole Adresse des aktuellen DTA-Buffers
getdta    move.w    #$2f,-(sp)      Funktionsnummer
```

```
          trap      #GEMDOS         Aktuelle DTA holen
          addq.l    #2,sp           Stack berichtigen
          move.l    d0,olddta       Adresse aktueller DTA merken
          rts

* Definiere den DTA-Buffer für File-Parameter
setdta    move.l    #dtabuff,-(sp) Adresse 44 Byte DTA-Buffer übergeben
setdta1   move.w    #$1a,-(sp)      Funktions-#
          trap      #GEMDOS         Aufruf
          addq.l    #6,sp           Stack berichtigen

          rts

* GETFNAME fordert zur Eingabe eines Datei- oder Pfadnamens auf
getfname  bsr       crlf            Neue Zeile
          move.l    #text9,d0       Zeiger auf interaktiven Text setzen
          bsr       printline       Ausgabe per GEMDOS
          lea.l     filename,a5     Benutze Namens-Puffer für die Eingabe
          bsr       getline         1 Zeile holen
          cmpi.b    #$1b,d0         Wurde ESCAPE zum Abbruch gegeben?
          rts

* Lege ein Schreib-/Lesefile mit Namen an
gcreatrw  bsr       getfname        Dateinamen anfordern

* Lege neues File mit Namen und Attribut an
* Lösche ggfs. vorhandenes File gleichen Namens
createrw  move.w    #0,-(sp)        Setze Attribut Lesen/Schreiben
          bra.s     create1         Verzweige immer, kurz
createrd  move.w    #1,-(sp)        Setze File-Attribut nur Lesen
create1   move.l    #filename,-(sp) Zeiger auf zu verwendenden Dateinamen
          move.w    #$3c,-(sp)      Funktionsnummer
          trap      #GEMDOS         Aufruf
          addq.l    #8,sp           Stack berichtigen
          move.w    d0,handle       Kanal-# merken
          rts                       Fehler, wenn negativ
fopen     move.w    #2,-(sp)        Datei-Attribut R/W
          move.l    #filename,-(sp) Zeiger auf Dateinamen
          move.w    #$3d,-(sp)      Funktions-# FOPEN
          trap      #GEMDOS         Aufruf
          addq.l    #8,sp           Stack berichtigen
          move.w    d0,handle       Kanal-# merken
          tst.l     d0              Fehler, wenn negativ
          rts                       Zurück mit Status

* Schliesse Datei mit handle-#
fclose    move.w    handle,-(sp)    Kanal nennen
          move.w    #$3e,-(sp)      Funktionsnummer
          trap      #GEMDOS         Aufruf
          addq.l    #4,sp           Stack berichtigen
          tst.w     d0              Fehlerstatus, wenn negativ
          rts
*    Prüfen, ob eine Datei lt. FILENAME vorhanden ist
sfirst    clr.w     sattrib         Normale Files suchen
*    sfirst1 = Einsprung für anders vorbesetztes sattrib
sfirst1   ;bsr       setdta          Setze Disk Transfer Adresse
          move.w    sattrib,-(sp)   Attribut des Files
          move.l    #filename,-(sp) Suchname
          move.w    #$4e,-(sp)      Funktions-#
          trap      #GEMDOS         Aufruf
          addq.l    #8,sp           Stack berichtigen
          tst.w     d0              1=nichts gefunden, 0=etwas gefunden
          rts                       Parameter stehen in DTABUFF
*    FSEEK positioniert innerhalb einer Datei
*    in D0.L muss +- der Offset übergeben worden sein

fseekbeg  move.w    #0,-(sp)        Suche ab Beginn des Files
          bra.s     fseek1
fseekend  move.w    #2,-(sp)        Suche ab Ende des Files
```

```
          bra.s     fseek1
fseek     move.w    #1,-(sp)        Suche ab laufender Position
fseek1    move.l    d0,-(sp)        +- Offset zum Positionieren
          move.w    #$42,-(sp)      Funktions-#
          trap      #GEMDOS         Aufruf
          add.l     #10,sp          Stack berichtigen
          tst.w     d0              Fehler, wenn negativ
          rts

* Einlesen von einer Datei mit handle-# ab Adresse bufptri
* Die zu lesende Menge muss in BUFLENI übergeben sein
fread     move.l    bufptri,-(sp)   Adresse Eingabe-Buffer f. Floppy
          move.l    bufleni,-(sp)   Vorgabe der Menge Bytes
          move.w    handle,-(sp)    Kanal-Nr.
          move.w    #$3f,-(sp)      Funktions-# FREAD
          trap      #GEMDOS         Aufruf
          add.l     #12,sp          Stack berichtigen
          tst.w     d0              Fehlerstatus erfassen
          rts

* Schreiben auf eine Datei mit handle-# ab Adresse bufptro
* Die zu schreibende Menge muss in BUFLENO übergeben sein
fwrite    move.l    bufptro,-(sp)   Adresse Ausgabe-Buffer f. Floppy
          move.l    bufleno,-(sp)   Vorgabe der Menge Bytes
          move.w    handle,-(sp)    Kanal-Nr.
          move.w    #$40,-(sp)      Funktions-# FREAD
          trap      #GEMDOS         Aufruf
          add.l     #12,sp          Stack berichtigen
          tst.w     d0              Fehlerstatus erfassen
          rts

**********************************
* Interaktive Texte
mtext     dc.b      escape,'E'      Menue anzeigen, Clear Screen
          dc.b      ' Menue, Auswahl mit Tastendruck',cr,lf,cr,lf
          dc.b      ' e = Maske erstmalig anlegen',cr,lf
          dc.b      ' l = Maske laden und neu formatieren',cr,lf
          dc.b      ' r = Residente Maske reformatieren',cr,lf
          dc.b      ' w = Maske wieder anzeigen',cr,lf
          dc.b      ' s = Maske auf Floppy speichern',cr,lf
          dc.b      ' ESC = Programm abbrechen',cr,lf,0
text9     dc.b      '               Eingabe Filename/Pathname:    ',0
textc     dc.b      '               Dieser Datei-Name ist unbekannt'
          dc.b      ', eine Taste drücken',cr,lf,0
texto     dc.b      '               Kann die Datei nicht öffnen,'
          dc.b      ' eine Taste drücken',cr,lf,0
textt     dc.b      cr,lf,' Programmende ??? j/n',cr,lf,0

* Arbeitsspeicher
dumm0     ds.w      1               align to even address
feldflag  ds.w      2               Flag negativ, wenn wir im Feld sind

produkt   ds.w      1               Dezimales Ergebnis
faktor    ds.w      1               Rechenfeld
bild      ds.b      2000            Bild der Maske/einer Texttafel
bildlim   ds.b      2               Delimiter der Maske

felddes   ds.w      1               Nimmt Zahl der aktiven Feldparms auf
* Aufbau von Feldparm: 1. Wort = Escape-Sequenz, 2. Wort = Feldlänge
feldparm  ds.l      75              75 Datenfelder möglich
z0        ds.b      81              Zeilen-Variable
z1        ds.b      81              Zeilen-Variable
z2        ds.b      81              Zeilen-Variable
z3        ds.b      81              Zeilen-Variable
z4        ds.b      81              Zeilen-Variable
z5        ds.b      81              Zeilen-Variable
z6        ds.b      81              Zeilen-Variable
z7        ds.b      81              Zeilen-Variable
z8        ds.b      81              Zeilen-Variable
```

```
z9          ds.b      81              Zeilen-Variable
z10         ds.b      81              Zeilen-Variable
z11         ds.b      81              Zeilen-Variable
z12         ds.b      81              Zeilen-Variable
z13         ds.b      81              Zeilen-Variable
z14         ds.b      81              Zeilen-Variable
z15         ds.b      81              Zeilen-Variable
z16         ds.b      81              Zeilen-Variable
z17         ds.b      81              Zeilen-Variable
z18         ds.b      81              Zeilen-Variable
z19         ds.b      81              Zeilen-Variable
z20         ds.b      81              Zeilen-Variable
z21         ds.b      81              Zeilen-Variable
z22         ds.b      81              Zeilen-Variable
z23         ds.b      81              Zeilen-Variable
z24         ds.b      81              Zeilen-Variable
z25         ds.b      1               Delimiter
dumm2       ds.w      1               Dummy to even address
bufptri     ds.l      1               Zeiger auf aktuellen Eingabepuffer
bufptro     ds.l      1               Zeiger auf aktuellen Ausgabepuffer
bufleni     ds.l      1               Enthält Menge zu lesender Bytes
bufleno     ds.l      1               Enthält Menge zu schreibender Bytes
handle      ds.w      1               log. Kanal-Nummer
sattrib     ds.w      1               Datei-Attribut
filtflag    ds.w      1
inbuff      ds.b      80              Allg. Eingabepuffer

* dtabuff enthält 44 Bytes mit File-Angaben
dummy       ds.w      1               align to even address
olddta      ds.l      1               Zeiger auf alte dta
dtabuff     ds.b      21              Unbenutzte Bytes
fattrib     ds.b      1               File-Attribute
ftime       ds.b      2               Zeit der Dateierstellung
fdate       ds.b      2               Datum der Dateierstellung
fsize       ds.b      4               Dateigrösse, Langwort
fnamfound   ds.b      14              Gefundener Filename + Extension, Ende DTA

filename    ds.b      80              Filename-Arbeitsspeicher
befehl      ds.b      1               Empfangener Befehl

 END
```

Roland Löhr

Stringsuche 68000

In Heft 42 des MICRO MAG /1/ wurde der schnelle Algorithmus von Boyer und Moore zur Stringsuche für den 6502 vorgestellt, und zwar speziell für den AIM 65. In Heft 45 wurde er auch für den CBM 710 in den dort abgedruckten einfachen Texteditor integriert. Er hat sich in der Praxis inzwischen sehr gut bewährt, auch auf dem 68000, für den nachstehend ein Programm abgedruckt ist. Hier findet man Instrings bei RAM-residenten Texten mit frappierender Geschwindigkeit. Dafür einige Anhaltspunkte: Wenn der gesuchte String am Ende eines Textes steht oder nicht gefunden werden kann (ungünstigster Fall), dann hat man 200 KB Text in etwa 1 Sekunde und 400 KB in etwa 2 Sekunden abgesucht. Dabei ist zu bemerken, daß die Suche umso schneller abläuft, je länger der eingetippte Suchstring ist.

Damit ergeben sich hervorragende Reaktionszeiten, die es nahelegen, dieses Suchverfahren auf Datenbestände (Datenbanken, Texte) anzuwenden, die in das RAM geladen worden sind, um schnelle Auskunftssysteme zu verwirklichen. Bei dieser Überlegung spricht auch mit, daß wir heute ganz selbstverständlich Speicher von 1 MB und mehr haben (Atari) und daß ggfs. ein schnelles Nachladen von der Festplatte möglich ist, so daß große Bestände abgesucht werden können.

Der Algorithmus: Beim Suchen sieht man häufig Algorithmen, die an einer Textstelle beginnen und von da ab zwei Strings Byte um Byte vergleichen. Sobald sich eine Ungleicheit ergibt, wird das Fenster auf den abzusuchenden Text um 1 Byte verschoben, und der Vergleich beginnt an dieser Stelle von Neuem. Das Suchfenster gleitet damit nur sehr langsam über den Text hinweg.

Die Überlegung von Boyer und Moore ist die folgende: Wenn ein im Textspeicher angetroffenes Zeichen gar nicht im Suchstring enthalten ist, dann kann man das Fenster auf den Text sofort um die volle Länge des Suchstrings weiterschieben und die Suche umso mehr beschleunigen, je länger der Suchstring ist. Gedanklich wird es etwas schwieriger, wenn das im Textspeicher angetroffene Zeichen auch im Suchstring einmal oder häufiger vorkommt. Dann muß man das Suchfenster auf den Text um eine Distanz verschieben, die vom ersten Vorkommen des fraglichen Buchstabens im Suchstring abhängt. Sie wird ab Ende des Suchstrings berechnet, wie auch der Vergleich vom Ende des Suchfensters aus rückwärts abläuft. Dazu kurze Beispiele:

```
Im Textspeicher steht                xxxxxESELyyy
Der Suchstring ist                   ESEL
Seine Buchstaben-Distanzen sind      1210

1. Fensterverschiebung um 4:         xxxxxESELyyy     weil x nicht vorkommt
   Suchstring                           ESEL

2. Fensterverschiebung um 1:         xxxxxESELyyy     das ist die Distanz des E
   Suchstring                           ESEL          Übereinstimmung
```

Man kann den Lösungsweg vereinfacht so beschreiben: Wenn ein Zeichen des Textes nicht im Suchstring vorkommt, rückt das Fenster um eine volle Stringlänge nach rechts. Im anderen Fall wird das Fenster so gerückt, daß unter den laufenden Buchstaben des Textes der gleiche Buchstabe des Suchstrings zu liegen kommt, womit ein Vergleich von hinten her wieder einige Aussicht auf Übereinstimmung hat.

Zum Programm: Bei BEFEHLEO zeigt es ein Menü und wartet auf eine Befehlstaste. Mit 'l' kann eine seqentielle Datei für Versuche mit Namen von der Floppy geladen werden. In der hiesigen Umgebung wird nur gefordert, daß $OO das Zeichen für ein Zeilenende ist (also ggfs. auf $OD $OA achten). Datensätze dürfen eine unterschiedliche Länge haben. Am Beginn haben sie die Marke '#', die nur für eine geordnete Anzeige verwertet wird. Auf den Suchalgorithmus hat dieses Sonderzeichen keinen Einfluß. - Mit geladenem Text kann mit '+' und '-' um einen Datensatz weiter bzw. zurück geblättert werden. Die Suchfunktion wird mit der Taste 'f' erstmalig ausgelöst. Man wird um einen kleingeschriebenen Suchstring gebeten. Die Suche erfolgt dann groß/klein. Die Tasten CR oder ENTER repetieren den Suchvorgang. Wenn nichts (mehr) gefunden wurde, erfolgt interaktiver Hinweis.

Die Suche wird am Label BOYBEF vorbereitet. Man erinnert, daß für die einzelnen Buchstaben des Suchstrings eine Distanz gegen das Ende des Fensters berechnet und in eine Tabelle eingetragen werden muß. Sie heißt hier DISTAB. Ein Buchstabe soll sich selbst in diese Tabelle mit seinem ASCI-Wert indizieren können. Das geschieht später am Label BOYVGLO. Die Tabelle DISTAB ist für 160 Zeichen vorgesehen, womit auch deutsche Sonderzeichen erfaßbar wären. Bei BOYFILL wird die Tabelle mit Nullen gefüllt. Eine Null soll hier heißen: Nimm die Länge des Suchstrings beim Fensterschieben. Man hätte auch die tatsächliche Länge nehmen können.

Bei BOYDIST wird der Suchstring zeichenweise von vorne her abgekämmt. Die Distanzen der Buchstaben werden ASCII-indiziert in die Tabelle DISTAB

eingeschrieben, wobei Nullen und ggfs. frühere Werte für den Buchstaben überschrieben werden (s.o. z.B. bei ESEL).

Die eigentliche Suchschleife findet sich zwischen den Labeln BOYVERGL und BOYVDEC, wobei auch der Befehl Decrement and Branch für die Länge des Strings zum Einsatz kommt. Wenn man diese Schleife mit der Bedingung EQUAL verlassen haben sollte, dann gibt es Übereinstimmung zwischen dem Suchstring und einer Textstelle, und man kann bei FOUND zur Anzeige des wie immer strukturierten Datensatzes schreiten.

Wenn es keine Übereinstimmung im Stringvergleich gab, dann verläßt man wegen DBNE die o.a. Schleife ebenfalls und muß das Suchfenster um die Distanz weiterschieben, die sich mit dem ASCII-Index in D2 ergibt. Das geschieht in den auf DBNE folgenden Befehlen. Ein Sonderfall ist hier der Zeilentrenner $00, für den um 1 Zeichen weitergeschoben wird.

Bei der Eingabe des Suchstrings wird man um Kleinschreibung gebeten. Daher muß im Textspeicher angetroffene Großschreibung für den Vergleich in Kleinschreibung umgeformt werden. Auch das geschieht in der Schleife. Und noch etwas: Als Stellvertreter für einen Buchstaben ist der Klammeraffe im Suchstring zugelassen. Wenn man dieses Zeichen antrifft, dann ist per se Gleicheit der Zeichen herzustellen. Das geschieht mit direkter Beeinflussung des Statusregisters mit dem Befehl MOVE.W #$04,CCR.

In der Summe ist festzustellen: Das Suchverfahren verlangt als Overhead das einmalige Initialisieren der Distanztabelle, die auch bei einer späteren Weitersuche mit der Taste CR benutzt wird. Von diesem Verwaltungsaufwand abgesehen, umfaßt die eigentliche Suchschleife mit Zeichenumwandlung und zugelassenen Stellvertreterzeichen nur 12 Befehle. Sie ist damit kompakt, führt schnell aus und hat den Vorteil, daß sie das Suchfenster in größeren Schritten weiterschiebt.

Lit: /1/ Albrecht, M.: Schneller Stringsuchen mit dem Algorithmus von Boyer & Moore, MICRO MAG Nr. 42, S. 23 ff.

```
* SUCHPROG
* Laden einer sequentiellen Datei
* und beschleunigte Stringsuche nach Boyer & Moore
* Copyright 1987 by Roland Löhr

* Verkehrszeichen
cr          equ         $0d             Carriage Return
lf          equ         $0a             Linefeed
escape      equ         $1b             Escape-Taste
esc         equ         $1b             Kurzform
space       equ         $20             Zwischenraum
GEMDOS      equ         1               Systemaufruf
y           equ         'Y'             in Escape-Sequenzen

 SECTION    ONE

main        movem.l     a0-a6,-(sp)     Register sichern
new         lea         textbuf,a3      1. Abspeicherungsstelle laden
            move.l      a3,edipntr      merken
            clr.b       (a3)            00 als Delimiter am Textanfang

befehle0    move.l      #mtext,d0       Menü anzeigen
            bsr         printline
            bsr         dircon          Befehlstaste holen
            cmpi.b      #escape,d0      und entschlüsseln
            beq         abbruch1        Abbruch mit ESCAPE

befehlea    move.b      d0,befehl       merke Befehlstaste
```

```
            cmpi.b    #'1',d0          Lesen von Floppy?
            beq       lesen            ja, Datei laden
            lea       textbuf,a3
            tst.b     (a3)             Buffer noch leer?
            beq.s     befehle0         dann Folgebefehle abschnüren !!!!

* Befehle, die erst bei gefülltem Textbuffer ausführbar sind:
befl        cmp.b     #'1',d0          Auf Topzeile setzen?
            beq       topzeile         ja
            cmp.b     #'+',d0          Down, in Folgesatz setzen?
            beq       down             ja
            cmp.b     #'-',d0          Up-Befehl, vorheriger Satz
            beq       up
            cmpi.b    #cr,d0           cr = Weitersuchen
            beq.s     sweiter
befehle1    cmpi.b    #'f',d0          Suche nach Boyer
            beq       boybef
            bne       befehle0         Befehl unbekannt

* Stringsuche fortsetzen
sweiter     move.l    d4,a2            Zeiger auf nächsten Datensatz
            cmpa.l    edipntr,a2       sind wir am Ende
            bcc       notfound         ja
            bra       boy1             Die Suche kann weitergehen

* Routinen der Positionierung auf Datensätze

topzeile    lea       textbuf,a3       Zeiger auf Textbeginn (1. Nutzbyte)
topz1       bra       found10          leerer Bildschirm und Anzeige Satz
down        move.l    d4,a3
            cmpa.l    edipntr,a3       sind wir am Ende
            bcc       notfound         ja
            move.l    nextsatz,a3
            cmpa.l    edipntr,a3
            bcc       notfound
            bra       found10
* up, Zeige den Satz davor an
up          move.l    satzbegin,a3     Beginn laufender Satz (1. Nutzbyte)
up2         cmpa.l    #textbuf,a3      nicht vor den Anfang!
            beq       notfound         hat geborgt
up2a        cmp.b     #'*',-(a3)       Krebsgang
            bne.s     up2
            tst.b     -(a3)            wegen satzparms bei found10
            bra       found10

abbruch1    move.b    #escape,d0       Cursor ausschalten
            bsr       conout
            move.b    #'f',d0
            bsr       conout
            movem.l   (sp)+,a0-a6      Register zurückholen
            clr.w     -(sp)            GEMDOS-Funktion 0
            trap      #GEMDOS          Rückkehr zum Betriebssystem

* Suchstring holen und  seine Parameter anlegen
* Suche nach Boyer & Moore
boybef      move.l    #text2,d0        Suchstring anfordern
            bsr       printline
boybef1     lea.l     inbuff,a5        Zeige auf Textbuffer
            bsr       getline          Suchstring holen
            cmp.b     #escape,d0       Will man das Suchen abbrechen?
            beq       befehle0         ja
            bsr       crlf             deutlich absetzen
            bsr       crlf             Neue Zeile
            lea.l     inbuff,a5        Wieder auf Eingabepuffer zeigen
            tst.b     (a5)             Abbruch bei leerer Eingabe
            beq       befehle0
* Rollenzuweisung der Register beim Suchen:
* A4 enthält Länge des Suchstrings,
```

```
* A2 merkt die Einsatzstelle des Suchens in TEXTBUF
* A3 ist Zeiger auf Textende
* A6 ist Zeiger auf Distanztabelle
* D3 enthält die Einsatzstelle
* D2 nimmt das Zeichen aus dem Textspeicher für den Vergleich auf
* D1 wird Schleifenzähler für den Stringvergleich benutzt
          clr.w     d3
          move.b    1(a5,a4.w),d3  Lade letztes Zeichen Suchstring
          lea       textbuf,a2     Zeige auf Beginn des Textspeichers
boyer     moveq     #39,d1         40 Schleifendurchläufe
          lea.l     distab,a5
boyfill   clr.l     (a5)+          fülle 160 Bytes mit Stringlänge 00
          dbra      d1,boyfill     Schleife

          lea.l     inbuff,a5      Zeige auf Anfang Suchstring
          move.w    a4,d2          Suchlänge-1 nach D2 für Schleife
          subq.w    #1,d2
          clr.w     d0             Reste entfernen
          lea.l     distab,a6      Lade Tabellenbasis fest nach A6
boydist   move.b    (a5)+,d0       Hole Stringzeichen Suchstring
          move.b    d2,(a6,d0.w)   Distanz in Tabelle einfüllen
boydist1  subq.w    #1,d2          Distanz-1
          bpl.s     boydist        Eintrag für alle Zeichen Suchstring
          move.b    #1,(a6)        Der Zeilentrenner $00 erhält
*                                  die Distanz 1

* A6 zeigt permanent auf die Distanztabelle DISTAB für Zeichen
* A5 zeigt bei jedem Durchlauf zunächst auf INBUFF
* A4 enthält Länge des Suchstrings
* A2 zeigt zunächst auf Textanfang, wird hochgezählt

          clr.l     d2
          clr.l     d3
boy1      lea.l     inbuff,a5      Zeiger auf Suchstring
          lea       distab,a6      Zeiger auf Distanztabelle
          adda.l    a4,a5          Setze hinter Ende Suchtext
          move.w    a4,d1          Index bilden
          sub.w     #1,d1          für Schleifenzählung -1
          move.w    #$ffff,d3      Schlüsselstelle initialisieren
boyvergl  addq.w    #1,d3          Bei Beginn: 00
          cmpi.b    #'@',-1(a5)    folgt Stellvertreter?
          bne.s     boyvgl         nein
          suba.l    #1,a5          Zeiger auf das Byte davor
          move.w    #$04,ccr       signalisiere: gleich per CCR
          bra.s     boyvdec        und springe über den Vergleich
* Schleife des zeichenweisen Vergleiches
boyvgl    move.b    (a2,d1.w),d2   Zeichen aus Textpuffer prüfen
boyv0     cmpi.b    #$41,d2        kleiner als Klammeraffe?
          bcs.s     boyv1          ja, nichts ändern
          ori.b     #$20,d2        Buchstaben klein machen!!
boyv1     cmp.b     -(a5),d2       Zeichenvergleich: Suchstring/Text
boyvdec   dbne      d1,boyvergl    ggfs. weiter in der Schleife

          beq       found          ja, Übereinstimmung!
          tst.b     d2             Null, Zeilenende bricht ab?
          bne.s     boyvg10        nein
          adda.l    d1,a2          Adresse der 00
          adda.l    #1,a2          +1=Adresse neues Nutzzeichen
          bra.s     boyvg14
boyvg10   move.b    (a6,d2.w),d2   Hole Offset des Zeichens
          bne.s     boyvg11        keine Null
          move.l    a4,d2          Stringlänge bei 00 hinzuaddieren
boyvg11   sub.l     d3,d2
          beq.s     boyvg12
          bpl.s     boyvg13
boyvg12   moveq.l   #1,d2
boyvg13   adda.l    d2,a2          Offset addieren
```

```
boyvgl4   cmpa.l    edipntr,a2      Sind wir am Ende?
          bcc       notfound
          bra.s     boy1            Suche fortsetzen

notfound  bsr       clearscr        Bildschirm löschen
          move.l    #text4,d0       Anzeige: Nichts gefunden
          bsr       printline
          bra       befehle0        neuen Befehl holen

found     move.l    a2,a3           String gefunden ab (A2)
found1    cmpi.b    #'#',-(a3)      Delimiter des Satzes suchen
          bne.s     found1          rückwärts gehen

found10   bsr       satzparms       Hole Parameter des Datensatzes
          bsr       clearscr        Clear Screen
found2    bsr       showsatz        Datensatz anzeigen
          bra       befehle0

clearscr  move.b    #escape,d0      Funktion Clear Screen
          bsr       conout          als Terminal-Sequenz
          move.b    #'E',d0
          bsr       conout
          move.b    #escape,d0      Funktion Cursor ein
          bsr       conout
          move.b    #'e',d0
          bra       conout          rts dort

* Mehre Zeilen eines Datensatzes anzeigen
showsatz  bsr       oneline         Quelle anzeigen
          bsr       oneline         Zeile1
          bsr       oneline         Zeile2
          bsr       oneline         Klassen
          bsr       oneline         Gruppen
          bsr       crlf
          move.l    a3,d4                Zeiger auf #
          move.l    a3,nextsatz
          addq.l    #1,d4           Weitersuche: 1. Nutzbyte
          rts
* Eine Zeile bis Terminator 00 ausgeben, Textzeiger hochzählen
oneline   move.b    (a3)+,d0        Laufendes Byte einer Zeile
          beq.s     onelcr          Abbruch, wenn 00
          bsr       conout          Ausgabe
          bra.s     oneline         Folgezeichen
onelcr    bra       crlf            rts dort, A3 zeigt schon hinter die 0

satzparms move.l    a3,satzbegin    Merke Satzbeginn bei #
          adda.l    #1,a3           vorwärts auf 1. Textbyte
          move.l    a3,satztext     Satzbeginn merken für Ausdruck
          rts

* Taste l, Lesen einer Datenbank von Floppy in das RAM
lesen     bsr       clearscr
          bsr       getfname        Lese Datei von der Floppy
          beq       befehle0        Abbruch mit Escape

          bsr       crlf
          bsr       getdta          alten Zeiger sichern
          bsr       setdta          neuen Zeiger setzen
          lea       filename,a5     Zeiger auf Dateinamen
          bsr       sfirst          File aufsuchen
          beq.s     lesen1          Datei bekannt
          move.l    #textc,d0       Datei nicht vorhanden
          bsr       printline
          bra       befehle0

lesen1    move.l    #textbuf,bufptri    Eingabepuffer=Textbuffer
lesen1b   move.l    fsize,bufleni       Dateigröße
          lea       filename,a5
          bsr       fopen
```

```
          bpl.s     lesen2
          move.l    #texto,d0        "Kann nicht öffnen"
          bsr       printline
          bra       befehle0

lesen2    bsr       fread            D0.L returns # of bytes read or error
          cmpi.b    #'1',befehl      welche Funktion
          beq.s     lesen2a
          move.l    edipntr,a3       Anhaengen
          bra.s     lesen2b
lesen2a   lea       textbuf,a3       alter Anfang
lesen2b   adda.l    fsize,a3
          suba.l    #1,a3
          move.l    a3,edipntr
          bsr       fclose
          bra       topzeile         Gleich satzparms setzen und anzeigen

* Interaktive Texte
* Menue anzeigen
mtext          dc.b     ' Menue, Auswahl mit Tastendruck',cr,lf,cr,lf
          dc.b      ' ESC - Programm abbrechen',cr,lf,07
          dc.b      ' 1   - 1. Datensatz anzeigen',cr,lf
          dc.b      ' +   - Folgesatz anzeigen',cr,lf
          dc.b      ' -   - vorherigen Satz anzeigen',cr,lf
          dc.b      ' f   - Suchfunktion',cr,lf
          dc.b      ' CR  - Suche fortsetzen',cr,lf,0
text9     dc.b      '            Eingabe Filename/Pathname:    ',07,0
textc     dc.b      '            Dieser Datei-Name ist unbekannt'
          dc.b      ', eine Taste drücken',cr,lf,07,0
texto     dc.b      '            Kann die Datei nicht öffnen,'
          dc.b      ' eine Taste drücken',cr,lf,07,0
text2     dc.b      cr,lf,escape,'J  kleingeschriebenen Suchstring eingeben:
',cr,lf,0
text4     dc.b      '   Nichts (mehr) gefunden',cr,lf,07,0

textf     dc.b      '            Achtung: Lesefehler !!',cr,lf,0

* Arbeitsspeicher
dumm0     ds.w      1               align to even address
regsave   ds.l      7               Ablage von Registern
filtflag  ds.w      1               Flag-Variable

distab    ds.l      40              160 Bytes Distanztabelle
befehl    ds.w      1               Befehlstaste merken
satzbegin ds.l      1               Zeiger Satzbeginn mit Zeichen #
satzlen   ds.w      1               Länge Datensatz in Bytes High/Low
satztext  ds.l      1               Zeigen auf Textbeginn im Satz
nextsatz  ds.l      1               Errechneter Zeiger satzbegin+satzlen
*                                   - Zeiger auf Folgesatz
nsatz     ds.l      1               Zeiger auf nächsten Satz
zzeiger   ds.l      1               Zeiger auf aktuellen Zeilenbeginn
textbegin ds.l      1               Beginn Textspeicher, ungerade Adresse
edipntr   ds.l      1               Zeiger auf Einsatzpunkt Editor

* textbuf ds.b                      Variable hinter allem anderen
```

□=

Atari ST-Serie - Softwareübersicht. Per März 1987 erschien die 4. Auflage dieser Übersicht im Umfang von etwa 540 Seiten. Das Buch enthält nach Themengruppen geordnet die Beschreibung von Programmen, ihre Herkunft, ihren Vertrieb und Ihren Preis. Es werden ferner die Anforderungen an die Hardware und die Datenkompatiblität zu anderen Programmen genannt. Die großen Themengruppen sind Branchenlösungen, CAD, Tabellenkalkulation, Dateiverwaltung, Grafik, KI, Medizin, Musik, Peripherie, Programmiersprachen, Text, Utilities, Wirtschaft u.v.a.m. - Wenn auch fertige Programme nicht immer glücklich machen, so findet man hier jedoch eine umfassende Orientierung, um welche Programme man sich ggfs. kümmern sollte.

Roland Löhr

68000 Sortierung

Das nachfolgende Assemblerprogramm dient der Sortierung von Datensätzen mit dem MC68000. Computer-spezifisch sind in ihm nur die Ein- und Ausgaben: Printline (nullterminierten String ausgeben), und die Routinen und Buffer bei SAFESORT: Anlegen einer Datei mit Namen, Schreiben auf die Datei, Datei schließen. Für diese GEMDOS-Umgebung (Atari) sind in Heft 47 einfache Lösungen abgedruckt. Bei anderen Computern (MSDOS, Amiga) laufen die Dinge analog.

Es wird davon ausgegangen, daß die Datensätze bereits im RAM resident sind, und zwar ab Adresse TEXTBUf, und daß EDIPNTR auf die nächst freie Adresse hinter den Datensätzen zeigt, wo im Verlaufe die Vektorfelder ALFAPTR und VEKTBEGIN aufgebaut werden. Datenfelder dürfen unterschiedlich lang sein, sie sind durch das Zeilenende-Zeichen Null untereinander getrennt. Ein Datensatz beginnt mit einer MARKE, die man sich nach Bedarf definieren kann. Wesentlich ist nur, daß das zu sortierende Feld einen festen ABSTAND zu MARKE hat. Man hat also fast alle Freiheiten. Es ist also nicht nötig, und wie in Datenbanken und anderer gegliederter Information oft üblich, mit festen Feldlängen zu arbeiten, wobei die Felder dann auch noch platzfressend nach hinten mit Spaces aufgefüllt werden.

Es ist die Grundidee des Programmes, daß bei den Datensätzen nichts transportiert werden soll. Es wird vielmehr ab VEKTBEGIN eine verbundene Liste aufgebaut, die in einem Langwort die Speicheradresse eines hinzukommenden Datensatzes enthält, dahinter als Wort die Länge und schließlich in je einem Wort die Ordnungsnummer des Vorgängers und die des sortierungsmäßigen Nachfolgers. Die Variable EINSTIEG enthält die Ordnungsnummer desjenigen Datensatzes, der die niedrigste Sortierung hat. Wenn die Sortierung beendet ist, wird das Programm bei SAFESORT fortgesetzt. Von EINSTIEG aus positioniert man auf jeweils einen Datensatz, schreibt ihn auf Floppy (auch RAM-Floppy) und hangelt sich mit den Verbindungszeigern auf die Nachfolger durch das Vektorfeld, bis schließlich hex FFFF anzeigt, daß nichts mehr folgt.

Die reine Sortierung erfolgt ungeheuer schnell: Bei 2200 Datensätzen mit etwa je 55 Bytes wurden bei der Sortierung (auf einstellbare) 12 Spalten 7,1 Sekunden gestoppt. Dieses Tempo wird mit einer segmentierten Sortierung wie folgt erreicht: Wenn das erste Zeichen des Sortierfeldes früher schon einmal vorkam, dann braucht man nicht noch alle Datensätze abzusuchen, die einen hexadezimal geringeren Anfangsbuchstaben haben. Man muß nur wissen, wo der niedrigste Einstieg für das erste Zeichen (Buchstabe, Zahl, Ziffer, Sonderzeichen ggfs. Kontrollzeichen) liegt. Zu diesem Zweck wird das zunächst bei Label SORTIERO zu $FFFF initialisierte Array ALFAPTR unterhalten. Wenn ein Anfangszeichen erstmals auftaucht, dann wird die Ordnungsnummer seines Datensatzes dort eingeschrieben. Aus dieser Nummer kann man nun im Array VEKTBEGIN nachschlagen, welche Datensatzadresse dazugehört. Mit den Zeigern auf die Nachfolger gleitet man dann an den Datensätzen mit gleichem Beginn-Buchstaben vorbei.

Wenn ein Zeichen am Feldbeginn noch nicht im Array ALFAPTR verankkert war, dann wird seine Ordnungsnummer am Label SORTFO per BSR ALFBEG in das Array gespeichert, die Suche beginnt dann aber beim Datensatz, auf den EINSTIEG (allerniedrigster Datensatz) zeigt.

Während der Sortierung können logisch folgende Fälle eintreten:

a) Ein Sortierfeld ist mit dem eines bereits sortierten Datensatzes gleich. Dann wird der hinzukommende Datensatz direkt hinter den bereits sortierten eingeordnet. Bei SORTEND werden dann die Zeiger entsprechend umgehängt.

b) Ein Datensatz ist größer als ein bereits sortierter. Dann sind die Vergleiche fortzusetzen (bei GROESSER), bis der hinzukommende Datensatz einmal kleiner ist als ein Nachfolger. Ausnahme davon: Der soeben verglichene Datensatz war der bisher größte (er hat $FFFF als Zeiger auf den Nachfolger). Dann ist der hinzukommende bei GROESS1 als bisher größter Datensatz an die Kette anzuhängen.

c) Ein hinzukommender Datensatz ist kleiner als einer in der bisherigen verketteten Liste. Dann ist er in jedem Fall beim Label KLEIN3 abzuspeichern. Es sind aber bei KLEINER noch folgende logischen Fälle abzuklären: ca) Der hinzukommende Satz ist kleiner als derjenige, der im Array ALFAPTR (Buchstabenbeginne) verankert ist. Dann ist dort die Nummer des Neulings zu verankern. cb) Der Neuling ist kleiner als derjenige Satz, auf den EINSTIEG weist. In diesem Fall ist die laufende Ordnungsnummer aus dem Register D0 nach Einstieg zu bringen. Der Neuling ist der sortiermäßig niedrigste Datensatz.

Zu den Vergleichen kleiner/gleich/größer: Deutsche Sonderzeichen sind nicht berücksichtigt. Kleingeschriebene Buchstaben werden als Großbuchstaben angesehen. Das geschieht bei den Labeln SORTV0, SORTV2 und ALFBEG. Zwei Felder sind gleich, wenn sie entweder in allen Sortierspalten (Parameter SPALTEN) gleich sind oder wenn sie eine gleiche geringere Länge haben und soweit gleich sind. Ist das Feld des hinzukommenden Strings länger als das verglichene Feld, war aber bis zu dessen Länge gleich, so wird der hinzukommende Datensatz als größer angesehen (ab Label SORTV3).

Zur Programmierung: Die Initialisierung reicht bis zum Label GEHVOR. Die Sortierung beginnt (mit Wiedereintrittspunkt) bei SORTFORT. Bei den Labeln KLEINER, GROESSER und SORTEND wird das Ergebnis von Vergleichen aufgearbeitet. Nach dem Sortiergang wird bei SAFESORT auf Floppy abgespeichert.

Es sind wohl alle 16 Register der CPU im Einsatz, einige zeigen fest auf die Vektorfelder oder auf den Textspeicher. Beim wichtigen Vergleichsbefehl CMP war zu beachten, daß er nur mit einem Datenregister als Ziel zusammenarbeitet. Der im Prinzip elegantere Befehl CMPM (Ax)+,(Ay)+ (Compare Multiple, Byte-Vergleich automatisch zu höheren Adressen)) war nicht einsetzbar, weil es ja auch auf eine Umsetzung kleiner Buchstaben in große ankam. Ansonsten wurden alle "besseren" Adressierungen benutzt: "Indirekt" (Ax), Distanz(Ax) und (Ax)+ sowie "Indirekt mit Index" z.B. (A0,D1). Insgesamt erhält man eine sehr kompakte Formulierung, die schnell ausführt.

```
* SORTIERE
* Programm zum Sortieren von Datensätzen,
* die im Speicher resident sind.
* Es wird physisch nichts umgeordnet, es wird vielmehr
* ein Vektorfeld gebildet, das Zeiger auf die Sortierfolge enthält.
* Der Programmteil SAFESORT speichert dann die Datensätze
* in der sortierten Reihenfolge auf eine Datei ab.

* Ein Zeiger hat in der verketteten Liste folgende Struktur:
* 1 Langwort - Adresse des laufenden Satzes
* 1 Wort der Laenge des Datensatzes
* 1 Wort mit der Zeiger-Nummer des Vorgaengers
* 1 Wort mit der Zeiger-nummer des Nachfolgers, zus. 10 Bytes

* Die Sortierung ist nach Anfangsbuchstaben segmentiert:
* im Array ALFAPTR wird festgehalten, in welcher Datensatz-Nummer
* auf den Datensatz mit dem niedrigsten Vorkommen eines Beginn-
```

```
* Buchstabens oder -Zeichens hingewiesen wird.
* Der Vektor EINSTIEG weist auf den allerersten Datensatz
* Es wird nur nach ASCII-Zeichen bis $7F sortiert
* Für deutsche Sonderzeichen müssen das Array ALFPTR erweitert
* und in der Routine ALFBEG eine Umsetzung vorgenommen werden.

* Folgende Voraussetzungen müssen erfüllt sein:
* Datensätze müssen mit einem einmaligen Startzeichen beginnen
* (hier ist es das Zeichen #). Datenfelder im Satz müssen mit hex 00
* enden. Am Satzende steht zweimal hex 00.
* Das zu sortierende Feld muß einen definierten Abstand
* zum Satzbeginn (bei #) haben, hier ist es ABSTAND.

* Erklärung von Konstanten:
marke     equ       '#'             Zeichen für Beginn Datensatz
abstand   equ       18              Distanz Beginn des Sortierfeldes
*                                   zum MARKER
spalten   equ       12              Zahl der Sortier-Spalten, einstellbar

sortiere  bsr       clearscr        Bildschirm löschen
          move.l    #textsort,d0    Sortiertext anzeigen, Adresse
          bsr       printline       Ausgabe Bildschirm
          move.l    edipntr,d0      Gehe hinter das Textende in EDIPNTR,
*                                   wo das freie RAM beginnt
          addq.l    #2,d0           addiere 2 um ggfs. 1 abzuziehen
          andi.l    #$fffffffe,d0   jetzt haben wir eine gerade Adresse!

sortiera  move.l    d0,alfaptr      Zeigerbereich für Alfa-Beginne anlegen
          add.l     #256,d0         fuer 128 Worte weiterrechnen
          move.l    #$ffffffff,d1   Muster fuer High Byte laden
          move.w    #63,d2          64 Langworte=128 Worte
          lea       alfaptr,a5      Zeiger auf Tabelle

sortier0  move.l    d1,(a5)+        Tabelle mit $FFFF fuellen
          dbra      d2,sortier0

* Die verbundene Liste mit den Vektoren für das Suchen

* und spätere Abspeichern beginnt hinter den Alfa-Zeigern

          move.l    d0,vektbegin    Beginn Vektorfeld/Datensatzzeiger
          move.l    d0,a6           A6=Laufzeiger auf Vektortabelle
*                                   im Aufbau, A6 nicht anders benutzt
          clr.w     d0              D0=Zaehler eingericheter Vektoren=0
          move.w    d0,einstieg     Einstieg bisher bei Vektor 00 merken
*                                   =Einstiegsloch des Maulwurfes


          lea       textbuf,a2      Beginn der Daten, Laufzeiger Textbuffer
          clr.w     d1
          lea       alfaptr,a0      zeige dauerhaft auf Alfa-Zeiger
          bsr       alfbeg          1. Alfa-Zeiger anlegen
          bsr       vektstor        Satzbeginn und Länge im Vektor merken
*                                   auf 2. Satz vorrücken

* Vorgänger und Nachfolger im 1. Vektorsatz zu $ffff markieren

          move.w    #$ffff,(a6)+    -1=kein Vorgaenger
          move.w    #$ffff,(a6)+    -1=kein Nachfolger in Vektor
gehvor    cmpi.b    #marke,(a2)+    setze auf 2. Datensatz
          bne.s     gehvor
          suba.l    #1,a2

*****************************************************************************

sortfort  cmpa.l    edipntr,a2      Ende der Saetze? Wiedereintrittspunkt
          bcc       safesort        ja, nun auf Floppy speichern
* nun koennen die Textketten verglichen werden
          move.l    a2,a5           merke Beginn neuer Datensatz/Vergleich
```

```
          addq.w    #1,d0           laufende Vektornummer erhöhen
          tst.b     d0
          bne.s     sortf0          Meldung alle 256 Saetze?

          move.l    d0,-(sp)        rette und gib den Text aus:
          move.l    a0,-(sp)        'Ich sortiere'
          move.l    #textsort,d0
          bsr       printline
          move.l    (sp)+,a0
          move.l    (sp)+,d0        und Register zurueck

sortf0    clr.l     d1              freies Rechenregister
          bsr       alfbeg          ALFPTR+Index anlegen, wenn unbekannt
*****************************************************************************
sortf1    move.w    d6,current      aktuell geladene Vektor-Nr. merken
          mulu      #10,d6          Jedes Vektorfeld hat 10 Bytes
          add.l     vektbegin,d6    + Anfangsadresse=
          move.l    d6,a3           aktuelles Vektorfeld eingestellt
          move.l    (a3),a4         Zeiger auf Datensatz eingestellt

* A2 ist Selektor-Zeiger für den nächsten einzusortierenden Datensatz
* A5 ist aufsteigender Zeiger auf die Bytes seines Sortierfeldes
* und wird auf A2 zurückgesetzt, wenn weiterer Satz zu vergleichen ist

* A4 ist Selektor-Zeiger auf bereits einsortierte Felder
* ab jeweiligem Einstiegspunkt

          move.w    6(a3),d6        Vektor-Nummer Vorgaenger laden
          move.w    8(a3),d7        Vektor-Nummer Nachfolger laden
          move.w    #spalten,d2     Spaltenzähler beim Sort
          adda.l    #abstand,a4     Setze auf Zeile1 Satz current
          adda.l    #abstand,a5     dito neu einzuordnender Satz

sortv0    tst.w     d2              Spaltenzähler
          beq       sortend         alle Sortierspalten fertig, Gleichheit
          subq.w    #1,d2           Spalte-1, Schleifenzähler
          move.b    (a4)+,d4        Vergleich vorbereiten
          andi.b    #$7f,d4         mask off
          cmpi.b    #'a',d4         nur Kleinbuchstaben wandeln
          bcs.s     sortv2          trifft nicht zu
          cmpi.b    #$7b,d4         Sonderzeichen?
          bcc.s     sortv2          ja
          andi.b    #$df,d4         Kleinbuchstaben groß machen
sortv2    move.b    (a5)+,d5        Abbruch bei 0!
          andi.b    #$7f,d5         mask off
          cmpi.b    #'a',d5         nur Kleinbuchstaben wandeln
          bcs.s     sortv3
          cmpi.b    #$7b,d5         Sonderzeichen?
          bcc.s     sortv3          ja
          andi.b    #$df,d5         Kleinbuchstaben groß machen
sortv3    move.b    d4,d1           Pruefung auf Zeilenenden
          or.b      d5,d1
          beq       sortend         beide Zeilen am Ende, weil 00, =gleich
          tst.b     d4              Einstiegssatz
          beq       groesser        der erreichte Satz ist laenger
          tst.b     d5
          beq.s     kleiner
sortv4    cmp.b     d4,d5           Vergleich Einstiegs/laufender Satz

          beq.s     sortv0          Weitere Spalten (-10) pruefen
          bcc       groesser

* Bei kleiner, groesser oder gleich enthalten jetzt:
* A2 Anfangsadresse laufender Satz
* D6 die Vektor-Nummer des Vorgaengers, D7 die des Nachfolgers

* Der neue Satz ist kleiner als ein sortierter
* Wenn der hinzukommende Satz den gleichen Anfangsbuchstaben,
```

```
* wie der bereits sortierte (größere) hat,
* und wenn dieser der Einstieg für den Buchstaben ist,
* dann den Einstiegszeiger umhaengen auf Wert von D0

kleiner   move.w    alfword,d1      Vektor des Beginn-Buchstabens zurück
          move.w    (a0,d1),d3      Vergleich im ALFAPTR vorbereiten
          cmp.w     current,d3      War der eben verglichene Datensatz
          bne.s     klein3          der kleinste in Beginn-Buchstaben?
          move.w    d0,(a0,d1)      ja, dann den laufenden dort verankern

klein3    bsr       vektstor        Satz-Anfangsadresse, Laenge merken
          cmpi.w    #$ffff,d6       Kommen wir vor den bisherigen Anfang?
          bne.s     kleiner0        nein
          move.w    d0,einstieg     Einstiegsvektor updated
kleiner0  move.w    d6,(a6)+        uebernimm den Vorgaenger, ggfs. $ffff
          move.w    current,(a6)+   Zeiger auf Nachfolger abspeichern
          move.w    d0,6(a3)        Hänge im Vergleichssatz um
          cmpi.w    #$ffff,d6       gab es einen Vorgaenger?
          beq.s     kleiner2        nein
          clr.l     d7
          move.w    d6,d7
          mulu      #10,d7          errechne sein Vektorfeld
          add.l     vektbegin,d7
          move.l    d7,a3           Feld erreicht
          move.w    d0,8(a3)        man ist dort jetzt Nachfolger
kleiner2  bra       sortfort

* Der hinzukommende Satz ist groesser als ein bisher überstrichener
groesser  cmpi.w    #$ffff,d7       Gibt es überhaupt einen Nachfolger?
          beq.s     groess1         nein, dann Satz dahinterhaengen
          clr.l     d6
          move.w    d7,d6           Vergleiche mit dem Nachfolger fort-
*                                   setzen, adressiere sein Vektorfeld
          cmpa.l    edipntr,a2      Ende aller Sätze?
          bcc       safesort        ja, nun abspeichern
          move.l    a2,a5           Beginn neuer Datensatz erinnern
          bra       sortf1          Nachfolger in D6 bereits adressiert,
*                                   gleite weiter über schon sortierte
*                                   Sätze bis Ende oder bis mal kleiner

* den hinzukommenden Satz als bisher groessten hintenanhaengen
groess1   bsr       vektstor        Adresse, Laenge im Vektorenfeld merken
          move.w    current,(a6)+   Zeiger auf Vorgaenger einsetzen
          move.w    d7,(a6)+        alten Nachfolger uebernehmen= $ffff
          move.w    d0,8(a3)        im Vorgaenger auf den laufenden weisen
          bra       sortfort        angehängt, weiter sortieren

* Bei Gleichheit aller Sortierstellen den laufenden Satz dahinterhaengen

sortend   bsr       vektstor        Satzbeginn und Länge ins Vektorfeld
          move.w    d0,8(a3)        Zeige im Vorgaenger auf den laufenden
*                                   Satz als Nachfolger
          move.w    current,(a6)+   er ist der Vorgaenger
          move.w    d7,(a6)+        alten Nachfolger uebernehmen
          cmpi.w    #$ffff,d7       sind wir jetzt letzter Satz?
          beq.s     sortend1        ja
          clr.l     d6
          move.w    d7,d6
          mulu      #10,d6
          add.l     vektbegin,d6
          move.l    d6,a3           alter Nachfolger erreicht
          move.w    d0,6(a3)        laufender dort als Vorgaenger
sortend1  bra       sortfort

****************** Unterprogramme  ****************************

* A6 zeigt auf nächst freie Stelle im Vektorenfeld
* Nun dort Satzbeginn und Länge f. späteres Abspeichern merken
* Per Hauptprogramm dann auch die verbundene Liste weiterführen
```

```
vektstor  move.l    a2,(a6)+        Merke Satzbeginn im Vektor
          move.l    a2,a5
          adda.l    #abstand,a2     Uebergehe Quelle in Satz
sortlen1  cmpi.b    #marke,(a2)+    Suche Beginn Folgesatz bei der Marke
          bne.s     sortlen1
          suba.l    #1,a2           Zeige auf das #
          move.l    a2,d1           Stand benutzen
          sub.l     a5,d1           Laenge
          move.w    d1,(a6)+        im Vektor merken
          rts

* Merke den ersten Buchstaben bei seinem ersten Auftreten in ALFPTR+Index
* A0 zeigt auf die Basis des ALFPTR

alfbeg    move.b    abstand(a2),d1  Buchstabe schon verankert?
          andi.b    #$7f,d1         mask off
          cmpi.b    #'a',d1         Sonderzeichen, Ziffern,Großbuchstaben
          bcs.s     alfbeg1         nicht wandeln
          cmpi.b    #$7b,d1         Sonderzeichen?
          bcc.s     alfbeg1         ja, nicht wandeln
          andi.b    #$df,d1         Kleinbuchstaben groß machen

alfbeg1   lsl.w     #1,d1           *2-Wortadresse alfa-Vektor
          move.w    d1,alfword      Vektoradresse merken
          cmpi.w    #$ffff,(a0,d1)  gibt es schon einen Vektor im alfaptr?
          bne.s     alfbeg2         ja
          move.w    d0,(a0,d1)      Vektor merken in alfaptr+Index D1
          move.w    einstieg,d6     Fange bei Suchen ganz am Beginn an
          rts

alfbeg2   move.w    (a0,d1),d6      Nimm bekannten Einstieg f. Buchstaben
          rts                       für Beginn der Absuche

**********  Abspeichern auf Floppy  ******************************

safesort  bsr       clearscr        Bildschirm löschen
          bsr       gcreatrw        Dateinamen holen und anlegen
          move.l    #textalf,d0     'ich speichere auf Floppy'
          bsr       printline       ausgeben
          clr.l     d0
          move.w    einstieg,d0     Nummer niedrigster Satz holen
safppp    mulu      #10,d0          Offset 1. Vektor berechnen
          add.l     vektbegin,d0    +Vektorbasis
          move.l    d0,a5
          move.l    (a5),bufptro    Ausgabepuffer bestimmt
          clr.l     d0
          move.w    4(a5),d0        Laenge eines Satzes
          move.l    d0,bufleno      Menge Bytes fuer Floppy

          bsr       fwrite          absetzen
          cmpi.w    #$ffff,8(a5)    Begrenzer letzter Satz?
          beq.s     safzzz          ja, fertig
          clr.l d0
          move.w    8(a5),d0
          bra.s     safppp          Speicherung fortsetzen

safzzz    bsr       fclose          Datei schliessen
          bra       help            Gehe in die Kommando-Ebene
*                                   + zeige Bedienerführung an

* Arbeitsspeicher
textbegin ds.l      1
edipntr   ds.l      1
alfaptr   ds.l      1
vektbegin ds.l      1
einstieg  ds.w      1
current   ds.w      1
alfword   ds.w      1
bufptro   ds.l      1
bufleno   ds.l      1
handle    ds.w      1
inbuff    ds.b      80
filename  ds.b      80
textbuf   ds.b      1
```

Martin Albrecht, 4350 Recklinghausen

Ein Texteditor Speichermodell

Die Datenstruktur eines Texteditors ist bestimmend für die Algorithmik und die daraus resultierende Arbeitsgeschwindigkeit. Im folgenden Aufsatz wird eine Variante eines alten Konzepts vorgestellt, die die Verwendung bewährter Algorithmen gestattet und hohe Verarbeitungsgeschwindigkeit mit Einfachheit verbindet.

Die natürlichste Datenstruktur für einen Text ist eine lineare Anordnung in Form einer Zeichenkette. Der Text wird zeichenweise hintereinander im Speicher abgelegt. Die Zeilenenden sind durch CR-Zeichen (Carriage Return) markiert. Bild 1a zeigt das Layout. Zur Handhabung genügen vier Variablen: Ein Zeiger (sot) auf den Beginn eines zusammenhängenden Speicherbereichs, der gleichzeitig den Textbeginn markiert, ein Zeiger (eob) auf das Speicherende (eob), ein Zeiger (bot) auf das aktuelle Ende des Textes (<= Speicherende) und ein Zeiger (now) auf die Textstelle, die gerade bearbeitet wird.

Diese Anordnung hat den Vorteil, daß z.B. Techniken wie der Boyer & Moore Suchalgorithmus nach /1/ leicht angewendet werden können. Jede Textstelle ist direkt und ohne Verwaltungsaufwand erreichbar. Der gesamte Text kann schnell auf Massenspeichern abgelegt oder von dort eingelesen werden. Blockoperationen wie Kopieren, Verschieben oder Löschen von Abschnitten lassen sich durch einfache Verschiebungen realisieren. Dieses simple Konzept hat aber einen entscheidenden Nachteil, um dessen Beseitigung es geht.

Alle Editierfunktionen lassen sich durch die Grundfunktionen Überschreiben, Einfügen und Löschen eines Zeichens aufbauen. Das Überschreiben ist dabei völlig problemlos. Es muß nur ein Zeichen ausgewechselt und der Schreibzeiger weitergerückt werden. Zum Einfügen und Löschen eines Zeichens muß jedoch der gesamte Text von der Schreibposition (now) bis zum Textende (bot) ein Zeichen nach oben oder unten verschoben werden. Diese Verschiebung benötigt einen Zeitaufwand, der proportional zur Größe des Blocks ist. Bei einem langen Text kann das Einfügen am Textende sehr schnell erfolgen, während dieselbe Operation am Textanfang unerträglich langsam ist.

Um den Textrest nicht für jedes Zeichen hin und her zu schieben, bearbeitet man die aktuelle Zeile separat in einem Puffer. Beim Übergang auf die nächste Zeile wird der Pufferinhalt gegen den ursprünglichen Zeileninhalt ausgetauscht.

AIM-65 Besitzer werden das Problem kennen. Der eingebaute Editor arbeitet nach diesem Verfahren. Als Abhilfe wurde die Verschieberoutine verbessert /2/. Eine beschleunigter Transport reicht aber nicht aus, wenn wie beim ATARI ST Speichergrößen von bis zu vier Megabyte bearbeitet werden müssen.

Es kommt also darauf an, den Textrest hinter der Schreibposition möglichst klein zu halten, damit schnell

eingefügt oder gelöscht werden kann. Ein Lösungsweg wäre den Text in viele kurze Abschnitte einzuteilen, die durch Zeiger untereinander verkettet sind. Als Folge handelt man sich eine vergleichsweise komplizierte Zeigerverwaltung ein, und die Vorteile des linearen Textblocks gehen verloren.

Um diesen Nachteil zu vermeiden wurde eine Hybridlösung gefunden, die die Forderung nach einem kurzen Textstück genauso gut erfüllt und mit zwei zusätzlichen Zeigervariablen auskommt.

Bild 1b zeigt das neue Layout. Der Text wird in zwei getrennte Blöcke unterteilt. Er bekommt in kurzem Abstand hinter der Schreibposition ein "hängendes" Textende mit dem neuen Zeiger tbot. Der Rest des Textes wird an das Ende des Speichers geschoben. Zu seiner Handhabung genügt der neue Zeiger rbeg und der alte Zeiger eob auf das Speicherende. Zwischen tbot und rbeg befindet sich der freie Bereich.

Fügt man bei now Text ein, dann muß nur der Bereich von now bis tbot nach unten verschoben werden. Überschreitet der Abstand von now nach tbot eine bestimmte Größe, so wird der untere Teil des Textblocks abgetrennt und nach rbeg verschoben. Beim Löschen geht es genau umgekehrt. Bei Unterschreitung eines minimalen Abstands zwischen now und tbot wird Text vom Rest ab rbeg entnommen und bei tbot angehängt.

Dieses Verfahren mutet kompliziert an. Es funktioniert aber fast ohne Änderungen mit den alten Routinen. Der Zeiger bot auf das Textende wird unverändert geführt. Der Transport der Textblöcke nach oben und unten erfolgt völlig transparent in der Routine, die beim Übergang auf eine neue Zeile den alten Inhalt gegen den Pufferinhalt austauscht. Ein Sonderfall entsteht, wenn der Schreibzeiger now auf eine neue Position gesetzt wird. Dann muß jedesmal die Transportroutine aufgerufen werden, um an dieser Stelle Text bereitzustellen.

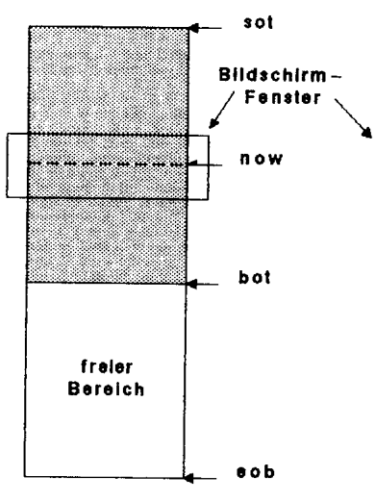


Bild 1a: altes Modell

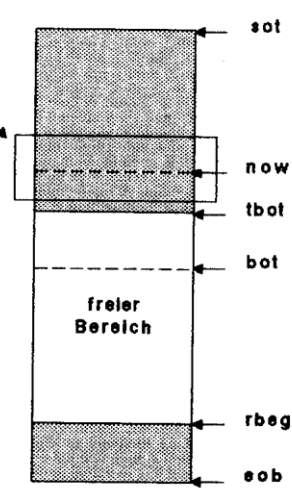


Bild 1b: neues Modell

Der mittlere Abstand zwischen now und tbot sollte so groß sein, daß alle auf dem Bildschirm sichtbaren Zeilen ohne zusätzliche Verschiebung erreichbar sind. So kann z.B. der Cursor schnell umgesetzt werden. Für bestimmte Routinen entstehen leichte Änderungen. Die Suchroutine muß z.B. in beiden Textblöcken suchen. Bei Blockoperationen ist es sinnvoll, den Text vor der Ausführung zu einem Block zusammenzufügen.

Das vorgestellte Verfahren wurde mit guten Resultaten in einem ATARI ST Texteditor verwendet. Die nachfolgenden Programm-Fragmente in M68000 Assemblersprache entstammen dieser Arbeit.

Der Editor hat ein Bildformat von 24 Zeilen und 160 Zeichen pro Zeile. Im Bild befinden sich also maximal 3840 Zeichen. Als minimaler Abstand zwischen now und tbot reichen 4000 Zeichen (ulim) aus, wenn sich now am Bildanfang befindet. Als Maximum wurden 6000 Zeichen (olim) gewählt. Bei Über- bzw. Unterschreitung dieser Grenzen wird der Abstand auf rund 5000 Zeichen eingestellt.

Die Routine **replac** tauscht eine Zeile im Textblock gegen die überarbeitete Zeile im Puffer aus. Die Längen beider Zeilen werden in den Variablen nowlen (Textblock) und buflen (Puffer) übergeben. Der Schreibzeiger now zeigt dabei immer auf den Beginn der aktuellen Zeile. Ein eventueller Speicherüberlauf wird an der Kollision von bot mit dem Speicherende eob erkannt und abgefangen. Nach dem Verschieben des Textes wird die Zeile im Textblock durch den Pufferinhalt ersetzt (jsr copyb). Danach sorgt die Transportroutine mud dafür, daß das hängende Textende tbot im erlaubten Abstand zu now bleibt.

In **mud** wird zunächst geprüft, ob eine Überschreitung der oberen oder unteren Grenze eingetreten ist. Ist das der Fall, dann wird ein Block festgelegt und transportiert. Die Aufteilung geschieht hier immer an einer Zeilengrenze (CR); dafür sorgt die Routine **zende**.

Das Unterprogramm **mblock** fügt den Text zu einem Block zusammen. Das könnte auch leicht mit mud gemacht werden, indem der Schreibzeiger now auf das Textende bot gesetzt wird. mblock arbeitet jedoch unabhängig von der Position von now.

replac und mud benutzen die Verschieberoutine **movmem**. movmem verschiebt einen Block von Adresse a0 bis a1 zur Position a2. Die Besonderheiten der M68000 CPU, wo Worte und Langworte immer an geraden Adressen beginnen müssen, werden dabei berücksichtigt. Die möglichen Adreßkombinationen, gerade/gerade, gerade/ungerade, ungerade/gerade und ungerade/ungerade werden jeweils durch eigene Programmteile unterstützt. Um zusätzlich Rechenzeit einzusparen, wird in den Schleifen immer ein Block von 64 Bytes übertragen. Die sich wiederholenden Befehlsfolgen sind hier nur angedeutet.

Literatur:

/1/ Martin Albrecht
Schneller Stringsuchen mit dem Algorithmus von Boyer & Moore
Micro Mag Nr. 42, S.23 ff.

/2/ Rüdiger Wollenberg, Artur Redder
Schnelles Replace für AIM-65
65xx Micro Mag Nr. 32, S.44 ff.

```
*-------------------------------------------------------------*
*    Konstanten                                               *
*-------------------------------------------------------------*
olim       equ    6000              * für künstliches Textende:
ulim       equ    4000              * oberes und unteres Limit
mid        equ (olim-ulim)/2        * halbe Differenz

*-------------------------------------------------------------*
*    Text Variablen                                           *
*-------------------------------------------------------------*
sot        ds.l       1             * Text(buffer)beginn
bot        ds.l       1             * Textende
eob        ds.l       1             * Textbufferende
now        ds.l       1             * aktuelle Textzeile
tbot       ds.l       1             * hängendes Textende
rbeg       ds.l       1             * Beginn Textrest

nowlen     ds.w       1             * Zeilenlänge im Speicher
buflen     ds.w       1             * Zeilenlänge im Puffer

movflag    ds.b       1             * Speicherüberlauf Flag

*-------------------------------------------------------------*
*    Replacefunktion: Zeile ersetzen, löschen, anfügen        *
*-------------------------------------------------------------*
replac     move.w     buflen,d0     * Distanz =
           sub.w      nowlen,d0     * neue Länge - alte Länge
           ext.l      d0            * auf 32 Bit erweitern
           beq        replac2       * =0, kein Verschieben
           movea.l    d0,a5
           adda.l     bot,a5        * Distanz+bot = neues bot
           cmpa.l     eob,a5        * Speicherende erreicht ?
           scc        movflag       * Speicherüberlaufflag
           bcs        replac1       * neues bot ist < eob
           rts                      * sonst zurück
replac1    move.l     a5,bot        * bot neu setzen
           add.l      d0,tbot       * tbot neu setzen
           movea.l    now,a0
           movea.l    a0,a2
           move.w     nowlen,d0
           ext.l      d0
           adda.l     d0,a0         * von = now + nowlen
           movea.l    tbot,a1       * bis = hängendes Textende
           move.w     buflen,d0
           ext.l      d0
           adda.l     d0,a2         * nach = now + buflen
           jsr        movmem        * Verschiebung ausführen
replac2    jsr        copyb         * Pufferinhalt kopieren
           jmp        mud           * Textende einpendeln

*-------------------------------------------------------------*
*    mud schafft ein hängendes Textende tbot hinter now       *
*    in einer Entfernung von mindestens ulim und höchstens    *
*    olim.                                                    *
*    mud wird beim Textersatz in replac aufgerufen und        *
*    arbeitet dort völlig transparent.                        *
*    mud muß beim Umsetzen von now unbedingt aufgerufen       *
*    werden!                                                  *
*-------------------------------------------------------------*
```

```
mud         movem.l   d0-d1/a0-a2,-(sp)    * d0,d1,a0 bis a2 retten
            movea.l   tbot,a0             * Textmenge hinter now =
            suba.l    now,a0              * tbot-now
            cmpa.l    #olim,a0
            bgt       mud2                * > olim, Block ablegen
            cmpa.l    #ulim,a0
            bge       mud4                * >= ulim, tue nichts

*--------- Block aufrücken ----------------------------------*
            movea.l   eob,a0              * < ulim, Block aufrücken
            cmpa.l    rbeg,a0
            beq       mud4                * kein Rest da, geht nicht
            suba.l    rbeg,a0             * eob-rbeg = Restlänge
            subq.l    #1,a0               * -1
            move.l    now,d0
            addi.l    #ulim+mid,d0        * now+ulim+mid-tbot ergibt
            sub.l     tbot,d0             * Blocklänge f. Aufrücken
            cmpa.l    d0,a0               * noch so viel da ?
            bhi       mud1                * ja
            move.l    a0,d0               * sonst Rest nehmen
mud1        movea.l   rbeg,a1             * Beginn des Rests
            move.l    a1,a0               * rbeg = von (a0)
            adda.l    d0,a1               * + Größe
            jsr       zende               * + Zeilenrest
            move.l    a1,rbeg             * = bis (a1)
            addq.l    #1,rbeg             * +1 = neues rbeg
            movea.l   tbot,a2             * tbot = nach (a2)
            move.l    a2,d0               * tbot = tbot+Blockgröße+1
            add.l     a1,d0
            sub.l     a0,d0
            addq.l    #1,d0
            move.l    d0,tbot             * tbot neu setzen
            bra       mud3                * Transport ausführen
*--------- Block ablegen ------------------------------------*
mud2        movea.l   tbot,a0
            subq.l    #1,a0               * tbot-1 = bis
            movea.l   now,a1
            adda.l    #ulim+mid,a1        * now+ulim+mid
            jsr       zende               * + Zeilenrest
            addq.l    #1,a1               * +1 = von
            move.l    a1,tbot             * und neues tbot
            exg       a0,a1               * tausche von und bis
            movea.l   rbeg,a2             * Beginn des Rests
            subq.l    #1,a2               * -1
            adda.l    a0,a2
            suba.l    a1,a2               * - Blockgröße
            move.l    a2,rbeg             * wird neues rbeg

mud3        jsr       movmem              * Transport ausführen
            movea.l   tbot,a0
            clr.b     (a0)                * 0 bei tbot einsetzen
mud4        movem.l   (sp)+,d0-d1/a0-a2   * d0,d1 a0 bis a2 holen
            rts

*-------------------------------------------------------------*
*    Zeilenende = CR finden                                   *
*-------------------------------------------------------------*
zende       cmpi.b    #cr,(a1)+
            bne       zende
```

```
          subq.l    #1,a1
          rts

*-------------------------------------------------------------*
*    Text zu einem Block zusammenfügen                        *
*-------------------------------------------------------------*
mblock    move.l    eob,d0          * ist überhaupt ein
          cmp.l     rbeg,d0         * Textrest da ?
          beq       mblock1         * nein
          movea.l   rbeg,a0         * rbeg = von (a0)
          sub.l     a0,d0           * Restgröße
          movea.l   eob,a1
          move.l    a1,rbeg         * eob wird neues rbeg
          subq.l    #1,a1           * eob-1 = bis (a1)
          movea.l   tbot,a2         * tbot = nach (a2)
          add.l     a2,d0           * tbot = tbot + Restgröße
          move.l    d0,tbot         * neues tbot
          jsr       movmem          * Block aufrücken
          movea.l   tbot,a0
          clr.b     (a0)            * bei tbot 0 einsetzen
mblock1   rts

*-------------------------------------------------------------*
*    verschiebe Speicherinhalt, maximal 4 Megabyte !          *
*    a0=von, a1=bis (einschließlich), a2=nach                 *
*-------------------------------------------------------------*
movmem    move.l    a1,d0
          sub.l     a0,d0
          addq.l    #1,d0           * Blockgröße + 1
          cmpa.l    a0,a2
          beq       movmem23        * nichts zu verschieben
          bhi       movmem12        * verschiebe nach unten

*-------- verschiebe nach oben -------------------------------*
movmem1   move.l    a0,d1           * von-Adresse
          btst      #0,d1
          bne       movmem8         * ungerade
          move.l    a2,d1           * nach-Adresse
          btst      #0,d1
          bne       movmem9         * gerade/ungerade
movmem2   divu      #64,d0          * gerade/gerade
          bra       movmem4
movmem3   move.l    (a0)+,(a2)+     * 64 Bytes = 16 Langworte
          move.l    (a0)+,(a2)+     * übertragen
          ...
          ...
          move.l    (a0)+,(a2)+
movmem4   dbra      d0,movmem3      * Schleife
movmem5   swap      d0              * Rest übertragen
          bra       movmem7
movmem6   move.b    (a0)+,(a2)+     * byteweise
movmem7   dbra      d0,movmem6
          rts
movmem8   move.l    a2,d1           * ungerade
          btst      #0,d1
          beq       movmem9         * ungerade/gerade
          move.b    (a0)+,(a2)+     * gerade machen
          subq.l    #1,d0           * Länge-1
          bra       movmem2         * dann wie gerade/gerade
```

```
movmem9    divu       #64,d0
           bra        movmem11
movmem10   move.b     (a0)+,(a2)+       * 64 Bytes übertragen
           ...
           ...
           move.b     (a0)+,(a2)+
movmem11   dbra       d0,movmem10       * Schleife
           bra        movmem5           * Rest übertragen

*--------  verschiebe nach unten -----------------------------*
movmem12   adda.l     d0,a0             * übertrage von hinten
           adda.l     d0,a2
           move.l     a0,d1             * von-Adresse
           btst       #0,d1
           bne        movmem19          * ungerade
           move.l     a2,d1             * nach-Adresse
           btst       #0,d1
           bne        movmem20          * gerade/ungerade
movmem13   divu       #64,d0            * gerade/gerade
           bra        movmem15
movmem14   move.l     -(a0),-(a2)       * 64 Bytes = 16 Langworte
           move.l     -(a0),-(a2)       * übertragen
           ...
           ...
           move.l     -(a0),-(a2)
movmem15   dbra       d0,movmem14       * Schleife
movmem16   swap       d0                * Rest übertragen
           bra        movmem18
movmem17   move.b     -(a0),-(a2)       * byteweise
movmem18   dbra       d0,movmem17
           rts
movmem19   move.l     a2,d1             * ungerade
           btst       #0,d1
           beq        movmem20          * ungerade/gerade
           move.b     -(a0),-(a2)       * gerade machen
           subq.l     #1,d0             * Länge-1
           bra        movmem13          * dann wie gerade/gerade
movmem20   divu       #64,d0
           bra        movmem22
movmem21   move.b     -(a0),-(a2)       * 64 Bytes übertragen
           ...
           ...
           move.b     -(a0),-(a2)
movmem22   dbra       d0,movmem21       * Schleife
           bra        movmem16          * Rest übertragen
movmem23   rts
```

Das große Amiga-Jahrbuch 1987. Dieses Jahrbuch im Umfang von etwa 312 Seiten (DM 10,-) wird als offizieller Katalog bezeichnet. Er ist durchwürzt mit kürzeren Darstellungen zu einzelnen den Amiga betreffenden Themen und vor allem mit viel Produkt- und auch Eigenwerbung. Meistenteils dürften die auf den Seiten 25-150 enthaltenen Hinweise auf lieferbare Software am interessantesten sein.

Dipl.-Ing. Uwe Kornnagel, 6100 Darmstadt 14

MAINTOS

Das hier vorgestellte Programm MAINTOS soll den Assembler-Programmierer eines Atari vor unliebsamen Überraschungen schützen.

Beim Atari 260, 520 und 1040 bietet das Betriebssystem TOS die Möglichkeit, sich eine Anzahl von Speicherbereichen zuweisen zu lassen. Das ist eine sehr schöne Sache, da man temporäre Speicherplätze nicht mehr fest im Quellprogramm definieren oder am Ende eines Programmes als Blindlabel kennzeichnen muß. Das ist unter TOS auch sehr gefährlich, da man ggfs. unterschiedliche Programme gleichzeitig im Speicher halten möchte. Belegt man nun den temporären Speicher ohne Wissen des Betriebssystems, dann kann man oft durch Überschreiben eine schlimme Überraschung erleben.

Aus diesem Grunde belegt das Betriebssystem nach dem Laden eines Programmes immer den gesamten noch freien Speicher, damit sich der Anwender austoben kann. Will man nun im Programm vom Betiriebssystem einen Speicher mit dem Aufruf MALLOC anfordern, dann ist man in der Regel enttäuscht, da nur noch wenige Bytes frei gemeldet werden. - Der Programmvorspann MAINTOS schafft da Abhilfe. Hier wird die tatsächliche Länge des Programmes berechnet und der nicht benötigte Speicher mit dem Aufruf MSHRINK an das Betriebssystem zurückgegeben. Nun kann man alle seine Annehmlichkeiten nutzen, wie Overlays, dynamische Speicherzuweisung usw.

Der wichtigste Teil von MAINTOS ist die Berechnung des benötigten Speicherplatzes. Da jedes Programm eine Basepage hat (256 Bytes lang), in der alle für das Programm wichtigen Daten eingetragen sind, kann man hieraus alle Informationen lesen. Die Basepage wird nach dem Laden eines Programmes angelegt und ihre Adresse auf dem Stack abgelegt. Man erhält ihre physische Adresse durch den Aufruf MOVE.L 4(SP),An. Jetzt sind alle Informationen zugänglich. - Ein Programm besteht aus 3 Segmenten:

1. *Text Segment*
   Es enthält die ausführbaren Anweisungen und ggfs. Daten.
2. *Data Segment*
   Es enthält die initialisierten Daten.
3. *Bss Segment*
   Es enthält die nicht initialisierten Daten.

Die Größe des Text-Segmentes ist auf der Adresse Basepage+12 abgelegt. Die Länge des Data-Segmentes finden wir immer auf der Adresse Basepage+20 und die Adresse des Bss-Segmentes liegt auf Basepage+28. Ab der Adresse Basepage+129 findet man die beim Programmaufruf übergebenen Parameter (s. xxxx.TTP oder TOS übernimmt Parameter).

Die Basepage hat selbst eine Länge von 256 Bytes. Der wirklich benötigte Speicherplatz ergibt sich damit wie folgt:

129 + (Basepage+12)
+ (Basepage+20)
+ (Basepage+28)

Die Anfangsadresse eines Programme ist aus der Sicht des Betriebssystemes immer die Basepage. - Am Anfang des Programmes MAINTOS wird eine Datenstruktur –INFO angelegt. Sie soll dem Anwender die wichtigsten Informationen über sein Programm bereitstellen, nachdem es geladen wurde. Diese Struktur –INFO wird jedoch nicht vom Betriebssystem verwendet.

Am Beginn der beifolgenden Liste findet man Parameter für Makros, den denen die Systemfunktionen bequem aufgerufen werden können.

```
 15 *              <<<<<<<<<<   M A I N T O S   >>>>>>>>>>
 16 *
 17 *  METACOMCO MACROASSEMBLER
 18 *
 19 *  Programmsequenz MAINTOS mu& als Vorspann vor Ihr Assemblerprogramm
 20 *  gestellt werden. Da beim ATARI nach dem laden eines Programmes der
 21 *  gesamte Speicher belegt wird, mu& der nicht benötigte Platz wieder
 22 *  freigegeben werden.

STMT            SOURCE STATEMENT

 25 *  >>> Konstanten <<<
 26
 27 *      Allgemeine Konstanten
 28
 29 len.base     equ   $100               Länge der Basepage
 30 len.word     equ   2                  Länge eines Wortes
 31 len.long     equ   4                  Länge eines Langwortes
 32
 33 off.base     equ   4                  Offset für Basepage
 34 off.text     equ   $C                 Offset für Text-Segment
 35 off.data     equ   $14                Offset für Data-Segment
 36 off.bss      equ   $1c                Offset für Bss-Segment
 37
 38 .gemdos      equ   1                  Trap für Gemdos
 39 .bios        equ   14                 Trap für Bios
 40 .xbios       equ   13                 Trap für Xbios
 41

 43 *  GEMDOS CALLS
 44
 45 *       .name = Funktionscode
 46 *      _name = Anzahl der Bytes auf Stack
 47
 48 .pterm0      equ   0                  beende Programm mit Returncode 0
 49 _pterm0      equ   2
 50 .cconin      equ   1                  Zeichen von Tastatur einlesen
 51 _cconin      equ   2
 52 .cconout     equ   2                  Zeichen auf Console ausgeben
 53 _cconout     equ   4
 54 .cauxin      equ   3                  Zeichen von Schnittstelle einlesen
 55 _cauxin      equ   2
 56 .cauxout     equ   4                  Zeichen auf Schnittstelle ausgeben
 57 _cauxout     equ   4
 58 .cprnout     equ   5                  Zeichen auf Drucker ausgeben
 59 _cprnout     equ   4
 60 .crawio      equ   6                  Zeichen einlesen oder ausgeben
 61 _crawio      equ   4
 62 .crwain      equ   7                  Zeichen von Tastatur einlesen ohne Echo
 63 _crawin      equ   2
 64 .cnecin      equ   8                  wie crawin jedoch mit ^s,^q,^c Erkennung
 65 _cnecin      equ   2
 66 .cconws      equ   9                  String auf Console ausgeben
 67 _cconws      equ   6
 68 .cconrs      equ   $a                 String von Console einlesen
 69 _cconrs      equ   6
 70 .cconis      equ   $b                 liegt Zeichen an ?
 71 _cconis      equ   2
 72 .dsetdrv     equ   $e                 Schaltet auf Laufwerk
 73 _dsetdrv     equ   4
 74 .cconos      equ   $10                Console bereit für Ausgabe
 75 _cconos      equ   2
 76 .cprnos      equ   $11                Drucker bereit ?
 77 _cprnos      equ   2
 78 .cauxis      equ   $12                liegt Zeichen an Schnittstelle an ?
 79 _cauxis      equ   2
 80 .cauxos      equ   $13                Schnittstelle bereit zur Ausgabe ?
 81 _causos      equ   2
 82 .dgetdrv     equ   $19                Abfrage des aktuellen Laufwerks
 83 _dgetdrv     equ   2
 84 .fsetdta     equ   $1a                setze Disk Transferadresse
 85 _fsetdta     equ   6
 86 .super       equ   $20                schalte auf Supervisormode
 87 _super       equ   6
 88 .tgetdate    equ   $2a                Datum abfragen
 89 _tgetdate    equ   2
 90 .tsetdate    equ   $2b                Datum einstellen
 91 .tgettime    equ   $2c                Zeit abfragen
 92 _tgettime    equ   2
 93 .tsettime    equ   $2d                Zeit stellen
 94 _tsettime    equ   4
 95 .fgetdta     equ   $3f                Disk Transferadresse abfragen
 96 _fgetdta     equ   2
```

```
 97 .sversion    equ   $30                Versionsnummer abfragen
 98 _sversion    equ   2
 99 .dfree       equ   $36                Diskstatus abfragen
100 _dfree       equ   8
101 .dcreate     equ   $39                Ordner anlegen
102 _dcreate     equ   6

103 .ddelete     equ   $3a                Ordner löschen
104 _ddelete     equ   6
105 .dsetpath    equ   $3b                Ordner setzen
106 _dsetpath    equ   6
107 .fcreate     equ   $3b                Datei einrichten
108 _fcreate     equ   8
109 .fopen       equ   $3d                Datei eröffnen
110 _fopen       equ   8
111 .fclose      equ   $3e                Datei schließen
112 _fclose      equ   4
113 .fread       equ   $3f                Bytes von Datei lesen
114 _fread       equ   12
115 .fwrite      equ   $40                Bytes auf Datei schreiben
116 _fwrite      equ   12
117 .fdelete     equ   $41                Datei löschen
118 _fdelete     equ   6
119 .fseek       equ   $42                Zeiger in Datei Positionieren
120 _fseek       equ   10
121 .fattrib     equ   $43                Dateiatribute abfragen oder setzen
122 _fattrib     equ   10
123 .fdup        equ   $45                Abfrage des duplizierten Kanals
124 _fdup        equ   4
125 .fforce      equ   $46                dupilziere Kanal
126 _fforce      equ   6
127 .dgetpath    equ   $47                Path abfragen
128 _dgetpath    equ   8
129 .mshrink     equ   $4a                Gemdos belegten Speicher verringern
130 _mshrink     equ   12
131 .malloc      equ   $48                Gemdos belege Speicher
132 _malloc      equ   6
133 .mfree       equ   $49                belegten Speicher freigeben
134 _mfree       equ   6
135 .pexec       equ   $4b                Programm ausführen
136 _pexec       equ   16
137 .pterm       equ   $4c                Programm beenden mit Returncode
138 _pterm       equ   4
139 .fsfirst     equ   $4e                Finde ersten Eintrag
140 _fsfirst     equ   8
141 .fsnext      equ   $4f                Finde nächsten Eintrag
142 _fsnext      equ   2
143 .frename     equ   $56                Datei umbenennen
144 _frename     equ   12
145 .fdatime     equ   $57                Datum und Uhrzeit der Datei
146 _fdatime     equ   10
147
148 *  BIOS-CALLS
149 *           Bitte hier im Bedarfsfall nach gleichem Muster BIOS-Vektoren eintragen
150
151 *  XBIOS-CALLS
152 *           Bitte hier im Bedarfsfall nach gleichem Muster XBIOS-Vektoren eintragen

158 *  >>> Macros <<<
159
160 *  Macro Syscall tätigt einen GEMDOS, XBIOS oder BIOS Aufruf bei der Stack-
161 *  bereinigung wird im Macro selbsttätig ein ADDQ.L oder ADDA verwendet.
162 *  Die Parameter müssen wie bei GEMDOS CALLS wie folgt aufgebaut werden:
163 *              .Name          = Funktionscode
164 *              _Name          = Länge der Parameterliste auf dem Stack
165 *  Beim Aufruf des Macros wird nur der Name ohne . oder _ am Anfang benötigt.
166
167 SYSCALL     MACRO                       Systemaufruf allgemein
168             SPC   2
169                   MOVE.W   #.\1,-(SP)
170                   TRAP     #\2
171
172 .STACK   SET      _\1-8                  Stackbereinigungart
173         IFLT      .STACK                 weniger 8 Bytes ?
174                   ADDQ.l   #_\1,SP
175         SPC       2
176                   MEXIT
177         ENDC
178                   ADDA     #_\1,SP
179         SPC       2
180            ENDM
181
```

```
182 *  Gemdosaufruf       z.B.  gemdos cconout
183
184 gemdos    MACRO                          Gemdosaufruf
185             SYSCALL \1,.gemdos
186           ENDM
187
188 *  Xbiosaufruf        z.B.  xbios setscreen
189
190 xbios     MACRO                          Xbiosaufruf
191             SYSCALL  \1,.xbios
192           ENDM
193
194 *  Biosauruf          z.B.  bios  rwabs
195
196 bios      MACRO                          Biosaufruf
197             SYSCALL  \1,.bios
198           ENDM

201 *  Programmeinsprung und Initialisierung Coldstart
202
203 COLDSTART   BRA.S     START.IT
204
205 _INFO       EQU       *              Programminformations Structure
206 P.START     dc.l      0              physikalische Programm_Start_Adresse
207 P.LENGTH    dc.l      0              Programmlaenge in Bytes
208 P.ENDE      dc.l      0              physikalische Programm_End_Adresse
209 BASEPAGE    dc.l      0              physikalische Basepage_Adresse
210 MEM.FREE    dc.l      0              restlicher Speicherplatz für MALLOC
211
212 START.IT    lea       _INFO(pc),a5
213
214             move.l    off.base(sp),a0      Basepageadresse in a0
215             move.l    a0,3*len.long(a5)    und merken
216             move.l    #len.base,d0         Länge der Basepage in d0
217             add.l     off.text(a0),d0                + Länge Text
218             add.l     off.data(a0),d0                + Länge Data
219             add.l     off.bss(a0),d0                 + Länge Bss
220             move.l    d0,len.long(a5)      merke die Größe
221
222             move.l    d0,-(sp)             Aufruf MSHRINK vorbereiten
223             move.l    a0,-(sp)             Basepageadresse
224             clr.w     -(sp)                Dummywert
225
226             gemdos    mshrink              Systemaufruf
226+            SYSCALL mshrink,.gemdos


226+                  MOVE.W    #.mshrink,-(SP)
226+                  TRAP      #.gemdos
226+
226+.STACK    SET       _mshrink-8           Stackbereinigungart
226+         IFLT      .STACK               weniger 8 Bytes ?
226+                   ADDQ.l    #_mshrink,SP
226+         SPC       2
226+                   MEXIT
226+         ENDC
226+                   ADDA      #_mshrink,SP


227
228             lea       COLDSTART(pc),a0
229             move.l    a0,d0
230             move.l    a0,(a5)
231             add.l     len.long(a5),d0
232             move.l    d0,2*len.long(a5)

234 WARMSTART   lea       _INFO(pc),a5
235             move.l    #-1,-(sp)
236             gemdos    malloc
236+            SYSCALL malloc,.gemdos


236+                  MOVE.W    #.malloc,-(SP)
236+                  TRAP      #.gemdos
236+
236+.STACK    SET       _malloc-8            Stackbereinigungart
236+         IFLT      .STACK               weniger 8 Bytes ?
236+                   ADDQ.l    #_malloc,SP


237             move.l    d0,4*len.long(a5)
238
```

```
239 *  Das Programm mit dem Einsprung MAIN ist Ihr eigenes Programm es muß
240 *  mit RTS beendet werden. Sie können eine Wort im D0-Register als
241 *  Returncode mitgeben. Beim Aufruf von MAIN wird die Adresse der Daten-
242 *  Strukture _INFO im D0-Register und die Adresse von WARMSTART im A0-
243 *  Register dem Anwenderprogramm übergeben.
244
245             lea       _INFO(pc),a0
246             move.l    a0,d0
247             lea       WARMSTART(PC),a0
248
249             BSR       MAIN
250
251             andi.l    #$0000ffff,d0
252             move.w    d0,-(sp)
253             gemdos    pterm
253+            SYSCALL pterm,,gemdos


253+                  MOVE.W    #.pterm,-(SP)
253+                  TRAP      #.gemdos
253+
253+.STACK    SET       _pterm-8              Stackbereinigungart
253+        IFLT      .STACK                weniger 8 Bytes ?
253+                  ADDQ.l    #_pterm,SP

257 *-------------------------------------------------------------------------
258
259 * Hier beginnt der Bereich für Ihr Programm
260
261 MAIN
262
263 *  Hier endet Ihr Programm
264
265             clr.w     d0
266             rts
267     end
```

$^5\Sigma\phi^5$

Peter Engels, 5308 Rheinbach 14

C-64 Datenlogger

Oftmals besteht der Wunsch nach einem Meßgerät, mit welchem man digitale Signale messen und auswerten kann. Wenn es dann noch möglich ist, mehrere Kanäle parallel aufzunehmen, also einen Transienten-Recorder zu besitzen, dann ist man in den digitalen Schaltungen besser 'zu Hause' als sonst. Der Nachteil eines solchen Meßgerätes ist allerdings sein hoher Preis. Nimmt man einige Abstriche z.B. an der Meßfrequenz in Kauf, dann kann der Commodore C-64 ebenfalls mit geeigneter Software und minimaler Hardware gute Dienste leisten. Mit diesem Gedanken entstand der nachfolgende Daten-Logger nebst Programm, mit welchem es möglich ist, bis zu 16 Kanäle aufzuzeichnen und auf einen Drucker bzw. eine Disk zu geben.

Um dem Rechner 16 Eingabeleitungen zu verschaffen, wird er mit einer VIA 6522 versehen. Ihre doppelseitige Platine wird mit dem Expansionsport des Rechners verbunden. Bild 1 zeigt ihren Schaltplan. Sie kann auch für andere Anwendungen benutzt werden. Die beiden Gatter des 7409 dienen der zeitlichen Aufbereitung des O2-Signales aus dem Rechner. Alle Ports der VIA sind auf einen 24-poligen Sockel gelegt inkl. Masse und Handshake-Leitungen. Bild 2 gibt die Anschlußbelegung des Sockels wieder (Draufsicht). Das Programm setzt die VIA im Adreßbereich $DF00-$DF0F voraus. Dazu sind auf der Platine die Punkte I0 und F zu verbinden. Wird für eine andere Verwendung der Speicherbereich $DE00-$DEFF benötigt, dann sind die Punkte I0 und E zu verbinden.

Leer- sowie Fertigplatinen sind beim Autor zu erhalten (Peter Engels, Kreisstraße 29, 5308 Rheinbach 14).

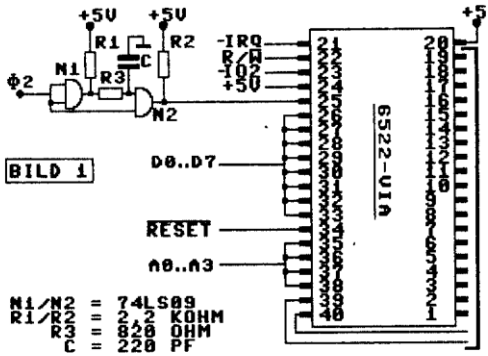
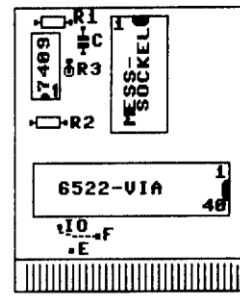
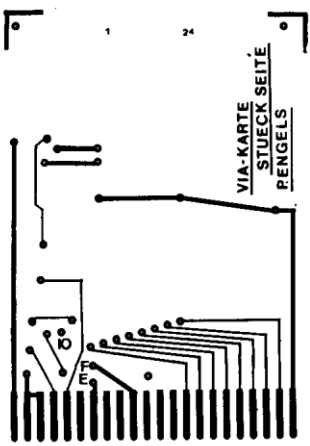
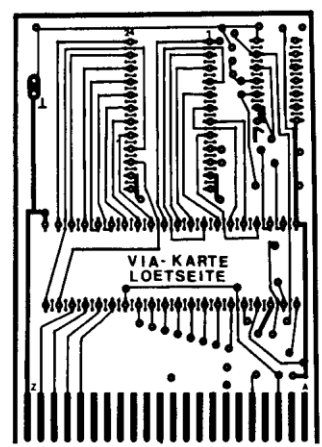


BILD 1

STUECKPLAN VIA-KARTE







ANSCHLUSS BELEGUNG
SOCKEL VIA-KARTE

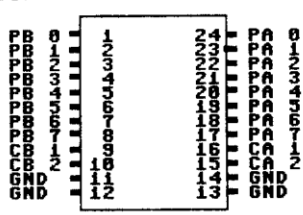


BILD 2

Software

Um das Programm bedienfreundlich zu halten, wurde die Menütechnik angewandt. Es kann als Autostart-Programm in ein EPROM ab $8000 geladen werden. Nach einem Reset meldet es sich im Menü, von dem man mit Tastendruck in ein Untermenü gelangt. Das Hauptmenü erlaubt die Wahl der Zeitbasis (0,5 mS, 1, 5 20 und 50 mS), Setzen des Triggers, Floppy-Funktionen, Schirm- und Druckerausgabe sowie Messen. Die Untermenüs sind:

Zeitbasis ändern. Wird eine der Tasten 1...6 gedrückt, so ändert sich die Statusanzeige am oberen Bildrand, die internen Flags werden gesetzt und die Pointer auf den Speicherstart eingestellt. Das Menü wird mit 'X' verlassen.

Trigger wählen. Dieses Menü setzt die Trigger-Leitungen an CA2 und CB2. Der On-Trigger bestimmt das Starten des Meßvorganges (pos. oder neg. Flanke an CA2), während der OFF-Trigger das Ende der Messung festlegt (Flanke an CB2). Wird im Haupmenü die Taste M=Messen betätigt, dann wartet das Programm, bis die gewählte Flankenart an CA2 eintritt und beginnt dann die Messung. Das Signal an CB2 beendet sie. Im manuellen Trigger-Modus beginnt die Meßwertaufnahme mit Drücken der M-Taste und endet entweder nach Drücken der Stop-Taste oder beim erreichen des Speicherendes. Das manuelle Stoppen ist auch im Modus Trigger-OFF möglich.

Floppy-Funktionen. Aus diesem Menü können Meßwerte gespeichert bzw. zur Weiterbearbeitung geladen werden. Ebenso hat man die Catalog-Funktion und die Abfrage von Floppy-Fehlern. Dieses Menü wird mit 'X' verlassen.

Druckerausgabe. Sie kann auf jedem Drucker erfolgen, der die ESC/P-Sequenzen beherrscht, und erfolgt vom Speicherstart bis -ende, kann aber mit STOP vorzeitig abgebrochen werden. Bild 3 zeigt einen Probeausdruck. In ihm erscheinen die Zeitbasis und die Bezeichnung der Ports. Die seit dem Start verstrichene Zeit wird alle 10 Zeilen skaliert. - Die Geschwindigkeit des Ausdruckes entspricht der einer normalen Textausgabe im unidirektionalen Druck.

Schirmausgabe. Sie dient der schnellen Übersicht der im Speicher vorhandenen Daten. Jede Änderung an den Ports wird als neue Spalte dargestellt. Die Zahlen über dem Diagramm stellen die Zahl der Messungen hexadezimal dar, die die Daten gleichbleibend angestanden haben. Space gibt die nächste Splate aus. Ein Druck auf die Taste 'A' legt den Speicheranfang auf die aktuell eingestellte Salte, ein Druck auf 'E' setzt das Speicherende. Mit 'X' verläßt man das Menü.

Statuszeile. Sie wird in allen Menüs ausgegeben und enthält die momentan bearbeitete Speicherzelle (beim Messen und bei der Ausgabe), die Zeitbasis und den eingestellten Trigger. So ist man immer über die wichtigsten Parameter informiert.

Messung
Die Meßwertaufnahme erfolgt im Interrupt: Der Timer A im Baustein CIA1 wird auf die gewählte Zeitbasis programmiert, und zwar im Modus One Shot. Nach dem Zeitlaulauf löst er einen Interrupt aus. In der IRQ-Routine vergleicht man den momentanen Zustand der Ports mit dem letzten und legt die Werte zusammen mit der Zeit ab, wenn sich etwas änderte. Andernfalls wird nur die Zeit um eine Einheit erhöht. So wird viel Speicherplatz gespart, wenn sich nur wenig ändert.

Die nachstehende Liste mit der IRQ-Routine verdeutlicht den Meßalgorithmus. Sie wurde auf Geschwindigkeit getrimmt, um eine Zeitbasis von 0,5 mS zu erreichen. Das gesamte Programm kann hier aus Platzgründen nicht abgedruckt werden, es ist aber beim Autor erhältlich.

Druckerbetrieb. Die Ausgabe erfolgt über die Routine BSOUT bei $FFD2. Durch Verbiegen der Pointer kann man damit auch eine Parallelschnittstelle am Userport ansprechen. Man kann auch die Device-Nummer ändern. Die Übertragung muß über den Linear-Kanal des Druckers ohne Umrechnung erfolgen. Wenn man die Grafik-Einschaltsequenz des Druckers ändern will (Standard ist ESC 'K' $80 $01), dann kann auch das ab Adresse $833A geschehen, Bei weniger als 4 Zeichen ist die Folge mit 00 abzuschließen. - Das Programm liegt ab $8000 im Speicher und startet sich selbst, wenn es im EPROM ist. Nach dem Laden von Diskette betätigt man die Restore-Taste oder gibt SYS 33692.

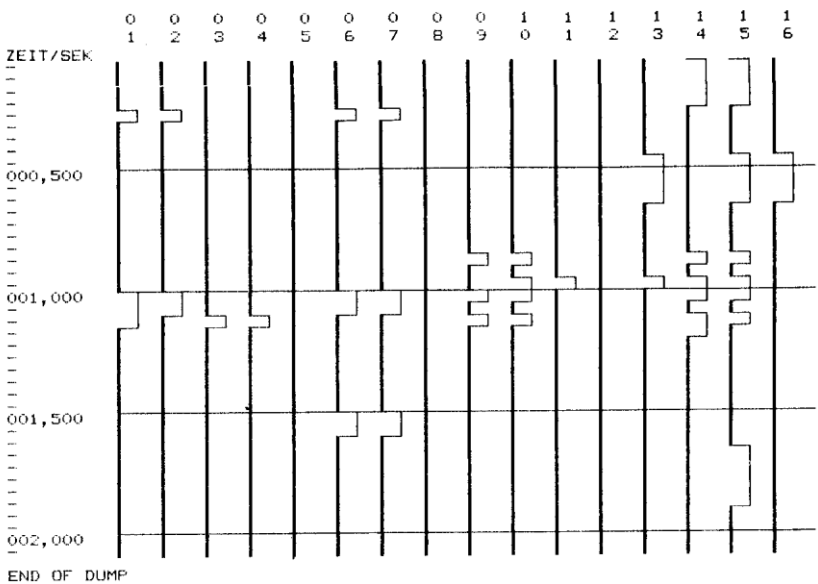


Bild 3: Ausgabe auf dem Drucker

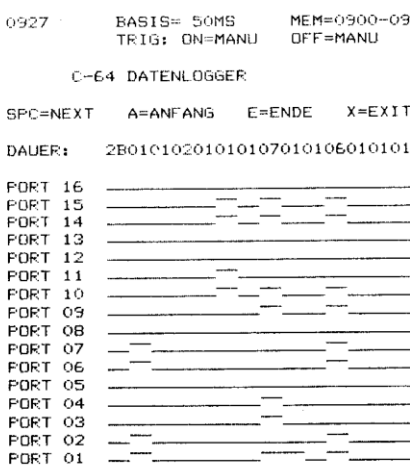


Bild 4: Bildschirmausgabe

```
86F4                ;SCAN-INTERRUPTROUTINE
86F4                ;=====================
86F4                ;
86F4                ;
86F4 AD 0D DC       SCANIRQ     LDA ICR             ;IRQ-FLAG LOESCHEN
86F7                ;
86F7 EE F7 08                   INC TIME            ;ZEIT = ZEIT+1
86FA F0 10                      BEQ READIN          ;>255 -> READIN
86FC                ;
86FC AD 01 DF                   LDA PORT1
86FF CD F8 08                   CMP PORTBUFF+1
8702 D0 08                      BNE READIN
8704 AD 00 DF                   LDA PORT2
8707 CD F9 08                   CMP PORTBUFF+2
870A F0 3B                      BEQ EXIT            ;KEINE AENDERUNG
870C                ;
870C AD 01 DF       READIN      LDA PORT1           ;PORTS LESEN
870F 8D F8 08                   STA PORTBUFF+1      ;UND BUFFERN
8712 AD 00 DF                   LDA PORT2
8715 8D F9 08                   STA PORTBUFF+2
8718                ;
8718 A0 00                      LDY #$00            ;BUFFER IN MEMORY
871A B9 F7 08                   LDA PORTBUFF,Y      ;ABLEGEN
871D 91 FE                      STA (POINTER),Y     ;=ZEIT+2PORTS
871F C8                         INY                 ;Y=1
8720 B9 F7 08                   LDA PORTBUFF,Y
8723 91 FE                      STA (POINTER),Y
8725 C8                         INY                 ;Y=2
8726 B9 F7 08                   LDA PORTBUFF,Y
8729 91 FE                      STA (POINTER),Y
872B                ;
872B D8                         CLD                 ;POINTER FUER NAECHSTES
872C 18                         CLC                 ;EINLESEN UM 3 ERHOEHEN
872D A5 FE                      LDA *POINTER
872F 69 03                      ADC #$03
8731 85 FE                      STA *POINTER
8733 A5 FF                      LDA *POINTER+1
8735 69 00                      ADC #$00
8737 85 FF                      STA *POINTER+1
8739                ;
8739 A9 00                      LDA #$00            ;ZEIT NULLEN
873B 8D F7 08                   STA TIME
873E                ;
873E A5 FF                      LDA *POINTER+1
8740 C9 7F                      CMP #H,MEMEND       ;ENDEFLAG
8742 D0 03                      BNE EXIT            ;SETZEN WENN >7F00
8744 CE FE 08                   DEC OFLAG           ;OFLAG=$FF
8747                ;
8747 68             EXIT        PLA                 ;REGISTER ZURUECK + RTI
8748 A8                         TAY
8749 68                         PLA
874A AA                         TAX
874B 68                         PLA
874C 40                         RTI
874D                            .LC
```

Editorial

Dieses Heft hat den thematischen Schwerpunkt der Assembler-Programmierung auf dem 68000, speziell auf dem Atari ST. Wie in früheren Heften zu anderen Computern, so wurde auch hier wieder angestrebt, zu häufig vorkommenden Aufgaben verständliche Lösungen in Software vorzuschlagen. Ich darf diesen Schwerpunkt und aktuelle Beobachtungen zum Anlaß für folgende Überlegungen nehmen:

Die meisten Leser dieser Zeitschrift haben sich mit einem 65xxx befaßt, viele auch mit anderen Prozessoren und mit ihnen aufgebauten Computern. Was man mit

einem 8-Bit 65xx machen kann, wurde besonders für die älteren Systeme von Commodore und für dem AIM 65 in vielen Beiträgen ausführlich und oft auch lehrhaft dargestellt. Die Computer waren seinerzeit nicht leistungsfähig genug ausgerüstet und mußten mit Ergänzungsvorschlägen für die Hardware versehen werden. - Es hat seit einiger Zeit nur noch wenig Anregungen zu 65xx gegeben. Es ist auch feststellen, daß der Computer PC-128 von Commodore, der eine zusätzliche CPU Z80 enthält und damit CP/M-Programme nachfahren kann, im ernsthaften Computer-Publikum wenig angekommen ist. Seine Markteinführung erfolgte zögernd im Herbst 1985. Heute im Mai 1987 weiß man nicht sicher, ob der Computer überhaupt noch in der Serie gebaut wird. Der Preisverfall ist bereits abzulesen. Dieses Modell dürfte der letzte Versuch gewesen sein, einen Computer mit 65xx in das breite Publikum zu tragen.

Es gibt inzwischen das System Apple II GS mit der CPU 65816 (8/16 Bit), die in dieser Zeitschrift bereits ausführlicher beschrieben wurde. Der Apple II GS hat einen recht hohen Preis. Es muß damit bezweifelt werden, ob fertige Computer mit 65xxx eine noch zunehmende Verbreitung finden können.

Anders ist die Lage bei 8/16-Bit CPUs und Platinen-Computern für Steuerungszwecke zu beurteilen, wo die CMOS-Versionen den Forderungen nach preiswertem Aufbau, geringem Stromverbrauch und zunehmend schnellerer Reaktion in der Prozeßkontrolle entgegenkommen. Die 65xxx-Familie bietet hier einige Auswahl mit einem aufwärts-kompatiblen Befehlessatz. Hier sind auch die Prozessoren von Mitsubishi M507xx mit hoher Taktrate und vielen Interface-Ports zu erwähnen. Größer noch ist die Auswahl bei den ähnlichen Einchippern von Motorola und Hitachi, die sich ebenso einfach programmieren lassen, wie ein 65xx. Es sind diese CPUs, mit deren Möglichkeiten man sich für eigene Aufbauten künftig befassen sollte.

Bei Personal Computern und Entwicklungshilfsmitteln geht der Trend inzwischen deutlich zu echten 16-Bit Geräten. Die einfachen IBM-Clones, die ja mit 8088 nur einen 8-Bit Datenbus haben, fallen derzeit preislich in den Bereich der Heimcomputer. IBM hat eine neue Produktlinie angekündigt, die den Markt etwas verunsicherte. Insider sagen, daß man keinen Computer nehmen sollte, der weniger als einen 80386 unter der Haube hat, weil die Architektur der niedrigeren Intel-Prozessoren in der Adressierung zugenagelt sei. Der Mangel an durchgehender Speicheradressierung mit einem Banking stattdessen betraf gleicherweise die Linie CBM 6xx/7xx, die inzwischen aufgegeben wurde und die zu preiswerten Ausverkaufsangeboten geführt hat. In der Sicht des Herausgebers sind die jetzt angekündigten Computer noch keineswegs umwerfend, denn das dem 80386 angepaßte Betriebssystem soll erst 1988 zur Verfügung stehen. Dieses Hinterherhinken ist eigentlich blamabel, denn IBM ist kapitalmäßig an Intel beteiligt und kann sich wegen dieser Verbindung auch softwaremäßig von der ersten Stunde des Chipentwurfes an auf die neuen Möglichkeiten einstellen. Und IBM sollte eigentlich über reichlich brainpower verfügen.

Die Welt der Computer ohne Intel-CPU wird derzeit von drei Herstellern belebt, Apple, Atari und Commodore. Apple baute die CPU 68000 zunächst in die Lisa und dann in den Macintosh ein, kam zu einer erheblichen Verbreitung und zu einem stattlichen Angebot an Software. Atari folgte ab Ende 1985 mit der ST-Linie und kam praktisch vom Nullpunkt auf einen beachtlichen Marktanteil, besonders hier in Deutschland. Der Anteil von Atari in Raunheim am Weltumsatz des Herrn Tramiel liegt bei 30%! Zu den ST-Rechnern gibt es inzwischen ein großes Angebot guter Software. - Wenn man einmal bei Commodore von der IBM-kompatiblen Linie absieht, so hat man im Bereich der Personal Computer nur noch den Amiga 2000 im Rennen (das Modell 500 gehört mehr in die Einsteigerklasse). Auch hier mit der CPU 68000, mit der Möglichkeit, Karten mit einem Intel-Prozessor einzustecken oder ggfs. auch eine mit der CPU 68020.

Wenn man den Amiga 2000 mit seiner Leistungsfähigkeit und auch mit der gegenüber dem 1000 verbesserten Tastatur lobend erwähnen muß, so darf man doch vermuten, daß er wegen seiner Preisklasse und wegen anderer Besonderheiten eher ein beschränktes Publikum finden wird. Für den normalen Bürobetrieb reicht ein Computer mit schwarzer unverschnörkelter Schrift auf weißem Hintergrund in einer flimmerfreien Anzeige, wie man sie z.B. bei Atari findet.

Die Entwicklung geht weiter zu Computern mit CPU 68020. Apple wird demnächst den Macintosh II mit dieser CPU liefern. Den Verlautbarungen von Motorola ist weiterhin zu entnehmen, daß dieser Chip für OEM-Kunden in Stückzahlen im weiteren Verlauf des Jahres 1987 nur noch etwa DM 100 kosten wird, nachdem die ersten Handmuster Ende 1985 noch etwa DM 2000 kosteten. Von der preislichen Seite her wird es damit kaum noch Hindernisse gegen die Verwendung dieses Chips geben. Der Wettbewerb wird damit weitere Hersteller veranlassen, ihn in Computer einzubauen. Der Herausgeber vermutet, daß das schon vor dem Ende 1987 der Fall sein wird und daß möglicherweise Atari einmal wieder den Durchbruch mit einem preiswerten System versucht. - Man darf weiterhin damit rechnen, daß eine andere weltweit renommierte, ältere Firma dann auch schon mit einem Computer mit 68030 auf den Plan tritt. Diese CPU mit ihrer Harvard-Architektur für die unabhängige und überlappende Ansprache von Programm- und Datenspeichern erhöht den möglichen Durchsatz abermals.

Herr Jack Tramiel (Atari) hat anläßlich der Pressekonferenz im März in Hannover angekündigt, es würde in diesem Jahre in den USA noch Krieg geben. Man konnte darin die Absicht ablesen, daß er die Großen der Branche herausfordern will. Man darf also erwarten, daß das Jahr 1987 noch gut für Überraschungen ist.

Ich erwähne diese Beobachtungen und Überlegungen, um dem Leser bewußt zu machen, daß wir hinsichtlich der Personal Computer in einer Zeit des absehbaren schnellen Umbruches sind und daß wir uns gedanklich darauf einzustellen haben. Insofern bin ich froh, mit den Beiträgen in diesem Heft Anregungen für die aktuelle und leistungsfähige Familie 68xxx geben zu können. Und aus den nunmehr längeren Erfahrungen als Programmierer und Buchautor für Assembler auf dem 68000 darf ich hinzufügen: Architektur und Befehlssatz dieser CPU machen die Programmierung zur Freude. Man hat eine Zwei-Adreß-Maschine mit Quell- und Zieloperanden. Damit muß man im ersten Durchgang nur noch etwa 50% der Zeilen schreiben, die man für einen 8-Bit Prozessor tippen muß. Es kommen aber weitere Leistungssteigerungen in der Programmierung hinzu, gar nicht einmal unbedingt durch die in die Hardware eingebaute Multiplikation und Division, sondern vor allem durch die Tatsache, daß man Zeiger auf Operanden nicht mühsam erhöhen muß. Das geschieht viel einfacher mit den Adressierungsarten für die Adreßregister. Man hat weiterhin eine große Anzahl von CPU-Registern und einen beliebig großen Stack. Umladevorgänge für Register sind damit seltener. In der Summe darf man wohl sagen, daß man auf dem 68000 mit etwa 1/3 notwendiger Programmzeilen das Gleiche erreicht wie in einem Programm für 65xx oder 68xx. Unabhängig vom schnelleren Takt der CPU sollte man daher ins Auge fassen, soweit das noch nicht geschehen ist, diese mögliche Leistungssteigerung in der Programmierung künftig zu benutzen.

□=□

Hard Disk SH 204 für Atari ST. Wenige Tage vor Redaktionsschluß konnte beim Autor die genannte Hardddisk in Betrieb genommen werden. Der Händler berichtet, daß sie bei einem Preis von derzeit knapp DM 1.300 beliebt ist und gut verkauft wird. Speichervolumen 20 MB. Es können bis zu vier Teilabschnitte eingerichtet werden (Partitions), wobei der größte zusammenhängende Abschnitt bis zu 16 MB enthalten kann. - Bei der mitgelieferten Systemdiskette zur

Formatierung gibt es dem Vernehmen nach Beschränkungen in der Zahl der Ordner oder Dateien, die in einer Partition verwaltet werden können. Atari-Händler haben inzwischen jedoch Formatierungsprogramme in der Hand, die die Beschränkungen überwinden. - In Kurzform kann man derzeit folgendes Urteil abgeben: Die Harddisk arbeitet außerordentlich schnell und stellt damit einen echten Gewinn dar. Die Boot-Routine kann auf die Harddisk gelegt werden, so daß man gleich ihr Inhaltsverzeichnis sieht und die gewählten Accessories im Desktop hat. Sie dürfte wegen ihrer Geschwindigkeit die Benutzung einer RAM-Disk entbehrlich machen. Beim Assemblieren von und zur Platte wurden keine auffällige Geschwindigkeitsunterschiede festgestellt. Ein Vergleich: Das Programm WORDPLUS wird in 4,8 Sek von der Harddisk geladen. Beim Laden aus der RAM-Disk dauert es 3,8 Sek. - Ein eigenes Programm unter GEMDOS zum Laden von Texten benötigt für 530 KB etwa 3,5 Sek beim Gebrauch der Harddisk. Zum Vergleich sei vermerkt, daß Wordplus selbst 58,4 Sek. für den gleichen Text (von der SH 204) benötigt, womit sich die Vermutung bestätigt, daß Programme durch die Einbindung in GEM langsamer werden. - Die Leistungsdaten sind damit beachtlich und kommen dem Wunsche entgegen, schnell von einem großen Datenbestand auf einen anderen umschalten zu können. Der eingebaute Ventilator ist etwas zu laut. Es sollte Möglichkeiten geben, ihn durch einen leiseren Typ zu ersetzen.

Commodore Amiga. Für die Übersetzung des in Heft 48 S. 38 besprochenen Buches von Jon Thomas Berry 'Inside the Amiga' stand dem Herausgeber während einiger Zeit ein Amiaga 1000 in der Ausführung NTSC zur Verfügung. Das Buch mit seinen zahlreichen in der Sprache C formulierten Systemprogrammen erscheint etwa Ende Juni 1987 beim te-wi Verlag. Es darf als eine klar formulierte lehrhafte Anleitung bezeichnet werden, mit der man die zahlreichen Dienste der drei Teile des Betriebssystemes für die Anwenderprogrammierung einsetzen kann, nämlich ausführender Kernel, AmigaDOS in der höheren Ebene der Ausführung und Intuition als intuitiv erfaßbare Benutzeroberfläche, ähnlich GEM beim Atari.

Zum Amiga: Das genannte NTSC-System entstammt einer Lieferung per Entwicklungssystem Anfang 1986. Dazu gehörten einige Bände Dokumenation, Cross-Compiler Lattice-C für MSDOS-Systeme, ein C-Compiler Lattice-C zur Benutzung auf dem Amiga (also native einzusetzen), BASIC und noch einiges. Die seinerzeit gelieferte Workbench (Benutzeroberfläche) Version 1.1 wurde inzwischen durch die Version 1.2 ersetzt. Der Amiga 1000 wird nicht mehr hergestellt (ein kurzes Künstlerleben) und durch das Modell 2000 als Personal Computer abgelöst. Dieses hat Slots wie ein IBM oder Apple, in die man in einiger Zeit eine CPU 68020 stecken kann oder auch einen Intel 80286 oder 80386 und andere Erweiterungskarten. Das ist im Grunde ein großzügiger Aufbau. Die Frage ist nur, ob man das alles braucht. Auf jeden Fall hat das derzeitige System eine deutlich verbeserte Tastatur. Daneben gibt es gewissermaßen als Heimcomputer das Modell 500.

Man darf berichten, daß die Anschaffung und Benutzung eines Compilers Lattice-C mit Überraschungen aufwartet. Unter der Versionsnummer 3.03 verbergen sich offensichtlich unterschiedliche Versionen, die sich in den mitgelieferten Bibliotheken unterscheiden. Die Hantierungshinweise in der Dokumentation verweisen dabei gelegentlich auf einzubindende Dateien, die gar nicht oder die nicht mehr mitgeliefert werden. Das betrifft insbesondere die Dateien c.o, LStartup.obj und AStartup.obj. Es standen hier der Compiler aus dem Entwicklungspaket zur Verfügung und eine im Mai 1987 zusätzliche gekaufte Version 3.03C. Mit beiden kam man zwar über die beiden Pässe des Compilers und über den Linker fehlerlos hinweg, das erzeugte Programm brachte andere Dinge auf den Bildschirm, als geplant, führte aber immer zum Absturz der Maschine. Gleicher Quelltext lief auf dem einen Compiler in Pass 1 fehlerlos, auf dem anderen wurden Include-Dateien offensichtlich anders angesprochen. Es waren lange frustrierende Versuche, die auch wegen der langsamen Arbeitsweise viel

Zeit benötigten. Die Beispielprogramme konnten erst bei einem anderen Amiga-Betreiber mit dessen Version von Lattice-C 3.03 erst dort und dann auch hier erfolgreich übersetzt und ausgeführt werden. Es verblieben geringe Unterschiede im Verhalten der Maschinen. Die Gründe für den unerwarteten Hergang sind nicht bekannt.

Bücher

Thies, K-D.: Die innovativen 80286/80386 Architekturen, Teil 1: Der 80286. te-wi Verlag, München 1986, 300 S. DM 59,-, ISBN 3-921 803-63-2. Der Autor verschiedener Bücher zur Familie 8xxxx wendet sich an Systemprogrammierer, Entwickler und Betreiber eines Computers mit 80286. Gezeigt werden Aufbau, Philosophie und praktische Programmierung in folgenden Kapiteln: Speicherorganisation, Memory Management und virtuelle Adressierung, Schutz, Tasks, Interrupts, E/A, Benutzung des 80287, Einführung in den "System Builder" BLD286. Der klar formulierte Text ist mit zahlreichen erklärenden Grafiken und mit Codesequenzen in Assemblersprache versehen, so daß er seiner Absicht, gründliche Informationen zu geben, gut gerecht werden dürfte.

Huckert, E.: C Toolbox Textverarbeitung. Verlag Markt & Technhnik, Haar 1986, 273 S., DM 69,-, ISBN 3-89090-364-9. Lieferung mit 2 Quelltext-Disketten (MSDOS) zum Buch. Untertitel: Vollständige Textverarbeitung inklusive Spellchecker und Silbentrennung in C-Quellcode für IBM-PC und Kompatible. Wir finden folgende Kapitel: Abriß der Textverarbeitung, Texterfassung (Screen-Editor), Textformatierung, Silbentrennung, Spellchecker, Textübertragung (Modem), Kombination zu einem Textsystem. In den einzelnen Kapiteln werden wird die Programmierung von zweckbezogenen Modulen dargestellt. Das Buch wendet sich an fortgeschrittene Hobby- und an Systemprogrammierer sowie an Studenten. Stofflich werden damit die Überlegungen und Lösungswege aufgezeigt, die bei der Anlage oder Erweiterung eines Textsystemes auftreten. Ob die Sprache C dabei das richtige Vehikel für eine Verwirklichung ist, wenn es um die Bearbeitung großer Texte geht, wird man wohl erst nach einer probeweisen Compilierung der Quelltexte auf den Disketten beurteilen können. Es hängt ja schließlich auch vom Compiler ab, wieviel Redundanz in den entstehenden Code einfließt und wie geschickt von den Möglichkeiten der CPU Gebrauch gemacht wird.

Schildt, H.: Professionelles C. Bei McGraw-Hill, Hamburg 1986, 340 S., DM 51,-, ISBN 3-89028-075-7. Übersetzung aus dem Amerikanischen. Kapitel: Überblick, Sortieren und Suchen, Warteschlangen, Stacks, verkettete Listen, Dynamische Zuteilung, Schnittstellen zum Assembler, zum Betriebssystem, Statistik, Datenkompression, Zufallszahlen, Zerlegung und Bewertung von Ausdrücken, Konvertierung von Pascal und Basic nach C, Effizienz der Programmierung, Anhänge. Das sorgfältig aufgemachte Buch enthält zahlreiche Grafiken und Beispielprogramme, die die Erfahrungen einer langen Programmierpraxis widerspiegeln. Wer in C programmiert, wird in diesem Buch zahlreiche Anregungen zu den genannten Themen für die eigene Arbeit finden.

Schildt, H.: Professionelles Turbo Pascal. Bei McGraw-Hill, Hamburg 1987, 302 S., DM 49,80, ISBN 3-89028-091-9. Dieses Buch entspricht dem vorerwähnten von Herbert Schildt, diesmal auf Pascal bezogen.

Sender, G.: Atari St - Arbeiten mit GEM. Band 1: Die AES-Bibliothek. Verlag Sybex, Düsseldorf 1986, 320 S., DM 68,-, ISBN 3-88745-626-2. Buch mit Diskette für Atari mit den C-Quelltexten. AES ist der Application Environment Server des GEM mit zahlreichen Bibliotheken und Funktionsaufrufen für Anwenderprogramme. Diese machen ihrerseits Gebrauch von Funktionen des VDI

(Virtual Device Interface) mit seinen grafischen Fähigkeiten und universellen Schnittstellen.

Das sorgfältig aufgemachte Buch hat drei Hauptteile. Im ersten findet man eine verständliche verbale Erklärung der Bibliotheken und ihrer Funktionen. Damit wird ein sehr guter Überblick geschaffen. Der zweite Teil enthält in C formulierte und reich kommentierte Beispielprogramme zum Gebrauch der Funktionen. Der dritte Abschnitt belegt schließlich die einzelnen Funktionen mit Namen, Aufgabe, Parametern und einem beispielhaften Aufruf in C. - Das Buch zeichnet sich durch die verständlichen Zusammenhänge aus, die es wohl besser darstellt, als andere Bücher, die dem Rezensenten zu GEM in die Hand kamen. Sein Vorteil liegt auch in der Programmdiskette, die dazu anregt, mit den Parametern zu experimentieren. Wer mit Fenstern und Menüs programmieren möchte und nicht nur unter TOS, der sollte dieses Buch bei seiner Wahl berücksichtigen. Die Dokumentation der Parameter wird auch für den Assembler-Programmierer wertvoll sein.

Baeseler, F.; Heck, B.: Desktop Publishing. Bei MacGraw-Hill, Hamburg 1987, 290 S., DM 69,50, ISBN 389028-090-0. Laserdrucker, Macintosh mit Pagemaker und DTP (Desktop Publishing, nit mit DDT zu verwechseln!) sind derzeit sehr im Gespräch. Es geht einfach um die Frage, wie man mit dem Personal Computer und weiteren Hilfsmitteln Dokumente oder auch ganze Bücher am Bildschirm so gestaltet, daß der Laserdrucker die Vervielfältigung übernehmen kann, wenn es sich um kleine Auflagen handelt oder daß er reprografische Vorlagen schafft. Das lebensnah geschriebene Buch erscheint im richtigen Augenblick, um die neuen Möglichkeiten zu erklären und auch Erwartungen zu dämpfen, daß nun jeder morgen als Seitengestalter und Drucker weitermachen kann. Immerhin wollen auch die Regeln der schwarzen Kunst beherrscht sein. Dieses Buch ist lesenswert und sollte herangezogen werden, wenn Projekte zum DTP anstehen. - Die Autoren haben ihr Buch selbst reprofähig gestaltet. Es läßt sich sehen. Der Fachmann erkennt natürlich kleine Unterschiede zu den bisherigen Satzverfahren, die konturenschärfer arbeiten.

Roland Löhr

AES-Parameter

Zur Fortsetzung im nächsten Heft beginnen wir hier mit dem Abdruck einer Aufrufroutine nebst Tabelle für den Atari ST. Es geht um die Benutzung der Funktionen des AES, des Application Environment Managers unter GEM. Diese Funktionen werden wie solche des TOS mit einer Nummer aufgerufen, zusammen mit der Übergabe von Standard-Parametern, die denen der Tabelle AESPARMS entsprechen. Die Übergabe erfolgt im Format Wort, und zwar für die Funktionsnummer im Feld CONTRL, für die Parameter bei CONTROL+2, +4, +6 und +8. Normalerweise wird bei CONTROL+8 eine 0 eingetragen. Die Funktion 112 macht hiervon eine Ausnahme, sie verlangt hier eine 1. - Die Felder ab CONTROL können beliebig an gerader Adresse (zusammenhängend) liegen. Beim Aufruf der AES-Funktionen muß man im Register D1.L den Zeiger auf diese Felder übergeben.

Mit dem Programm lt. nachfolgender Liste können alle Funktionen bequem aufgerufen werden, ohne daß man einzelne Unterprogramme schreiben muß. Eine Besonderheit ergab sich dadurch, daß die Funktionsnummern bei 10 beginnen. Das war bei der Berechnung des anzusprechenden Tabellenplatzes zu berücksichtigen. - Auf den Gebrauch des AES gehen wir in einem weiteren Beitrag ein.

```
* AESPARMS
* Umsetzung von AES-Aufrufen. Funktionsnummer in D0.B uebergeben
* Für RSRC_GADDR, Funktion 112 Besonderheiten beachten!
* by Roland Löhr 10.3.1987
```

```
aes       and.l     #$ff,d0          High Bytes immer ausblenden
          move.w    contrl           = Opcode
          lea       aesparms,a0      Setze aud die Tabelle
          moveq.l   #3,d1            Multiplikation vorbereiten
          mulu      d0,d1            =Offset der Funktionsparameter+30
          sub.w     #30,d1           weil die Opcodes mit 10 beginnen
          move.b    (a0,d1.w),d0     1. Parm
          move.w    d0,contrl+2      in das Array
          move.b    1(a0,d1.w),d0    2. Parm
          move.w    d0,contrl+4      in das Array
          move.b    2(a0,d1.w),d0    3. Parm
          move.w    d0,contrl+6
          clr.w     contrl+8         on default
          cmpi.w    #112,contrl      ist es RSRC_GADDR?
          bne.s     aes1             nein
          move.w    #1,contrl+8
aes1      move.l    #aespb,d1        Zeiger auf AES-Parameterblock
          move.w    #aesk,d0         AES-Kennung
          trap      #gem             GEM aufrufen
          rts                        Nun ggfs. Rückgaben uebernehmen

* Parameter der AES-Funktionen
* fuer contrl+2, contrl+4 und control+6
* Die Parameter müssen auf das Format WORD ausgedehnt werden
* Roland Loehr, 8.3.1987

aesparms                             *   Application Manager
        dc.b    0,1,0 Fnkt.  10      *   appl_init
        dc.b    2,1,1 Fnkt.  11          appl_read
        dc.b    2,1,1 Fnkt.  12          appl_write
        dc.b    0,1,1 Fnkt.  13          appl_find
        dc.b    2,1,1 Fnkt.  14          appl_tplay
        dc.b    1,1,1 Fnkt.  15          appl_trecord
        dc.b    0,0,0 Fnkt.  16
        dc.b    0,0,0 Fnkt.  17
        dc.b    0,0,0 Fnkt.  18
        dc.b    0,1,0 Fnkt.  19          appl_exit
* Event Manager
        dc.b    0,1,0 Fnkt.  20          evnt_keybd
        dc.b    3,5,0 Fnkt.  21          evnt_button
        dc.b    5,5,0 Fnkt.  22          evnt_mouse
        dc.b    0,1,1 Fnkt.  23          evnt_mesag
        dc.b    2,1,0 Fnkt.  24          evnt_timer
        dc.b    16,7,1 Fnkt. 25          evnt_multi
        dc.b    2,1,0 Fnkt.  26          evnt_dclick
        dc.b    0,0,0 Fnkt.  27
        dc.b    0,0,0 Fnkt.  28
        dc.b    0,0,0 Fnkt.  29
* Menu Manager
        dc.b    1,1,1 Fnkt.  30          menu_bar
        dc.b    2,1,1 Fnkt.  31          menu_icheck
        dc.b    2,1,1 Fnkt.  32          menu_ienable
        dc.b    2,1,1 Fnkt.  33          menu_tnormal
        dc.b    1,1,2 Fnkt.  34          menu_text
        dc.b    1,1,1 Fnkt.  35          menu_register
        dc.b    0,0,0 Fnkt.  36
        dc.b    0,0,0 Fnkt.  37
        dc.b    0,0,0 Fnkt.  38
        dc.b    0,0,0 Fnkt.  39
```

Fortsetzung folgt

Neue Assembler

Neu konzipierte Assembler für 6502, R65C02, 65802/65816
und 507xx (Mitsubishi) auf CBM 710/720

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 AIM 65 und PC-128 werden Softwareprodukte nicht mehr geliefert.

6800/6802/6803/6303 Cross-Assembler
für CBM-Systeme auf Anforderung

Cross-Assembler auf ATARI ST

Ab Ende Juni 1987 sind Assembler lieferbar für 65xxx und 6805/68705
Weitere Assembler in Vorbereitung

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

MICRO MAG

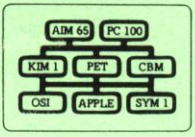
HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☏ (04102) 5 58 16

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.

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,--