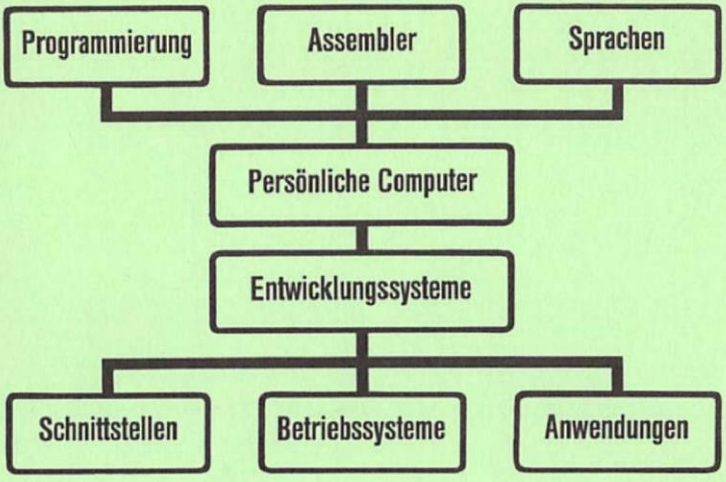
MICRO MAG

DM 9,50 September 1986



68000

Inhaltsverzeichnis

Editor PC-128 3
GEMDOS-Routinen (ATARI ST) 27
68000 Filter 36
CBM-Eprommer 43
Assembler-Praxis auf Atari ST 58
Patches für die C-Bibliothek des Atari ST 59
Editorial 61
Bücher 61

ATARI ST

ATARI ST

ASSEMBLER-PRAXIS AUF ATARI ST

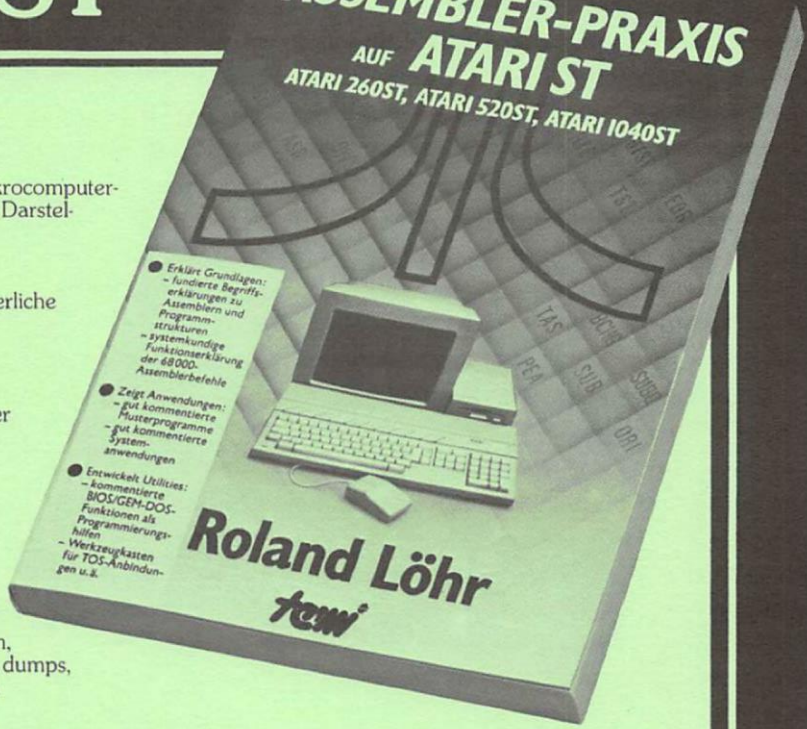
Roland Löhr
...ein Altmeister der Assembleranwendung, Herausgeber des Mikrocomputer-Magazins MICRO MAG, veröffentlicht bei te-wi seine souveräne Darstellung der Assemblerprogrammierung auf ATARI STs.

Erklärt Grundlagen:

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

Zeigt Anwendungen:

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



Entwickelt Hilfsprogramme:

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

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

ca. 300 Seiten, Softcover, DM 59,–

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

Neue Assembler

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

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

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

Ausstattung und Lieferung wie vor. DM 350,--

6800/6802/6803/6303 Cross-Assembler

Für CBM-Systeme und PC-128 in Vorbereitung

6805 und 68705 Cross-Assembler auf ATARI ST

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

Mathe-ROM für 6502, Sockel $D000

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

Den CBM 610/710/720 eindeutschen!

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

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

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

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

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

Roland Löhr

Editor PC-128

Mit dem Artikel "Banking im PC-128" in Heft 46 gingen wir bereits ausführlicher auf den doch sehr leistungsfähigen und preiswerten Computer von Commodore ein. Besonders wurden die Konfigurationsregister der Memory Management Unit abgehandelt und wie man mit Ihnen den Durchgriff durch Speicherbänke bewirkt. Zur Erinnerung kurz: Fünf Register der MMU sind immer und bei allen Speicherkonfigurationen "sichtbar". Unter Adresse $FF00 liegt das eigentliche Konfigurationsregister MMUCR. Durch Befehle wie STA, STX oder STY schaltet es mit sofortiger Wirkung Festwertspeicher, RAM und den I/O-Bereich entprechend den Wünschen des Programmierers ein. In den vier nachfolgenden Adressen findet man Lade-Konfigurationsregister. Wenn man etwas in sie hineinschreibt, dann wird die Konfiguration ebenfalls umgeschaltet, jedoch nicht etwa mit dem aktuellen Inhalt des Akkus bei einem STA-Befehl, sondern mit dem Wert, den man früher einmal in die entsprechenden Prä-Konfigurationsregister ab Adresse $D501 abgelegt hatte. - Dieser Mechanismus spart z.B. das Sichern des Akkus, wenn man ein Byte unter Einschaltung des I/O-Bereiches unter Benutzung der ROM-Routinen ausgeben will. Mit der MMU können ferner ein RAM-Bereich für alle Bänke als gemeinsam (common) erklärt und die Lage der Zero Page sowie der Stack-Page disponiert werden.

Die Konfiguration

Der nachstehende Text-Editor für den PC-128 macht von den vorerwähnten Einrichtungen Gebrauch und ist neben seiner Nützlichkeit ein gutes Programmierbeispiel, wie man mit den neuen Gegebenheiten fertig wird. Die Zero-Page wird zwar nicht verschobén und auch nicht die Stack-Page. Versuche zeigten jedoch, daß man die Zero Page ungestraft verlegen kann, ohne daß der Rechner abstürzt. Es besteht aber für eine solche Verlegung gar keine Notwendigkeit, wenn man auf den Gebrauch des BASIC vorübergehend verzichtet. In diesem Fall kann man fast alle Zellen ab $0A...$8F für eigene Assembler-Programmierung mißbrauchen, ohne die Parameter des Kernels zu stören.

In Heft 46 wurde bereits erwähnt, daß man den Bereich des aktiven Befehles nicht mit einer Speicherkonfiguration per MMU ausblenden darf. Wenn man also an Daten in einer anderen Bank arbeitet, dann sollte die Bearbeitungsroutine im gemeinsamen RAM aller Bänke liegen, das ja immer eingeschaltet ist. Das Betriebssystem des PC-128 macht von dieser Möglichkeit Gebrauch, es verlegt etliche Routinen in den niedrigen RAM-Bereich. Der hier abgedruckte Editor kopiert bei der Initialisierung ebenfalls seine Routinen in den Zeilen 951 bis 1094 in das RAM ab Adresse $800. Diese Routinen dienen vor allem dem Transport von ganzen Zeichenketten von Bank zu Bank, der Anzeige von Zeilen und der Blockverschiebung. Dabei wird die Speicherkonfiguration möglichst nur einmal am Beginn und am Ende einer solchen Funktion umgeschaltet, was gegenüber den Routinen des Betriebssystemes zu einer erheblichen Zeitersparnis führt. Lediglich bei der Such-Routine SUCH4 muß für den Vergleich zweier Strings dauernd zwischen Bank0 und Bank1 hin- und hergeschaltet werden. Dafür ist der Suchalgorithmus, der in ähnlicher Form in Heft 42 abgedruckt wurde, außerordentlich schnell: Man findet einen Instring, der am Ende einer 64KB-Bank steht, schneller als in 1 Sekunde.

Wegen der Benutzung des RAM-Bereiches ab $800, der normalerweise dem BASIC als Runtime-Stack dient, wurde der Common-Bereich in Zeile 204 auf 4KB festgelegt.

Einige Zeilen zuvor wurde die CPU auf 2 MHz geschaltet (FAST-Modus). So haben wir grob folgende Konfiguration des Rechners: Gemeinsames RAM bis Adresse $FFF, Routinen mit Bankdurchgriff ab $800. Wie im Kernel, so wird auch hier der Eingabepuffer, über den fast alles läuft, ab Adresse $200 unterhalten. Das Textprogramm residiert in Bank0. In dieser Bank ist normalerweise RAM bis $BFFF verfügbar. In den darüberliegenden Adressen sind der Kernel und der I/O-Bereich eingeschaltet. An etwa zwei Stellen wird vorübergehend auch einmal der Festwertspeicher $8000...BFFF eingeschaltet. - Der eigentliche Textspeicher befindet sich aber in Bank1 ab $1000. Er darf sich bis $FEFF erstrecken, denn ab $FF00 liegen ja die sichtbaren MMU-Register. So stehen dem Anwender knapp 60KB Textspeicher zur Verfügung, Platz genug für etwa 1000 Zeilen mit 60 Anschlägen. Oder ein anderes Beispiel: Der Quelltext dieses Editor-Programmes (31 KB) würde fast zweimal in den Textspeicher passen.

Zum Programm

Der Editor-Teil des Betriebsprogrammes enthält zahlreiche nützliche Funktionen für die Editierung am Bildschirm. Meistens werden sie gar nicht direkt mit JSR aufgerufen, sondern in einer der folgenden Formen: 'Gib ein Steuerzeichen auf den Bildschirm aus' oder als Escape-Sequenz, z.B. als ESC+d (entferne eine Zeile und ziehe die nachfolgenden heran, benutzt bei KILL) oder als ESC+i (Insert einer Leerzeile). Auf diese Weise erhalten wir einen vorwiegend Screen-orientierten Editor, in dem wir mit den Cursor-Tasten von Zeile zu Zeile setzen können. Wer will, kann jedoch auch mit den Tasten 'u' (up), 'd' (down) oder CR von Zeile zu Zeile rangieren. In eine bereits abgeschlossene Zeile kommt man mit der Taste 'c' (change) hinein. Dann gelten hier die Tasten für die seitlichen Cursorbewegungen, Insert, Delete sowie die Zeichentasten für das Überschreiben von vorhandenem Text. Eine mit 'c' zur Editierung aufgemachte Zeile wird mit CR wieder in geänderter oder unveränderter Form verlassen. Wenn sich die Länge einer Zeile durch das Editieren verändert hat, dann wird im weiteren Verlauf entweder MOVBACK oder KILSUB1 (Zeilen 764, 789) aufgerufen, um den Rest des Textes als Block zu verschieben.

Eine Besonderheit ist beim Editieren zu beachten: Das Ende einer abgeschlossenen Zeile wird durch einen Schrägstrich (Slash) auf hellem Grund (invers) sichtbar gemacht. Dieses Zeichen darf beim Editieren nicht überschrieben werden. Man darf es jedoch durch Insert/Delete links davon seitlich verrücken. Andere Textsysteme fassen eine Zeile des gespeicherten Textes wie eine Bildzeile mit immer 80 Zeichen an. Das vereinfacht zwar viele Überlegungen, führt aber zu einer massiven Vergeudung des Speicherplatzes, weil insbesondere Programmzeilen immer viele Zwischenräume (Spaces) am Zeilenende haben. In diesen Fällen lassen sich erheblich weniger Zeilen in einer RAM-Bank unterbringen. Mit dem sichtbaren Zeilenende haben wir jedoch ein Mittel, kein Byte zu verschwenden und ggfs. dennoch eine gewollte Zahl von Spaces an ein Zeilenende zu schreiben. Der Nachteil ist, daß man so einfach nicht mit dem Cursor über das Geschriebene hinwegfahren kann, es sei denn, man treibt noch mehr Verwaltungsaufwand.

Zu den Rangierbefehlen gehören 't' (Top, oberste Zeile einstellen), 'b' (Bottom, auf letzte Zeile positionieren), '+' und '-', mit denen man eine Bildschirmseite weiter oder zurückgeht (eine Zeile wird dabei immer überlappend in beiden Screens angezeigt) sowie '#' (Zahlenkreuz, positioniere auf die Zeile Nr. ...).

Block-Befehle sind 'C' und 'K', mit denen ein Bereich von mehr als 3 KB invers markiert und in einen Blockpuffer (Scratchpad) gebracht werden kann (Cut, die Funktion wird mit 'C' gestartet und mit 'C' beendet). Mit 'K' (kopiere) wird dieser Extra-Text dann an beliebige Stelle wieder eingezogen. Bei den entsprechenden Routinen CUT und COPY ab Zeile 812 ist auf folgende

Besonderheit hinzuweisen: Der Blockpuffer liegt ab Adresse $F000 in Bank0, also 'hinter' dem normal eingeschalteten Kernel nebst I/O. Für die Transporte muß hier daher vorübergehend RAM per Konfigurationsregister eingeschaltet werden.

Alle hier genannten Befehle führen sehr schnell aus, schneller auch als auf vielen Textsystemen, die der Autor z.B. auf 68000-Rechnern (8 MHz) gesehen hat. Möglich wird das durch weitgehenden Verzicht auf Routinen des Betriebssystems, die Parameter durch die Bänke transportieren sollen. Die in das Common RAM verlegten Routinen (ab Zeile 951) und auch die in Heft 46 vorgeschlagenen dürften dabei auch für andere Aufgaben die meisten Bedarfsfälle abdecken. Sicher läßt sich einiges noch eleganter lösen. Wesentlich ist wohl die Feststellung, daß man den PC-128 recht problemlos programmieren und auch für ernsthafte Zwecke benutzen kann, wenn man auf die jeweils notwendige Konfiguration achtet. Mit etwas Planung ist es dann möglich, fast allen vorhandenen RAM-Speicher für eine Anwendung auszunutzen. Bei den hier zusätzlich zum Editor entwickelten Assemblern (6502/65C02/65816 sowie 6805/68705) wird der Raum bis etwa $5000 in Bank0 für das Assembler-Programm benutzt. Ab $5000 liegt dann die Symboltafel. Erzeugter Code kann an beliebiger Stelle in Bank1 und ggfs. in Bank0 abgelegt werden, so daß sich ein sehr großzügiges Arbeiten ergibt.

Ein- und Ausgabe

Laufende Eingaben von der Tastatur werden mit der Taste 'e' begonnen und mit der Sequenz ESC+§ abgeschlossen. Hier ist vor allem das Unterprogramm GETSTRING im Einsatz (ab Zeile 252), das auch die seitlichen Cursorbewegungen sowie Insert und Delete verfolgt. Der Autor hat es in ähnlicher Form bereits für 6502 allgemein, für den CBM 610...720 und für den Atari (68000) in den Heften 44, 45 und 46 veröffentlicht. Es hat sich eigentlich immer bewährt, zumal auf ihm aufbauend auch ASCII-Zeichen als Zahleneingaben ausgewertet werden können, wie hier in NUMMER ab Zeile 608. - Auch die laufende Eingabe mitten in vorhandenen Text hinein wird mit GETSTRING bewirkt: Routine NSERT ab Zeile 463. Text kann aber auch von der Floppy gelesen werden. An das Ende mit READ (Zeile 555) und mitten hinein mit READNSERT (480). - Die Ausgabe von Text erfolgt mit DISLIN (im Common RAM) auf den Bildschirm, mit WRIT FLOP auf die Floppy und mit PRINT auf den Drucker. Auch die Hilfe-Funktion (Taste 'h', ab Zeile 1408) steht als BS-Ausgabe zur Verfügung. In diesem Programmteil wird das Unterprogramm KPRIMM des Kernels benutzt: Man kann einen mit 00 begrenzten Text direkt hinter den Aufruf JSR KPRIMM schreiben. Der Text wird auf die aktive Einheit ausgegeben. Die Programmfortsetzung erfolgt an der Stelle, die auf die 00 folgt.

```
0003 000000          start      =$2000
0004 000000                     .fil f1:edi-definitions
0005                            ;$ko"edi-definitions"
0006                            ;deklarationen für editor by roland löhr
0008                            ;verkehrszeichen
0009 000000          lf         =$0a              ;linefeed
0010 000000          blank      =$20              ;zwischenraum
0011 000000          space      =blank
0012 000000          komma      =$2c
0013 000000          quote      =$27              ;einfaches hochkomma
0014 000000          rubout     =$7f              ;ruecktaste
0015 000000          null       =0
0016 000000          cr         =$0d              carriage return
0017 000000          escape     =$1b              escape-zeichen
0018 000000          delete     =$14
0019 000000          insert     =$94              tastenzeichen
0020 000000          revers_on  =$12              reverse schrift ein
0021 000000          curs_down  =$11              cursor down character
0022 000000          cursor_up  =$91              cursor up character
0023 000000          revrs_off  =$92              bzw. aus
0024 000000          home       =$13              home cursor
0025 000000          clrscreen  =$93              clear screen und home
```

```
0027                    ;********* hardware-umgebung ***
0028 000000 mmucr       =$ff00          sichtbares konfigurationsregi
                                        ster der mmu
0029 000000 mmucr1      =$d500          mmu konfigurationsregister
0030 000000 bank0       =mmucr+1        mnemon bankbezeichnung
0031 000000 bank1       =mmucr+2
0032 000000 symbank     =$3f            reine ram-bank 0 für symbole
0033 000000 pcra        =$d501          praekonfigurationsregister a
0034 000000 lcra        =$ff01          sichtbares lade-konfiguration
                                        sregister
0035 000000 pcrb        =$d502          preakonfigurationsregister b
0036 000000 lcrb        =$ff02          sichtbares lade-konfiguration
                                        sregister
0037 000000 mmurcr      =$d506          ram-konfigurationsregister
0039                    ;********* systemvariablen cbm pc-128
0040                    ;in der zero page
0041 000000             *=$0e           diverse zeilen- und längenzei
                                        ger
0042 00000e lstchr      *=*+1           letzter buchstabe
0043 00000f zl_len      *=*+1           laenge eingabezeile
0044 000010 such_len    *=*+1           laenge suchstring aktuelle su
                                        che
0045 000011 such_lenw   *=*+1           vorhaltung beim weitersuchen
0046 000012 sav_len     *=*+1           alte laenge gesichert
0047 000013 temp        *=*+1           temporary storage
0048 000014 mode        *=*+1           00=normale texteingabe, $ff=c
                                        hange-mode
0049 000015 t::tbeg     *=*+1           ;zeiger auf den textbeginn im
                                         eingabepuffer
0050 000016 index1      =$22            zeiger auf eingabepuffer
0052 000016             *=$26           diverse pointer
0053 000026 movptr      *=*+2           pointer fuer blockmove
0054 000028 such_pntr   *=*+2           einsetzstelle beim fortgesetz
                                        ten suchen
0055 00002a end         *=*+2           pointer beim stringsuchen
0056 00002c symsav      *=*+2           temp ablage
0057 00002e sympag      *=*+1           zahl zu movender pages
0059                    ;diverse flags
0060 00002f ins_flag    *=*+1           00=append-, $ff=insert-modus
0061 000030 flop_vekt   *=*+2           indirekt vektor, floppy ausga
                                        be
0062 000032 disflag     *=*+1           display flag beim zeilentrans
                                        port
0064                    ;blockmove: to=zielpointer, t2=quellpointer, t1
0064                    =menge-1
0065 000033 to          =$60            adressenpointer/bewertung von
                                         ausdruecken
0066 000033 t1          =$63            dito, abfolge: low/high/bank
0067 000033 t2          =$66            dito
0068 000033 t::tptr     =$7a            pointer in den tastaturbufer
                                        ab $200
0069 000033             *=$7b
0070 00007b sctop       =$e5            oberste zeile bildschirm-fens
                                        ter
0071 00007b pntr        =$ec            cursorspalte
0072 00007b tblx        =$eb            cursorzeile
0073 00007b curspalte   =pntr           alias aktive cursorspalte
0074 00007b curzeile    =tblx           aktive cursorzeile 0...18
0075 00007b qtsw        =$f4            quote-flag, anfuehrungszeiche
                                        n
0076 00007b insflg      =$f6            insert-flag esc/a/c
0077 00007b indx        =$ea            position letztes zeichen in z
                                        eile
0078 00007b rvsflg      =$f3            revers-flag
0079 00007b la          =$b8            aktives log. file
0080 00007b sa          =$b9            aktive sekundaradresse
0081 00007b dflto       =$9a            aktive ausgabe-geraete-nr.
0082 00007b fa          =$ba            aktive geraete-nummer
0083 00007b fnlen       =$b7            laenge file-name
0084 00007b fnadr       =$bb            pointer auf den file-name
0085 00007b fnbank      =$c7            bank des aktiven file-name
0086 00007b status      =$90            i/o-status
```

```
0088                        ;********  variablen des benutzers, zero-page
0089 00007b      inbufp     =$fa              ;zeiger in den eingabepuffer
0090 00007b      nowln      =$fb              pointer auf aktuelle zeile im
                                               editor-buffer
0091 00007b      botln      =$fd              pointer auf ende des editor-b
                                              uffers
0092                        ;****** zeiger-variablen, zero-page
0094                        ;
0095                        ;variablen des anwenders
0096 00007b      inbuff     =$0200            allg. eingabepuffer= kb-buffe
                                              r
0097 00007b      common     =$0800            common ram area im basic run
                                              time stack
0098 00007b      putchar    =common           transport eines zeichens
0099 00007b      putchar1   =common+_putchar1-_putchar
0100 00007b      putline    =common+_putline-_putchartransport einer zeile,
                                              bottom
0101 00007b      tonowl     =common+_tonowl-_putchartransport auf lfd. stell
                                              e
0102 00007b      such4      =common+_such4-_putcharsuchen in bank1
0103 00007b      movblck    =common+_movblck-_putcharblockmove nach links
0104 00007b      getchar    =common+_getchar-_putcharholen eines zeichens
0105 00007b      getchar1   =common+_getchar1-_putchar
0106 00007b      movback3   =common+_movback3-_putcharverschiebung in einer
                                              seite
0107 00007b      movbackr   =common+_movbackr-_putcharverschiebung ganzer se
                                              iten
0108 00007b      common1    =$0900
0109 00007b      getline    =common1
0110 00007b      dislin     =common1+_dislin-_getline
0111 00007b      dislin1    =common1+_dislin1-_getline
0113 00007b      findbuff   =$1300            suchstring aufbewahren
0114 00007b      such_tab   =$1400            verschiebe-tabelle beim strin
                                              gsuchen
0116 00007b                 *=$1e00
0117                        ;verwaltungsinformation
0118 001e00      zeizhl     *=*+2             ;zeilenzähler, dezimal high/l
                                              ow, ass
0119 001e02      nfeld      *=*+2             feld f. kuenstl. zeilen-numme
                                              rn, ed
0120 001e04      cursavez   *=*+1             merke cursorzeile
0121 001e05      cursaves   *=*+1             merke cursorspalte
0122 001e06      fundort    *=*+2             bei find: beginn des gefunden
                                              en strings
0123 001e08      cutflag    *=*+1             0=scratchpad ist leer, sonst
                                              gefuellt
0124 001e09      cutfrom    *=*+2             startadresse block-kopieren
0125 001e0b      cutto      *=*+2             endadresse block-kopieren
0126 001e0d      blockende  *=*+2             endadresse im scratchpad-spei
                                              cher ab $0f000-0feff
0128                        ;********  konstanten des benutzers
0129 001e0f      anfang     =$1000            beginn des editor-buffers in
                                              bank 0
0131                        ;********  systemroutinen pc-128
0132 001e0f      puthex     =$b8c2            accu als 2 hexbytes ausgeben
0133 001e0f      crsrbl     =$cb21            blinkenden cursor einschalten
0134 001e0f      kstop      =$ffe1            run/stop-taste lesen
0135 001e0f      kunlsn     =$ffae            unlisten auf ieee ausgeben
0136 001e0f      klistn     =$ffb1            dito listen
0137 001e0f      ksecnd     =$ff93            secundaeradresse ausg.
0138 001e0f      kckout     =$ffc9            ausgabekanal oeffnen, la in x
                                               uebergeben
0139 001e0f      kbsout     =$ffd2            zeichen auf aktiven kanal aus
                                              geben
0140 001e0f      kciout     =$ffa8            ein byte aus akku auf ieee-bu
                                              s ausg.
0141 001e0f      error4     =$f685            file not found error
0142 001e0f      kgetin     =$ffe4            hole 1 zeichen aus tastatur-b
                                              uffer
0143 001e0f      kbasin     =$ffcf            zeichen von aktivem kanal hol
                                              en
0144 001e0f      kopen      =$ffc0
```

```
0145 001e0f                kclose    =$ffc3
0146 001e0f                kclrch    =$ffcc              alle kanaele schliessen
0147 001e0f                kprimm    =$ff7d              textausgabe bis zu einer null
0149 001e0f                ktksa     =$ff96              ;sekundaeradresse ausgeben na
                                                         ch talk
0150 001e0f                kuntlk    =$ffab              ;untalk auf ieee-bus ausgeben
0151 001e0f                ktalk     =$ffb4              ;dito talk
0152 001e0f                kchkin    =$ffc6              ;eingabekanal oeffnen
0153 001e0f                kclall    =$ffe7              ;alle files schliessen
0154 001e0f                kplot     =$fff0              cursorposition setzen/lesen
0156 001e0f                          .efi
0157 001e0f                          .fil f1:ass-ed1
0158                                 ;§ko"ass-ed1"
0159 /*
0160 editorprogramm fuer cbm pc-128, copyright 1986 by roland loehr
0161 angepaßt für den Assembler
0162 datum: 14.8.86
0164 befehlstasten sind:
0165 e=texteingabe tastatur
0166 t=topzeile einstellen
0167 b=bottom, hinter letzte zeile gehen
0168 u=up, 1 zeile aufwaerts gehen
0169 d=down, 1 zl. abwaerts gehen
0170 cr=wie down
0171 space=laufende zeile anzeigen
0172 q=quit, zu basic zurueck
0173 k=kill, 1 zl. entfernen
0174 i=insert, 1 zl. zwischenschieben
0175 <=laufender insert mehrerer zeilen
0176 r=read v. floppy hinter vorh. text, device 8
0177 o=read von floppy mitten in den vorh. text
0178 l=list, unterbrechbar mit c= und #=zl-nr
0179 f=find, instring-funktion, stringsuche
0180 w=weiter suchen, fortsetzung von find
0181 c=change, zeile zum neuen editieren aufmachen
0182 C=Cut Zeilenbereich markieren, ins Scratchpad bringen
0183 K=kopieren aus dem scratchpad in den text
0184 a=assembler aufrufen
0185 ?=symboltafel ausgeben
0186 m=sprung zum monitor
0187 s=scratch all text in this bank
0188 p=print auf device 4
0189 &=sequ. file auf floppy schreiben, device 8
0190 #=setze auf zeile mit nummer
0191 */
0192                                  ;**** main program
0194 001e0f                          *=start             in bank 0
0195                                  ;zunaechst die mmu zurechtsetzen
0196 002000 a90e           main      lda #$0e            konfiguration: kernel aktiv,
                                                         ram-bank 0
0197 002002 8d00ff                   sta mmucr           i/o aktiv, ram $0000-$bfff
0198 002005 8d01d5                   sta pcra            dauerhafte praekonfiguration
                                                         fuer bank 0
0199 002008 a97f                     lda #$7f            fuer bank 1: reine ram-bank
0200 00200a 8d02d5                   sta pcrb            preakonfiguration
0201 00200d a901                     lda #1              fast modus einschalten
0202 00200f 8d30d0                   sta $d030
0203 002012 a905                     lda #$05
0204 002014 8d06d5                   sta mmurcr          4k common area im ram unten
0205 002017 a900                     lda #null           null vor den anfang bringen
0206 002019 8d0002                   sta inbuff          null an den anfang
0207 00201c 8d081e                   sta cutflag         scratchpade ist leer
0208 00201f 2021cb                   jsr crsrbl          cursor einschalten
0209 002022 201727                   jsr initbank1       bankpointer einrichten
0211                                 ;****** kommando-ebene des editors
0212 002025 a993           editor    lda #clrscreen      print clear screen
0213 002027 20d2ff                   jsr kbsout
0214 00202a a0ba                     ldy #mes10-mes0
0215 00202c 20b127                   jsr mesout          warmstart abfragen
0216 00202f 204227                   jsr getchr          taste holen
0217 002032 c94a                     cmp #'j'            ja?
0218 002034 f003                     beq editor0
0219 002036 203227                   jsr inittext        leeren editor anlegen
```

```
0220 002039 20b525    editor0   jsr screenclr
0221 00203c 201929              jsr balken          ueberschrift anzeigen
0222 00203f           editor1                       ;kommandoentschluesselung
0223 00203f a900      getbef    lda #0
0224 002041 200008              jsr putchar         eine null an den schluss
0225 002044 204227              jsr getchr          zeichen vom keyboard holen
0226 002047 a21e                ldx #30             zahl-1 der tastatur-befehle
0227 002049 dd5e2d    befpar    cmp tasten,x        erlaubte befehle
0228 00204c f006                beq ongoto
0229 00204e ca                  dex
0230 00204f 10f8                bpl befpar
0231 002051 4c3f20              jmp getbef          keine gueltige taste
0233 002054 8a        ongoto    txa                 multipliziere x mit 2
0234 002055 0a                  asl a
0235 002056 aa                  tax
0236 002057 bd7e2d              lda jmptbl+1,x
0237 00205a 48                  pha
0238 00205b bd7d2d              lda jmptbl,x
0239 00205e 48                  pha
0240 00205f 60                  rts                 execute command via stack
0242                            ;****** hauptschleife der texteingabe
0243 002060 a047      text_ein  ldy #mes1-mes0      2. text ausgeben
0244 002062 20b127              jsr mesout
0246 002065 20b525    inloop    jsr screenclr
0247 002068 207120    inloop1   jsr getstring       eine zeile eingeben
0248 00206b 207527              jsr appendlin
0249 00206e 4c6820              jmp inloop1
0251                            ;**** unterprogramm zur eingabe einer zeile
0252 002071 a900      getstring lda #00             normaler eingabemodus
0253 002073 f002                beq getstr_1
0254 002075 a9ff      getstr_ff lda #$ff            eingabemodus bei change
0255 002077 8514      getstr_1  sta mode
0256 002079 a50f                lda zl_len          alte zeilenlaenge automatisch
                                                     sichern
0257 00207b 8512                sta sav_len
0258 00207d 204227    getstr_2  jsr getchr          zeichen von tastatur holen
0259 002080 a4ec      ausgab    ldy pntr            cursor am zeilenende?
0260 002082 c04f                cpy #79
0261 002084 9006                bcc inlin5          nein
0262 002086 c90d                cmp #cr             kommt zeilenabschluss?
0263 002088 f002                beq inlin5          ja
0264 00208a a99d                lda #$9d            cursor left, mach platz fuer
                                                     'cr'
0265 00208c 48        inlin5    pha
0266 00208d c90d                cmp #cr             invers darzustellen?
0267 00208f d010                bne inlin6
0268 002091 2414                bit mode            normal oder change?
0269 002093 300a                bmi inlin5a         change
0270 002095 a912                lda #revers_on
0271 002097 20d2ff              jsr kbsout
0272 00209a a92f                lda #'/'
0273 00209c 20d2ff              jsr kbsout
0274 00209f a90d      inlin5a   lda #cr
0275 0020a1 20d2ff    inlin6    jsr kbsout          zeichen an bildschirm
0276 0020a4 a900                lda #null           cancel quote mode
0277 0020a6 85f4                sta qtsw            ins flag
0278 0020a8 68        inlin6a   pla
0279 0020a9 c90d                cmp #cr             zeile jetzt uebernehmen?
0280 0020ab f069                beq fini            ja
0281 0020ad 48                  pha
0282 0020ae 297f                and #$7f            strip off
0283 0020b0 c920                cmp #$20            filter fuer steuerzeichen, au
                                                     ch geshiftet
0284 0020b2 b02f                bcs druckbar?
0286 0020b4 68                  pla                 steuerzeichenbehandlung
0287 0020b5 850e                sta lstchr          als vorheriges zeichen
0288 0020b7 c914                cmp #delete
0289 0020b9 d01e                bne ndrul
0290 0020bb 2414                bit mode            change?
0291 0020bd 3001                bmi delchr1         ja
0292 0020bf c8        delchr    iny                 linksverschiebung im buffer
0293 0020c0 c04f      delchr1   cpy #79            zeilenende?
```

```
0294 0020c2 f00d            beq aspace          ja, noch space an das ende
0295 0020c4 b90002          lda inbuff,y
0296 0020c7 88              dey
0297 0020c8 990002          sta inbuff,y
0298 0020cb c6ea            dec indx            zahl benutzte zeichen
0299 0020cd c8              iny
0300 0020ce 4cbf20          jmp delchr
0301 0020d1 a920    aspace  lda #space
0302 0020d3 990002          sta inbuff,y
0303 0020d6 4c7d20          jmp getstr_2
0305 0020d9 c994    ndru1   cmp #insert
0306 0020db d003            bne ndru2
0307 0020dd 205827          jsr inschr          ab cursor rechts ruecken
0308 0020e0 4c7d20  ndru2   jmp getstr_2
0310 0020e3 a50e    druckbar? lda lstchr        ging esc voraus?
0311 0020e5 c91b            cmp #escape
0312 0020e7 d00f            bne store           ; also aspeichern
0313 0020e9 68              pla
0314 0020ea 850e            sta lstchr
0315 0020ec c940            cmp #'@'            quit-befehl? escape/ @
0316 0020ee f003            beq exit
0317 0020f0 4c7d20          jmp getstr_2
0319 0020f3 68      exit    pla                 ende der eingabe
0320 0020f4 68              pla
0321 0020f5 4c3920          jmp editor0
0323 0020f8 24f6    store   bit insflg          insert-mode on?
0324 0020fa 1003            bpl store1          nein
0325 0020fc 205827          jsr inschr          platz machen
0326 0020ff 68      store1  pla
0327 002100 a4ec            ldy pntr            cursorposition
0328 002102 88              dey
0329 002103 990002          sta inbuff,y        in den eingabepuffer
0330 002106 850e            sta lstchr
0331 002108 e6ea            inc indx            zahl uebernommene zeichen
0332 00210a c4ea            cpy indx
0333 00210c b003            bcs inlh
0334 00210e 4c7d20          jmp getstr_2
0336 002111 84ea    inlh    sty indx            jeweils hoechster wert
0337 002113 4c7d20          jmp getstr_2
0338 002116 a4ea    fini    ldy indx            routine ab zeilenabschluss
0339 002118 a900    fini1   lda #0
0340 00211a 990002          sta inbuff,y        00 an den schluss
0341 00211d 840f            sty zl_len
0342 00211f 60              rts                 zeile kann jetzt uebernommen
                                                werden
0345 002120         bottom                      an das textende gehen, taste
                                                b
0346 002120         bottm1
0347 002120         bottom2
0348 002120 20ea27          jsr setbot          nowln=botln
0349 002123 4c3221          jmp up
0351 002126         top                         topzeile ansteuern, taste t
0352 002126 20b525          jsr screenclr
0353 002129 20d927          jsr topno
0354 00212c 20c425          jsr showscr         screen holen und anzeigen mit
                                                 cursor home
0355 00212f 4c3f20          jmp getbef
0357 002132         up                          ;1 zeile aufwaerts im text, t
                                                aste u
0358 002132 200028          jsr prevch          pointer nowln 1 zeichen zurue
                                                ckseetzen
0359 002135 20f025  up0     jsr topzeile?
0360 002138 90ec            bcc top             wir sind vor den anfang gerat
                                                en
0361 00213a 200028  up1     jsr prevch          das zeichen davor holen
0362 00213d d0f6            bne up0
0363 00213f 20f327          jsr add_yreg        nowln wieder +1
0364 002142 a5eb            lda tblx            sind wir in zeile 1?
0365 002144 c901            cmp #1
0366 002146 d009            bne up3             nein
0367 002148 20b525          jsr screenclr
0368 00214b 20c425          jsr showscr         screen anzeigen
```

```
0369 00214e 4c3f20          jmp getbef
0370 002151 20ba25    up3   jsr movcurup      cursor
0371 002154 4c3f20          jmp getbef
0373 002157           down                    ;folgezeile suchen/anzeigen,
                                               taste d
0374 002157 208a08          jsr getchar       was folgt?
0375 00215a d005            bne down1
0376 00215c 20e625          jsr amend?        ende?
0377 00215f b0bf            bcs bottm1        ende des textes
0378 002161 200809    down1 jsr dislin        zeile holen
0379 002164 20f127          jsr addzl_len
0380 002167 a5eb            lda tblx          welche bs-zeile?
0381 002169 c918            cmp #24
0382 00216b f003            beq return        letzte
0383 00216d 4c3f20          jmp getbef
0385 002170           return                  ;laufende zeile anzeigen, tas
                                               te space
0386 002170 200809          jsr dislin        zeile aus buffer zeigen
0387 002173 20ba25          jsr movcurup      cursor
0388 002176 4c3f20    return1 jmp getbef
0390 002179           plus_scr                ;den folgenden screen anzeige
                                               n, taste +
0391 002179 a6eb            ldx tblx          cursorzeile
0392 00217b e017      pl0   cpx #23
0393 00217d f00a            beq pl1
0394 00217f 200009          jsr getline       laenge der restzeilen berueck
                                               sichtigen
0395 002182 20f127          jsr addzl_len
0396 002185 e8              inx
0397 002186 4c7b21          jmp pl0
0398 002189 20b525    pl1   jsr screenclr
0399 00218c 20c425          jsr showscr       einen screen anzeigen
0400 00218f 4c3f20          jmp getbef
0402 002192 200028    minus_scr jsr prevch    vor die laufende zeile gehen
0403 002195 20f025          jsr topzeile?
0404 002198 b003            bcs minus_sc0     nein
0405 00219a 4c2621          jmp top
0406 00219d 18        minus_sc0 clc           wieviel zeilen zurueck?
0407 00219e a914            lda #20           normale zeilenzahl
0408 0021a0 65eb            adc tblx          cursorzeile, aktuell
0409 0021a2 aa              tax               zaehlschleife
0410 0021a3 20f025    minus_sc1 jsr topzeile?
0411 0021a6 b003            bcs minus_sca     nein
0412 0021a8 4c2621          jmp top
0413 0021ab 200028    minus_sca jsr prevch
0414 0021ae d0f3            bne minus_sc1
0415 0021b0 ca        minus_sc2 dex
0416 0021b1 08              php
0417 0021b2 20f025          jsr topzeile?
0418 0021b5 b004            bcs minus_sc3     nein
0419 0021b7 28              plp
0420 0021b8 4c2621          jmp top           ja
0421 0021bb 28        minus_sc3 plp
0422 0021bc 10e5            bpl minus_sc1
0423 0021be a001            ldy #1            setze auf zeilenanfang
0424 0021c0 20f327          jsr add_yreg      per nowln
0425 0021c3 20b525          jsr screenclr
0426 0021c6 20c425          jsr showscr       screen anzeigen
0427 0021c9 4c3f20          jmp getbef
0429 0021cc a900      quit  lda #0            zu basic zurueck
0430 0021ce 8d00ff          sta mmucr         konfiguration
0431 0021d1 85e5            sta sctop         voller bildschirm
0432 0021d3 a93f            lda #$3f          alter zustand
0433 0021d5 8d01d5          sta pcra          praekonfiguration
0434 0021d8 a904            lda #$04
0435 0021da 8d06d5          sta mmurcr        auch in der ram-konfiguration
0436 0021dd 60        quit1 rts               editor verlassen, basic zurue
                                               ck, taste q
0438 0021de a0cc      loesch ldy #mes12-mes0  text einer bank loeschen, tas
                                               te s
0439 0021e0 20b127          jsr mesout
0440 0021e3 204227          jsr getchr        was wird getippt?
```

```
0441 0021e6 48                pha
0442 0021e7 20d2ff            jsr kbsout          echo
0443 0021ea 20fa25            jsr crlf            zeilenvorschub
0444 0021ed 68                pla                 antwort-zeichen
0445 0021ee c94a              cmp #'j'            ja?
0446 0021f0 f003              beq loeschl
0447 0021f2 4c3f20            jmp editor1
0448 0021f5 203227    loeschl jsr inittext
0449 0021f8 4c3920            jmp editor0
0451 0021fb           kill                        ;angezeigte zeile entfernen,
                                                  taste k
0452 0021fb 20e625            jsr amend?
0453 0021fe 9003              bcc kill1
0454 002200 4c2021            jmp bottom          schon am ende
0455 002203 200009    kill1   jsr getline         zeilenlaenge holen
0456 002206 207628            jsr kilsub          zeile im buffer entfernen
0457 002209 a91b              lda #escape
0458 00220b 20d2ff            jsr kbsout          zeilen heranrücken
0459 00220e a944              lda #'d'            zeileninhalt loeschen u. ranr
                                                  uecken
0460 002210 20d2ff            jsr kbsout
0461 002213 4c3f20            jmp getbef          befehlsebene
0463 002216 a91b      nsert   lda #escape         leerzeile schaffen, taste i
0464 002218 20d2ff            jsr kbsout
0465 00221b a949              lda #'i'
0466 00221d 20d2ff            jsr kbsout          i leerzeile schaffen
0467 002220 a900              lda #0              bedeutet: nur 1 zeile einfueg
                                                  en
0468 002222 852f      nsert1  sta ins_flag
0469 002224 207120    nsert2  jsr getstring       eingabezeile von tastatur hol
                                                  en
0470 002227 203122            jsr mov_pack        routinen zum eingliedern
0471 00222a 242f              bit ins_flag        modus?
0472 00222c 30f6              bmi nsert2          laufende eingabe
0473 00222e 4c7021            jmp return          folgezeile anzeigen
0475 002231 201028    mov_pack jsr movback        fuege eingabezeile in den tex
                                                  tbuffer ein
0476 002234 201f08            jsr tonowl          zeile bei nowln einfuegen
0477 002237 206328            jsr korrbotln       zeiger berichtigen
0478 00223a 4cf127            jmp addzl_len       setze nowln auf folgende z.,
                                                  rts dort
0480 00223d a9ff      readnsert lda #$ff          read file, insert ab nowln. t
                                                  aste o
0481 00223f 4ce322            jmp read_in         weiter wie bei read
0483 002242 20b525    lin_nsert jsr screenclr
0484 002245 a9ff              lda #$ff            signalisiere insertmodus
0485 002247 d0d9              bne nsert1          laufende zeilen-einschiebung,
                                                  taste <
0487 002249           writ_flop                   write file to floppy, taste %
0488 002249 20ccff            jsr kclrch          alles erst schliessen
0489 00224c 20b525            jsr screenclr       saubere leinwand!
0490 00224f 208a27            jsr open_flop       open floppy
0491 002252 20aeff            jsr kunlsn
0492 002255 a6b8              ldx la              logische adresse
0493 002257 20c9ff            jsr kckout          logisches file oeffnen
0494 00225a a908              lda #8              device
0495 00225c 859a              sta dflto
0496 00225e a903              lda #3
0497 002260 2093ff            jsr ksecnd          sekundaeradresse
0498 002263 a900              lda #0
0499 002265 8590              sta status
0500 002267 20e625    writ_f1 jsr amend?
0501 00226a b02f              bcs writ_end        ja
0502 00226c 200009    writ_f2 jsr getline         zeile in den eingabepuffer
0503 00226f 20f127            jsr addzl_len
0504 002272 a000              ldy #0
0505 002274 8413      writ_f3 sty temp
0506 002276 a5ba              lda fa
0507 002278 20b1ff            jsr klistn
0508 00227b a5b9              lda sa
0509 00227d 2093ff            jsr ksecnd
0510 002280 a413              ldy temp
```

```
0511 002282 b90002           lda inbuff,y      hole zeichen
0512 002285 d002             bne writ_f4
0513 002287 a90d             lda #cr           cr statt 00 ausgeben
0514 002289 48       writ_f4 pha
0515 00228a 20a8ff           jsr kciout        ausgabe auf device ieee
0516 00228d 20aaff           jsr kunlsn
0517 002290 68               pla
0518 002291 a413             ldy temp
0519 002293 c8               iny
0520 002294 c90d             cmp #cr
0521 002296 d0dc             bne writ_f3
0522 002298 4c6722           jmp writ_f1
0524 00229b 4c2f23   writ_end jmp readend      file schliessen
0526 00229e 20ccff   print    jsr kclrch        printer-ausgabe per 'p', clos
                                                e channels
0527 0022a1 a904             lda #4            device 4
0528 0022a3 20b1ff           jsr klistn
0529 0022a6 a9ff             lda #255
0530 0022a8 2093ff           jsr ksecnd        ;ohne sekundaeradresse
0531 0022ab 20e1ff   print_1 jsr kstop         ;stop-taste gedrueckt?
0532 0022ae f026             beq print_end     ;ja
0533 0022b0 a5fc             lda nowln+1
0534 0022b2 c5fe             cmp botln+1
0535 0022b4 d006             bne print_1a
0536 0022b6 a5fb             lda nowln
0537 0022b8 c5fd             cmp botln
0538 0022ba f01a             beq print_end
0539 0022bc 200009   print_1a jsr getline      zeile holen
0540 0022bf 20f127           jsr addzl_len     nowln schon erhoehen
0541 0022c2 a000             ldy #0
0542 0022c4 b90002   print_2  lda inbuff,y
0543 0022c7 d002             bne print_3
0544 0022c9 a90d             lda #cr           statt 00 ein cr ausgeben
0545 0022cb 20a8ff   print_3  jsr kciout        auf den bus
0546 0022ce c8               iny
0547 0022cf c90d             cmp #cr
0548 0022d1 d0f1             bne print_2
0549 0022d3 4cab22           jmp print_1
0550 0022d6 a90d     print_end lda #cr
0551 0022d8 20a8ff           jsr kciout
0552 0022db 20aaff           jsr kunlsn        geraet abschalten
0553 0022de 4c7021           jmp return        zeile nochmals anzeigen
0555 0022e1          read                      ;text einziehen vom seq file,
                                                 taste r
0556 0022e1 a900             lda #0            modus mit anhaengen, sonst in
                                                sert-modus mit $ff
0557 0022e3 852f     read_in sta ins_flag
0558 0022e5 20b525           jsr screenclr     saubere leinwand
0559 0022e8 208a27           jsr open_flop
0560 0022eb a6b8             ldx la
0561 0022ed 20c6ff           jsr kchkin
0562 0022f0 a000     readf0  ldy #0
0563 0022f2 840f             sty zl_len
0564 0022f4 20e4ff   read1   jsr kgetin
0565 0022f7 a8               tay               zeichen sichern
0566 0022f8 a590             lda status        ggfs. file unbekannt
0567 0022fa 4a               lsr a
0568 0022fb 4a               lsr a
0569 0022fc b04c             bcs readerr       fehler
0570 0022fe 98               tya               zeichen zurueck
0571 0022ff c903             cmp #$03          steuerzeichen?
0572 002301 d005             bne read3
0573 002303 20e4ff   readfile jsr kgetin
0574 002306 f0fb     read2   beq readfile
0575 002308 a40f     read3   ldy zl_len
0576 00230a c90d             cmp #cr
0577 00230c d002             bne read4
0578 00230e a900             lda #0            cr zu null machen
0579 002310 990002   read4   sta inbuff,y
0580 002313 e60f             inc zl_len
0581 002315 2490             bit status
0582 002317 7016             bvs readend
0583 002319 c900             cmp #00
```

```
0584 00231b d0e6          bne readfile
0585 00231d 242f          bit ins_flag      insert oder append?
0586 00231f 1008          bpl read_app
0587 002321 c60f          dec zl_len
0588 002323 203122        jsr mov_pack
0589 002326 4cf022        jmp readf0
0590 002329 207527  read_app  jsr appendlin   zeilentransport und pointer e
                                              rhoehen
0591 00232c 4cf022        jmp readf0
0593 00232f 242f    readend   bit ins_flag
0594 002331 1008          bpl readenda
0595 002333 c60f          dec zl_len
0596 002335 203122        jsr mov_pack      insert-mode
0597 002338 4c3e23        jmp readend1
0598 00233b 207527  readenda  jsr appendlin   eof erkannt, zeile noch anhae
                                              ngen
0599 00233e a901    readend1  lda #1          log. file-#
0600 002340 38      readendc  sec
0601 002341 20c3ff        jsr kclose        close eingabekanal
0602 002344 20ccff        jsr kclrch
0603 002347 4c3920        jmp editor0
0605 00234a 2085f6  readerr   jsr error4     file not found
0606 00234d 4c3e23        jmp readend1      close + befehlsebene
0608 002350 20d927  nummer    jsr topno       zeile mit nummer aufsuchen, t
                                              aste #
0609 002353 a900          lda #0
0610 002355 8d021e        sta nfeld
0611 002358 8d031e        sta nfeld+1       zeilenzaehler=0
0612 00235b 8528          sta such_pntr     vorgabezaehler =0
0613 00235d 8529          sta such_pntr+1   dezimale zaehlung
0614 00235f 20b525        jsr screenclr
0615 002362 a0e2          ldy #mes13-mes0   aufforderung nummer einzugebe
                                              n
0616 002364 20b127        jsr mesout
0617 002367 207120        jsr getstring     zeile einholen
0618 00236a a000          ldy #0
0619 00236c b90002  number0   lda inbuff,y    was wurde eingegeben?
0620 00236f f01c          beq numsuch       ende der zeile
0621 002371 c8            iny
0622 002372 c920          cmp #space
0623 002374 f0f6          beq number0       uebergehe fuehrende spaces
0624 002376 20ff25        jsr dezfilter     dezimale zahl?
0625 002379 9012          bcc numsuch       ende dezimalstring
0626 00237b 290f          and #$0f          zifferteil isolieren
0627 00237d a203          ldx #3            4 linksverschiebungen
0628 00237f 0628    number1   asl such_pntr
0629 002381 2629          rol such_pntr+1
0630 002383 ca            dex
0631 002384 10f9          bpl number1
0632 002386 0528          ora such_pntr
0633 002388 8528          sta such_pntr     ziffer eingezogen und addiert
0634 00238a 4c6c23        jmp number0       naechste ziffer
0635 00238d 20e625  numsuch   jsr amend?     bottom?
0636 002390 9006          bcc numsuch1
0637 002392 201929        jsr balken
0638 002395 4c2021        jmp bottom
0640 002398 ad031e  numsuch1  lda nfeld+1     vergleichsschleife
0641 00239b c529          cmp such_pntr+1
0642 00239d d013          bne numsuch2
0643 00239f ad021e        lda nfeld
0644 0023a2 c528          cmp such_pntr     vorgabe
0645 0023a4 d00c          bne numsuch2
0646 0023a6 201929        jsr balken
0647 0023a9 20b525        jsr screenclr     richtig positioniert, saubere
                                              anzeige
0648 0023ac 20c425        jsr showscr       anzeige eines screens mit cur
                                              sor home
0649 0023af 4c3f20        jmp getbef
0650 0023b2 200009  numsuch2  jsr getline     zeile ohne anzeige holen
0651 0023b5 20f127        jsr addzl_len     nowln+laenge
0652 0023b8 20c527        jsr addnfeld      zeile+1
0653 0023bb 4c8d23        jmp numsuch       weiter hindurchbaggern
```

```
0655 0023be            list                    ;text listen, taste l
0656 0023be 208a08     list_l    jsr getchar   zeichen holen
0657 0023c1 f022                 beq list_2
0658 0023c3 200809     list_la   jsr dislin
0659 0023c6 20f127               jsr addzl_len
0660 0023c9 20e4ff               jsr kgetin    irgendwas gedrueckt?
0661 0023cc c900                 cmp #0
0662 0023ce f0ee                 beq list_1
0663 0023d0 c90d                 cmp #cr       cr-taste?
0664 0023d2 f008                 beq list_12
0665 0023d4 20e4ff     list_11   jsr kgetin    wird taste losgelassen
0666 0023d7 d0fb                 bne list_11   warteschleife
0667 0023d9 4cbe23               jmp list_1    weiter listen
0668 0023dc 200809     list_12   jsr dislin    eingestellte zeile noch zeige
                                                n
0669 0023df 20ba25               jsr movcurup
0670 0023e2 4c3f20               jmp getbef
0671 0023e5 20e625     list_2    jsr amend?    ende?
0672 0023e8 90d9                 bcc list_la   nein
0673 0023ea 4c2021     list_2a   jmp bottom
0675 0023ed 20fa25     list_n    jsr crlf
0676 0023f0 a912                 lda #revers_on
0677 0023f2 20d2ff               jsr kbsout
0678 0023f5 a902                 lda #2        schnell den bereich $b000 ein
                                                schalten
0679 0023f7 8d00ff               sta mmucr
0680 0023fa ad031e               lda nfeld+1   ausgabe der zeilenzahl im flu
                                                ge
0681 0023fd 20c2b8               jsr puthex
0682 002400 ad021e               lda nfeld
0683 002403 20c2b8               jsr puthex
0684 002406 8d01ff               sta bank0     normalbetrieb
0685 002409 20fa25               jsr crlf
0686 00240c 20e4ff     list_n1   jsr kgetin
0687 00240f d0fb                 bne list_n1   ,warten bis taste losgelassen
0688 002411 4c3f20               jmp getbef
0689 002414            find                    ;zeichenkette finden, taste f
0690 002414 20b525               jsr screenclr
0691 002417 a06e                 ldy #mes4-mes0
0692 002419 20b127               jsr mesout
0693 00241c 207120               jsr getstring suchstring anfordern
0694 00241f a50f                 lda zl_len
0695 002421 8510                 sta such_len  laenge merken
0696 002423 8511                 sta such_lenw vormerkung f. weit_such
0697 002425 a000                 ldy #0        den suchstring merken
0698 002427 b90002     find1     lda inbuff,y  eingabepuffer
0699 00242a 990013               sta findbuff,y
0700 00242d c8                   iny
0701 00242e c900                 cmp #00       zeilenende?
0702 002430 d0f5                 bne find1
0703 002432 209f28     find2     jsr such      suche durchfuehren
0704 002435 b003                 bcs find2a
0705 002437 4c2021               jmp bottom    nichts (mehr) gefunden
0706 00243a a5fb       find2a    lda nowln
0707 00243c 8d061e               sta fundort   stelle merken
0708 00243f a5fc                 lda nowln+1
0709 002441 8d071e               sta fundort+1
0710 002444 200028     find2b    jsr prevch    gehe zum zeilenanfang zurueck
0711 002447 d0fb                 bne find2b    pointer zeigt noch nicht auf
                                                eine null
0712 002449 a001                 ldy #1        setze auf anfang zeile
0713 00244b 20f327               jsr add_yreg
0714 00244e 20b525               jsr screenclr
0715 002451 20c425               jsr showscr   einen screen von hier ab anze
                                                igen
0716 002454 4c3f20               jmp getbef
0718 002457 18         weit_such clc           weiter suchen, taste w
0719 002458 ad061e               lda fundort   wo find aufhoerte
0720 00245b 6901                 adc #1        1 stelle weiter
0721 00245d 85fb                 sta nowln
0722 00245f ad071e               lda fundort+1
0723 002462 6900                 adc#0
```

```
0724 002464 85fc            sta nowln+1     pointer um 1 erhoeht, weiter
                                            suchen
0725 002466 20d228          jsr such3
0726 002469 b0cf            bcs find2a      etwas gefunden
0727 00246b 4c2021          jmp bottom
0728 00246e         change                  string ersetzen, taste c
0729 00246e a912            lda #revers_on
0730 002470 20d2ff          jsr kbsout
0731 002473 200809          jsr dislin
0732 002476 20ba25          jsr movcurup
0733 002479 207520          jsr getstr_ff   eingabemodus fuer change
0734 00247c c612            dec sav_len
0735 00247e 38              sec
0736 00247f a50f            lda zl_len
0737 002481 e512            sbc sav_len
0738 002483 d00f            bne change1
0739 002485 201f08          jsr tonowl      zeile gleicher laenge ueberne
                                            hmen
0740 002488 20ba25          jsr movcurup
0741 00248b 200809          jsr dislin      zeile wieder normal anzeigen
0742 00248e 20ba25          jsr movcurup    cursor
0743 002491 4c3f20          jmp getbef
0745 002494 a000    change1 ldy #0          ungleiche string-länge
0746 002496 a5fc            lda nowln+1
0747 002498 c5fe            cmp botln+1
0748 00249a d009            bne change2
0749 00249c a5fb            lda nowln
0750 00249e c5fd            cmp botln
0751 0024a0 d003            bne change2
0752 0024a2 4c2021          jmp bottom      war schon textende, anzeigen
0754 0024a5 c60f    change2 dec zl_len      wegen des begrenzer-zeichens
0755 0024a7 c612            dec sav_len
0756 0024a9 a50f            lda zl_len
0757 0024ab 8513            sta temp        datensicherung
0758 0024ad 38              sec
0759 0024ae a50f            lda zl_len
0760 0024b0 e512            sbc sav_len
0761 0024b2 901d            bcc raffen      neuer string ist kuerzer
0762 0024b4 850f            sta zl_len      nur um die differenz nach rec
                                            hts ruecken
0763 0024b6 c60f            dec zl_len
0764 0024b8 201028          jsr movback     es wird gerueckt
0765 0024bb 206328          jsr korrbotln   zeiger botln korrigieren
0766 0024be a513    change21 lda temp       laenge eingabestring
0767 0024c0 850f            sta zl_len      wiederherstellen
0768 0024c2 201f08          jsr tonowl      transport in den speicher
0769 0024c5 20ba25          jsr movcurup    cursor
0770 0024c8 200809          jsr dislin      zeile wieder normal zeigen
0771 0024cb 20ba25          jsr movcurup    cursor
0772 0024ce 4c3f20          jmp getbef
0774 0024d1 38      raffen  sec
0775 0024d2 a512            lda sav_len     versatzmenge fuer kilsub
0776 0024d4 e513            sbc temp
0777 0024d6 850f            sta zl_len
0778 0024d8 a5fc            lda nowln+1     korrigierte zeile ist kuerzer
0779 0024da 48              pha             sichern
0780 0024db a5fb            lda nowln
0781 0024dd 48              pha             sichern
0782 0024de 18              clc
0783 0024df 6513            adc temp        laenge eingabezeile
0784 0024e1 6902            adc #2
0785 0024e3 85fb            sta nowln       zieladresse b. verschiebung
0786 0024e5 a5fc            lda nowln+1
0787 0024e7 6900            adc #0
0788 0024e9 85fc            sta nowln+1
0789 0024eb 207e28          jsr kilsub1     block move ausfuehren
0790 0024ee 68              pla
0791 0024ef 85fb            sta nowln
0792 0024f1 68              pla
0793 0024f2 85fc            sta nowln+1     alter pointer w. hergestellt
0794 0024f4 4cbe24          jmp change21
0796 0024f7 a900    monitor lda #0
0797 0024f9 85e5            sta sctop       volles fenster
```

```
0798 0024fb 20b525          jsr screenclr       aktion saubere mattscheibe
0799 0024fe a900            lda #0
0800 002500 8d00ff          sta mmucr           alles fuer basic, kernel eins
                                                chalten
0801 002503 a93f            lda #$3f            alter zustand
0802 002505 8d01d5          sta pcra            praekonfiguration
0803 002508 a904            lda #$04
0804 00250a 8d06d5          sta mmurcr          auch in der ram-konfiguration
0805 00250d 4c23b0          jmp $b023           exit to monitor with 'm'
0807 002510                 .efi
0808 002510                 .fil fl:ed3
0809                        ;$ko"ed3"
0810                        ;2. teil des editorprogrammes fuer cbm pc-128
0812 002510         cut                         ;einen textblock markieren, i
                                                ns scratchpad bringen, taste
                                                C
0813 002510 a900            lda #00
0814 002512 8560            sta to              pointer auf $f000 einrichten
0815 002514 a9f0            lda #$f0
0816 002516 8561            sta to+1
0817 002518 8d081e          sta cutflag         =buffer benutzt
0818 00251b a912    cut1    lda #revers_on
0819 00251d 20d2ff          jsr kbsout
0820 002520 209809          jsr dislin          zeile invers anzeigen
0821 002523 20f127          jsr addzl_len
0822 002526 a93f            lda #$3f            aktiviere vollen ram-bereich
0823 002528 8d00ff          sta mmucr           bank0
0824 00252b a0ff            ldy #$ff
0825 00252d c8      cutm    iny
0826 00252e b90002          lda inbuff,y        transportschleife
0827 002531 9160            sta (to),y
0828 002533 d0f8            bne cutm            transportschleife ins scratch
                                                pad
0829 002535 8d01ff          sta bank0           alte konfiguration
0830 002538 18              clc                 pointer erhoehen
0831 002539 c8              iny
0832 00253a 98              tya                 zeilenlaenge
0833 00253b 6560            adc to              zielpointer
0834 00253d 8560            sta to
0835 00253f 8d0b1e          sta cutto           merkadresse blockende
0836 002542 a561            lda to+1
0837 002544 6900            adc #0
0838 002546 8561            sta to+1
0839 002548 8d0c1e          sta cutto+1
0840 00254b c9fe            cmp #$fe            grenze?
0841 00254d f00b            beq cutend
0843 00254f 204227  cut2    jsr getchr          taste holen
0844 002552 c90d            cmp #cr             cursor down oder cr?
0845 002554 f0c5            beq cut1            weiter protokollieren
0846 002556 c9c3            cmp #'C'            als abschluss
0847 002558 d0f5            bne cut2
0848 00255a 4c3f20  cutend  jmp getbef
0850 00255d         copy                        ;einen block vom scratchpad e
                                                inziehen, taste K
0851 00255d 2c081e          bit cutflag         ist was im scratchpad?
0852 002560 3003            bmi copy1           ja
0853 002562 4c3f20          jmp getbef
0854 002565 a900    copy1   lda #0
0855 002567 8528            sta such_pntr       quellpointer
0856 002569 a9f0            lda #$f0            auf scratchpad richten
0857 00256b 8529            sta such_pntr+1
0858 00256d a5fc            lda nowln+1         pointer sichern
0859 00256f 48              pha
0860 002570 a5fb            lda nowln
0861 002572 48              pha
0862 002573 a93f    copy1a  lda #$3f            volle ram-bank 0 einrichten
0863 002575 8d00ff          sta mmucr
0864 002578 a000            ldy #0              transportschleife einrichten
0865 00257a b128    copy2   lda (such_pntr),y   quelle/ziel in bank0
0866 00257c 990002          sta inbuff,y
0867 00257f c8              iny
0868 002580 c900            cmp #0
0869 002582 d0f6            bne copy2           kein zeilenende
```

```
0870 002584 8d01ff              sta bank0           alte konfiguration
0871 002587 18                  clc
0872 002588 98                  tya                 quellpointer erhoehen
0873 002589 6528                adc such_pntr
0874 00258b 8528                sta such_pntr
0875 00258d a529                lda such_pntr+1
0876 00258f 6900                adc #0
0877 002591 8529                sta such_pntr+1
0879 002593 88                  dey
0880 002594 840f                sty zl_len          wird spaeter wieder erhoeht
0881 002596 203122              jsr mov_pack        zeile einfuegen, nowln, botln
                                                     erhoehen
0882 002599 38                  sec
0883 00259a a528                lda such_pntr       sind wir am ende scratchpad?
0884 00259c ed0b1e              sbc cutto
0885 00259f a529                lda such_pntr+1
0886 0025a1 ed0c1e              sbc cutto+1
0887 0025a4 90cd                bcc copy1a
0888 0025a6 68                  pla                 alten zeilenpointer holen
0889 0025a7 85fb                sta nowln           zeigt jetzt auf beginn
0890 0025a9 68                  pla
0891 0025aa 85fc                sta nowln+1         der einfuegung
0892 0025ac 20b525              jsr screenclr
0893 0025af 20c425              jsr showscr
0894 0025b2 4c3f20              jmp getbef
0896                            ;***** diverse unterprogramme
0897 0025b5 a993        screenclr lda #clrscreen    bildschirm loeschen
0898 0025b7 4cd2ff              jmp kbsout
0899 0025ba a991        movcurup  lda #cursor_up
0900 0025bc 4cd2ff              jmp kbsout
0901 0025bf a911        movcurdwn lda #curs_down
0902 0025c1 4cd2ff              jmp kbsout
0903 0025c4 a5fb        showscr   lda nowln           einen screen mit cursor in ze
                                                     ile 1 zeigen
0904 0025c6 48                  pha                 zeiger retten
0905 0025c7 a5fc                lda nowln+1
0906 0025c9 48                  pha
0908 0025ca a216                ldx #22             23 zeilen weitergehen, 1 scre
                                                     en, taste +
0909 0025cc 20e625      sh_sc1    jsr amend?
0910 0025cf b009                  bcs sh_sc3
0911 0025d1 200809      sh_sc2    jsr dislin
0912 0025d4 20f127                jsr addzl_len     zeilenlaenge dazu
0913 0025d7 ca                    dex
0914 0025d8 10f2                  bpl sh_sc1
0916 0025da 68          sh_sc3    pla
0917 0025db 85fc                  sta nowln+1       zeiger zurueck
0918 0025dd 68                    pla
0919 0025de 85fb                  sta nowln
0920 0025e0 a913                  lda #home         home cursor
0921 0025e2 20d2ff                jsr kbsout
0922 0025e5 60                    rts
0924 0025e6 38          amend?    sec               hat nowln das textende erreic
                                                     ht?
0925 0025e7 a5fb                  lda nowln
0926 0025e9 e5fd                  sbc botln
0927 0025eb a5fc                  lda nowln+1
0928 0025ed e5fe                  sbc botln+1
0929 0025ef 60                    rts               carry set, wenn ja
0930 0025f0 38          topzeile? sec               hat nowln vor den beginn zuru
                                                     eckgesetzt?
0931 0025f1 a5fb                  lda nowln
0932 0025f3 e900                  sbc #<anfang      definition des textspeichers
0933 0025f5 a5fc                  lda nowln+1
0934 0025f7 e910                  sbc #>anfang
0935 0025f9 60                    rts               carry clear, wenn unterschrit
                                                     ten
0937 0025fa a90d        crlf      lda #cr           cr/lf auf aktiven kanal ausge
                                                     ben
0938 0025fc 4cd2ff                jmp kbsout        rts dort
0940 0025ff c930        dezfilter cmp #'0'          filter auf dezimalzahlen
0941 002601 9005                  bcc dezz          carry clear: keine dezimalzah
                                                     l
```

```
0942 002603 c93a            cmp #$3a          groesser '9'?
0943 002605 b002            bcs dezg          ja
0944 002607 38              sec
0945 002608 60        dezz  rts
0946 002609 18        dezg  clc
0947 00260a 60              rts
0949                        ;######## routinen zm transport in das common r
0949                        am
0950                        ;per putloop und putloop1
0951 00260b a000      _putchar ldy #0           1 byte ablegen
0952 00260d 8d02ff    _putchar1 sta bank1       parameter-bank
0953 002610 91fd            sta (botln),y
0954 002612 8d01ff          sta bank0           programmbank
0955 002615 c8              iny
0956 002616 60              rts
0957 002617 a000      _putline ldy #0           transport einer zeile an das
                                                textende
0958 002619 b90002    _putline1 lda inbuff,y
0959 00261c 08              php
0960 00261d 8d02ff          sta bank1
0961 002620 91fd            sta (botln),y
0962 002622 8d01ff          sta bank0
0963 002625 c8              iny
0964 002626 28              plp
0965 002627 d0f0            bne _putline1
0966 002629 60              rts
0968 00262a a000      _tonowl ldy #0            transport einer zeile an die
                                                laufende stelle
0969 00262c b90002    _tonowl1 lda inbuff,y
0970 00262f 08              php
0971 002630 8d02ff          sta bank1
0972 002633 91fb            sta (nowln),y
0973 002635 8d01ff          sta bank0
0974 002638 c8              iny
0975 002639 28              plp
0976 00263a d0f0            bne _tonowl1

0977 00263c 60              rts
0979 00263d a200      _such4  ldx #0            schluesselposition, suchrouti
                                                ne
0980 00263f 8d02ff    _such4a sta bank1
0981 002642 b1fb            lda (nowln),y     textzeichen
0982 002644 8d01ff          sta bank0
0983 002647 d90013          cmp findbuff,y    suchstring
0984 00264a f006            beq _such4b
0985 00264c a8              tay               zeichen als verschiebeindex m
                                                erken
0986 00264d 68              pla               stack berichtigen
0987 00264e 68              pla
0988 00264f 4cd929          jmp such5
0989 002652 e8        _such4b inx
0990 002653 88              dey
0991 002654 10e9            bpl _such4a       noch nicht alles bytes
0992 002656 8d01ff          sta bank0
0993 002659 38              sec               string gefunden
0994 00265a 60              rts               gefunden, carry=1
0996 00265b a000      _movblck ldy #0           blockmove
0997 00265d 8d02ff          sta bank1         verschiebung in bank1
0998 002660 b126      _movblck1 lda (movptr),y transport in bank1
0999 002662 91fb            sta (nowln),y     bei kill von rechts nach link
                                                s
1000 002664 c8              iny
1001 002665 c900            cmp #0
1002 002667 d0f7            bne _movblck1
1003 002669 98              tya               zwei pointer erhoehen
1004 00266a 18              clc
1005 00266b 6526            adc movptr
1006 00266d 8526            sta movptr
1007 00266f a527            lda movptr+1
1008 002671 6900            adc #0
1009 002673 8527            sta movptr+1
1010 002675 98              tya
1011 002676 18              clc               pointer erhoehen
```

```
1012 002677 65fb            adc nowln
1013 002679 85fb            sta nowln
1014 00267b a5fc            lda nowln+1
1015 00267d 6900            adc #0
1016 00267f 85fc            sta nowln+1
1017 002681 38              sec                 textende?
1018 002682 a526            lda movptr
1019 002684 e5fd            sbc botln
1020 002686 a527            lda movptr+1
1021 002688 e5fe            sbc botln+1
1022 00268a 90cf            bcc _movblck        nein
1023 00268c a900            lda #0
1024 00268e a8              tay
1025 00268f 91fb            sta (nowln),y       0 anhaengen
1026 002691 8d01ff          sta bank0
1027 002694 60              rts
1028 002695 a000    _getchar  ldy #0            1 byte aus der textbank holen
1029 002697 8d02ff  _getchar1 sta bank1         parameter-bank
1030 00269a b1fb            lda (nowln),y
1031 00269c 08              php                 status 00?
1032 00269d 8d01ff          sta bank0           programmbank
1033 0026a0 c8              iny
1034 0026a1 28              plp                 status
1035 0026a2 60              rts
1036 0026a3 8d02ff  _movback3 sta bank1
1037 0026a6 b1fb    _movback4 lda (nowln),y     transport nun in der page
1038 0026a8 91fd            sta (botln),y
1039 0026aa 88              dey
1040 0026ab d0f9            bne _movback4
1042 0026ad b1fb            lda (nowln),y       nun das byte mit y=0
1043 0026af 91fd            sta (botln),y
1044 0026b1 8d01ff          sta bank0
1045 0026b4 60              rts                 und weiter nach tonowl
1047 0026b5 8d02ff  _movbackr sta bank1
1048 0026b8 a0ff    _movback8 ldy #$ff          1 volle page movan
1049 0026ba b1fd    _movback9 lda (botln),y
1050 0026bc 9126            sta (movptr),y
1051 0026be 88              dey
1052 0026bf d0f9            bne _movback9
1054 0026c1 b1fd            lda (botln),y       das unterste byte zuletzt
1055 0026c3 9126            sta (movptr),y
1056 0026c5 c6fe            dec botln+1
1057 0026c7 c627            dec movptr+1
1058 0026c9 c62e            dec sympag          volle pages fuer move
1059 0026cb d0eb            bne _movback8       weitere pages moven
1061 0026cd e6fe            inc botln+1         korrektur, unterlauf
1062 0026cf 8d01ff          sta bank0
1063 0026d2 4c2c28  _mb9    jmp movback2
1065 0026d5 a9ff    _getline lda #$ff           hole eine zeile, ohne sie anz
                                                uzeigen
1066 0026d7 8532            sta disflag
1067 0026d9 a000            ldy #0
1068 0026db f002            beq _dislin1
1069 0026dd a000    _dislin   ldy #0            anzeige einer zeile und trans
                                                port nach inbuff
1070 0026df 8d02ff  _dislin1  sta bank1         umschaltung
1071 0026e2 b1fb            lda (nowln),y
1072 0026e4 990002          sta inbuff,y
1073 0026e7 f00e            beq _dislin2        zeilenende
1074 0026e9 8d01ff          sta bank0           programmbank aktiv
1075 0026ec 2432            bit disflag
1076 0026ee 3003            bmi _dislin1a
1077 0026f0 20d2ff          jsr kbsout          anzeige
1078 0026f3 c8      _dislin1a iny
1079 0026f4 4c0a09          jmp dislin1
1080 0026f7 8d01ff  _dislin2  sta bank0
1081 0026fa c8              iny
1082 0026fb 84ea            sty indx
1083 0026fd 840f            sty zl_len
1084 0026ff 2432            bit disflag         anzeigen?
1085 002701 300f            bmi _dislin3
1086 002703 a912            lda #revers_on      zeilenende zeigen
1087 002705 20d2ff          jsr kbsout
```

```
1088 002708 a92f            lda #'/'
1089 00270a 20d2ff          jsr kbsout
1090 00270d a90d            lda #cr
1091 00270f 20d2ff          jsr kbsout
1092 002712 a900      _dislin3 lda #0
1093 002714 8532            sta disflag
1094 002716 60        _dislin4 rts
1096 002717 a200      initbank1 ldx #0              routinen ins common ram bring
                                                    en
1097 002719 bd0b26    putloop   lda _putchar,x      in das common ram
1098 00271c 9d0009              sta common,x
1099 00271f e8                  inx
1100 002720 e0ca                cpx #_getline-_putchar
1101 002722 d0f5                bne putloop
1102 002724 a200                ldx #0              2. transport
1103 002726 bdd526    putloop1  lda _getline,x
1104 002729 9d0009              sta common1,x
1105 00272c e8                  inx
1106 00272d e042                cpx #initbank1-_getline
1107 00272f d0f5                bne putloop1
1108 002731 60                  rts
1110 002732 20d927    inittext  jsr topno           stelle topzeile ein
1111 002735 85fd                sta botln           pointer auf ende textspeicher
1112 002737 86fe                stx botln+1
1113 002739 a900                lda #0
1114 00273b 200008              jsr putchar         0 an den textanfang
1115 00273e 200208              jsr putchar1        2. null
1116 002741 60                  rts
1118 002742 20e4ff    getchr    jsr kgetin          zeichen aus tastaturpuffer ho
                                                    len
1119 002745 c900                cmp #null           leer?
1120 002747 f0f9                beq getchr
1121 002749 c913                cmp #$13            home? abfiltern
1122 00274b f0f5                beq getchr
1123 00274d c993                cmp #$93            clear/home?
1124 00274f f0f1                beq getchr
1125 002751 c911                cmp #$11            cursor down zu cr machen
1126 002753 d002                bne getend
1127 002755 a90d                lda #cr
1128 002757 60        getend    rts
1130                            ;******* verschiebungen bei insert
1131 002758 a04f      inschr    ldy #79             max-spalte
1132 00275a c4ea                cpy indx            schon erreicht?
1133 00275c f016                beq insend          ja
1134 00275e 88        inslop    dey                 zeige auf zeichen davor
1135 00275f b90002               lda inbuff,y
1136 002762 c8                  iny                 zeige auf stelle dahinter
1137 002763 990002              sta inbuff,y
1138 002766 88                  dey
1139 002767 c4ec                cpy pntr            insert-stelle erreicht?
1140 002769 d0f3                bne inslop
1141 00276b 24f6                bit insflg          im insert-mode keinen space!
1142 00276d 3005                bmi insend
1143 00276f a920                lda #space          dort space einsetzen
1144 002771 990002              sta inbuff,y
1145 002774 60        insend    rts
1147 002775 200c08    appendlin jsr putline         transport der eingegebenen ze
                                                    ile
1148 002778 98                  tya                 laenge der zeichenkette
1149 002779 18                  clc                 zum pointer botln addieren
1150 00277a 65fd                adc botln
1151 00277c 85fd                sta botln
1152 00277e a5fe                lda botln+1
1153 002780 6900                adc #null
1154 002782 85fe                sta botln+1
1155 002784 a900                lda #0              00 als delimiter an den schlu
                                                    ss
1156 002786 200008              jsr putchar
1157 002789 60                  rts
1159 00278a a086      open_flop ldy #mes6-mes0      open floppy channel 1,3,8
1160 00278c 20b127              jsr mesout
1161 00278f 207120              jsr getstring       zeile holen
```

```
1162 002792 84b7                sty fnlen           laenge filename
1163 002794 a900                lda #<inbuff
1164 002796 85bb                sta fnadr
1165 002798 a902                lda #>inbuff
1166 00279a 85bc                sta fnadr+1         zeige auf filename
1167 00279c a901                lda #1              log file
1168 00279e 85b8                sta la
1169 0027a0 a903                lda #3
1170 0027a2 85b9                sta sa
1171 0027a4 a908                lda #8
1172 0027a6 85ba                sta fa
1173 0027a8 a900                lda #0
1174 0027aa 8590                sta status
1175 0027ac 18                  clc
1176 0027ad 20c0ff              jsr kopen           open 1,3,8, filename
1177 0027b0 60                  rts
1178                            ;***** interaktive ausgaben
1180 0027b1 200129    mesout    jsr poslin0         fenster in zeile 0
1181 0027b4 b9522c    mesout1   lda mes0,y
1182 0027b7 f007                beq txtend
1183 0027b9 20d2ff              jsr kbsout
1184 0027bc c8                  iny
1185 0027bd 4cb427              jmp mesout1
1186 0027c0 a901      txtend    lda #01             fenster wieder ab zeile 1
1187 0027c2 85e5                sta sctop
1188 0027c4 60                  rts
1190 0027c5 f8        addnfeld  sed                 dezimal arbeiten, zeilenzahl+
                                                    1
1191 0027c6 18                  clc
1192 0027c7 ad021e              lda nfeld           =1 zeile zu suchen
1193 0027ca 6901                adc #1
1194 0027cc 8d021e              sta nfeld
1195 0027cf ad031e              lda nfeld+1
1196 0027d2 6900                adc #0
1197 0027d4 8d031e              sta nfeld+1
1198 0027d7 d8                  cld                 wieder binaer
1199 0027d8 60                  rts
1201 0027d9 a900      topno     lda #0
1202 0027db 8d021e              sta nfeld           zeilenzaehler=0
1203 0027de 8d031e              sta nfeld+1
1204 0027e1 a900                lda #<anfang        richte textspeicher ein
1205 0027e3 a210                ldx #>anfang
1206 0027e5 85fb      topno1    sta nowln           zeiger auf topzeile
1207 0027e7 86fc                stx nowln+1
1208 0027e9 60                  rts
1210 0027ea a5fd      setbot    lda botln           setze nowln auf textende
1211 0027ec a6fe                ldx botln+1
1212 0027ee 4ce527              jmp topno1
1214 0027f1 a40f      addzl_len ldy zl_len          nowln+zeilenlaenge
1215 0027f3 18        add_yreg  clc                 addiere y zu nowln
1216 0027f4 98                  tya
1217 0027f5 65fb                adc nowln
1218 0027f7 85fb                sta nowln
1219 0027f9 a5fc                lda nowln+1
1220 0027fb 6900                adc #0
1221 0027fd 85fc                sta nowln+1
1222 0027ff 60                  rts
1224 002800 38        prevch    sec                 erniedrige nowln
1225 002801 a5fb                lda nowln           subtrahiere jeweils 1
1226 002803 e901                sbc #1
1227 002805 85fb                sta nowln
1228 002807 a5fc                lda nowln+1
1229 002809 e900                sbc #0
1230 00280b 85fc                sta nowln+1         pointer erniedrigt
1231 00280d 4c8a08              jmp getchar         hole 1 zeichen und rts
1233                            ;***** 1 zeile in textspeicher einfuegen
1234 002810           movback                       ;eine zeile in den textspeich
                                                    er einfuegen.
1235 002810 e60f                inc zl_len          =wirkliche insert-menge
1236 002812 a5fd                lda botln           begrenzung bei $ff80 abfragen
1237 002814 e980                sbc #$80
1238 002816 a5fe                lda botln+1
```

```
1239 002818 e9ff            sbc #$ff
1240 00281a 9008            bcc movback1
1241 00281c a0fb            ldy #mes14-mes0     speicher voll
1242 00281e 20b127          jsr mesout
1243 002821 4c3e23          jmp readend1        close file
1245 002824 a5fe    movback1 lda botln+1         vorlaeufiges tafelende retten
1246 002826 852d            sta symsav+1
1247 002828 a5fd            lda botln
1248 00282a 852c            sta symsav
1250 00282c 38      movback2 sec
1251 00282d a5fd            lda botln
1252 00282f e5fb            sbc nowln
1253 002831 a8              tay                 zaehler in der page
1254 002832 a5fe            lda botln+1         bestimme zahl pages
1255 002834 e5fc            sbc nowln+1
1256 002836 d017            bne movback7        verschiebung voller seiten
1257 002838 98              tya                 menge in der seite=0?
1258 002839 d003            bne movbck2
1259 00283b 4c1f08          jmp tonowl          ; ja put behind
1261 00283e 88      movbck2 dey                 y=$ff
1262 00283f 18              clc
1263 002840 a5fb            lda nowln           ab quelle zl_len stellen rech
                                                ts
1264 002842 650f            adc zl_len          laenge eingabezeile
1265 002844 85fd            sta botln           temp. zielpointer
1266 002846 a5fc            lda nowln+1
1267 002848 6900            adc #0
1268 00284a 85fe            sta botln+1
1269 00284c 4c9808          jmp movback3        im common ram
1271 00284f 852e    movback7 sta sympag         volle pages fuer move
1272 002851 c6fe            dec botln+1
1273 002853 18              clc
1274 002854 a5fd            lda botln           empfangspointer einrichten fu
                                                er verschiebung
1275 002856 650f            adc zl_len
1276 002858 8526            sta movptr
1277 00285a a5fe            lda botln+1
1278 00285c 6900            adc #0
1279 00285e 8527            sta movptr+1
1280 002860 4caa08          jmp movbackr
1282 002863 18      korrbotln clc               botln um zl_len korrigieren
1283 002864 a52c            lda symsav
1284 002866 650f            adc zl_len
1285 002868 85fd            sta botln
1286 00286a a52d            lda symsav+1
1287 00286c 6900            adc #0
1288 00286e 85fe            sta botln+1
1289 002870 a900            lda #00
1290 002872 a8              tay
1291 002873 91fd            sta (botln),y       00-begrenzer an das textende
1292 002875 60              rts                 normaler abschluss
1294                        ;***** 1 zeile im textspeicher entfernen
1295 002876 a5fc    kilsub  lda nowln+1         alten zeilenzeiger retten
1296 002878 852d            sta symsav+1
1297 00287a a5fb            lda nowln
1298 00287c 852c            sta symsav
1300 00287e 18      kilsub1 clc                 transportpointer einrichten m
                                                it
1301 00287f a50f            lda zl_len          aufbewahrte zeilenlaenge
1302 002881 65fb            adc nowln
1303 002883 8526            sta movptr
1304 002885 a5fc            lda nowln+1
1305 002887 6900            adc #0
1306 002889 8527            sta movptr+1
1307 00288b 205008          jsr movblck         routine im common ram, versch
                                                iebung
1309 00288e a5fb    kilend  lda nowln
1310 002890 85fd            sta botln
1311 002892 a5fc            lda nowln+1
1312 002894 85fe            sta botln+1
1313 002896 a52c            lda symsav          alte nowln zurueck
1314 002898 85fb            sta nowln
1315 00289a a52d            lda symsav+1
```

```
1316 00289c 85fc              sta nowln+1
1317 00289e 60                rts
1319                          ;suche suchstring im bereich nowln bis end
1320                          ;nach boyer & moore, micro mag nr. 42-85
1321                          ;algorithmus zum schnelleren stringsuchen
1322                          ;string gefunden: carry=1, nowln=adresse
1323                          ;nicht gefunden: carry=0, nowln>=end
1325 00289f a511      such    lda such_lenw
1326                          ;laengeneintrag in such_tab zunaechst m. string
1326                          laenge
1327 0028a1 8510              sta such_len        dieser parm wird spaeter zers
                                                  toert
1328 0028a3 48                pha                 restore later
1329 0028a4 a000              ldy #0
1330 0028a6 990014    such1   sta such_tab,y
1331 0028a9 c8                iny
1332 0028aa d0fa              bne such1           bis $ff
1333                          ;verschiebedistanz in such_tab
1334 0028ac a200              ldx #0              beginne bei suchstring
1335 0028ae c610              dec such_len
1336 0028b0 a510      such2   lda such_len        eintrag von stringlaenge-1 bi
                                                  s 0
1337 0028b2 bc0013            ldy findbuff,x      buchstabe als index in such_t
                                                  ab
1338 0028b5 990014            sta such_tab,y
1339 0028b8 e8                inx
1340 0028b9 c610              dec such_len
1341 0028bb 10f3              bpl such2
1342 0028bd a901              lda #1              die null hat verschiebung 1
1343 0028bf 8d0014            sta such_tab
1344 0028c2 68                pla
1345 0028c3 8510              sta such_len
1346                          ;berechne such-ende
1347 0028c5 a5fd              lda botln
1348 0028c7 38                sec
1349 0028c8 e510              sbc such_len
1350 0028ca 852a              sta end
1351 0028cc a5fe              lda botln+1
1352 0028ce e900              sbc #0
1353 0028d0 852b              sta end+1
1354 0028d2 a411      such3   ldy such_lenw       beginne vergleich vom stringe
                                                  nde
1355 0028d4 88                dey
1356 0028d5 203208            jsr such4           suchroutine im common ram
1357 0028d8 60                rts                 zur find-routine
1359                          ;suchstring noch nicht gefunden, nun schieben
1360 0028d9           such5                       ;buchstabe ist index f. such_
                                                  tab
1361 0028d9 8a                txa                 schluesselstelle
1362 0028da 49ff              eor #$ff            komplement
1363 0028dc 38                sec                 2er-komplement
1364 0028dd 790014            adc such_tab,y      verschiebedistanz - schluesse
                                                  lstelle
1365 0028e0 f002              beq such6
1366 0028e2 1002              bpl such7           distanz>0
1367 0028e4 a901      such6   lda #1
1368 0028e6 18        such7   clc
1369 0028e7 65fb              adc nowln
1370 0028e9 85fb              sta nowln
1371 0028eb a5fc              lda nowln+1
1372 0028ed 6900              adc #0
1373 0028ef 85fc              sta nowln+1
1374                          ;teste auf ende
1375 0028f1 c52b              cmp end+1
1376 0028f3 90dd              bcc such3
1377 0028f5 d008              bne such9
1378 0028f7 a5fb              lda nowln           low
1379 0028f9 c52a              cmp end
1380 0028fb 90d5              bcc such3
1381 0028fd f0d3              beq such3
1382 0028ff 18        such9   clc                 nicht gefunden, carry=0
1383 002900 60                rts
```

```
1385                                    ;***** interaktive meldungen
1386 002901 a900      poslin0   lda #0              positioniere auf zeile 0=fens
                                                    ter
1387 002903 85e5                sta sctop         fenste ab zeile nr. 0
1388 002905 a913                lda #home
1389 002907 20d2ff              jsr kbsout
1390 00290a a913                lda #home
1391 00290c 20d2ff              jsr kbsout
1392 00290f a91b                lda #escape
1393 002911 20d2ff              jsr kbsout
1394 002914 a951                lda #'q'          aktuelle zeile loeschen
1395 002916 4cd2ff              jmp kbsout        rts dort
1397 002919 a000      balken    ldy #0            ueberschriftsbalken ausgeben
1398 00291b 200129              jsr poslin0       gehe auf zeile 0
1399 00291e b9522c    balken1   lda mes0,y
1400 002921 f007                beq balken2
1401 002923 20d2ff              jsr kbsout
1402 002926 c8                  iny
1403 002927 4c1e29              jmp balken1
1404 00292a a901      balken2   lda #1
1405 00292c 85e5                sta sctop         normales fenster ab zeile 1
1406 00292e 60                  rts
1408 00292f 207dff    hilfe     jsr kprimm        folgenden text ausgeben
1409 002932 20202020            .byt '    befehlstasten sind: h = diese hilfe',c
                                                    r
1409 002959 0d
1410 00295a 45203d54            .byt 'e =texteingabe tastatur',cr
1410 002971 0d
1411 002972 54203d54            .byt 't =topzeile einstellen',cr
1411 002988 0d
1412 002989 42203d42            .byt 'b =bottom, hinter letzte zeile gehen',cr
1412 0029ad 0d
1413 0029ae 55203d55            .byt 'u =up, 1 zeile aufwaerts gehen',cr
1413 0029cc 0d
1414 0029cd 44204f44            .byt 'd oder cr =down, 1 zl. abwaerts gehen',cr
1414 0029f2 0d
1415 0029f3 53504143            .byt 'space =laufende zeile anzeigen',cr
1415 002a11 0d
1416 002a12 51203d51            .byt 'q =quit, zu basic zurueck',cr
1416 002a2b 0d
1417 002a2c 4b203d4b            .byt 'k =kill, 1 zl. entfernen',cr
1417 002a44 0d
1418 002a45 49203d49            .byt 'i =insert, 1 zl. zwischenschieben',cr
1418 002a66 0d
1419 002a67 3c203d4c            .byt '< =laufender insert mehrerer zeilen',cr
1419 002a8a 0d
1420 002a8b 52203d52            .byt 'r =read v. floppy hinter vorh. text, devic
                                                    e 8',cr
1420 002ab8 0d
1421 002ab9 4f203d52            .byt 'o =read von floppy mitten in den vorh. tex
                                                    t',cr
1421 002ae4 0d
1422 002ae5 4c203d4c            .byt 'l =list, unterbrechbar mit cr',cr
1422 002b02 0d
1423 002b03 46203d46            .byt 'f =find, instring-funktion, stringsuche',c
                                                    r
1423 002b2a 0d
1424 002b2b 57203d57            .byt 'w =weiter suchen, fortsetzung von find',cr
1424 002b51 0d
1425 002b52 43203d43            .byt 'c =change, zeile zum neuen editieren aufma
                                                    chen',cr
1425 002b80 0d
1426 002b81 41203d41            .byt 'a =assembler aufrufen',cr
1426 002b96 0d
1427 002b97 3f203d53            .byt '? =symboltafel ausgeben',cr
1427 002bae 0d
1428 002baf 4d203d53            .byt 'm =sprung zum monitor',cr
1428 002bc4 0d
1429 002bc5 53203d53            .byt 's =scratch all text in this bank',cr
1429 002be5 0d
1430 002be6 50203d50            .byt 'p =print auf device 4',cr
1430 002bfb 0d
```

```
1431 002bfc 23203d41          .byt '# =auf zeile mit nummer positionieren
1432 002c21 26203d53          .byt '& =sequ. file auf floppy schreiben, device
                                    8',cr,0
1432 002c4d 0d00
1433 002c4f 4c3f20            jmp getbef
1435 002c52 12        mes0    .byt revers_on,'befehle: '
1435 002c53 42454645
1436 002c5c 45205420          .byt 'e t b u d "cr" q k i r l f w c a m s n p o
                                    < & h + - ? # CK'
1437 002c97 0d00              .byt cr,null
1438 002c99 12        mes1    .byt revers_on,'texteingabe quit=esc/$'
1438 002c9a 54455854
1439 002cb0 0d00              .byt cr,null
1440 002cb2 0d12      mes2    .byt cr,revers_on,'top',cr,cr,null
1440 002cb4 544f50
1440 002cb7 0d0d00
1441 002cba 12        mes3    .byt revers_on,'end',cr,null
1441 002cbb 454e44
1441 002cbe 0d00
1442 002cc0 0d12      mes4    .byt cr,revers_on,'finden',cr,null
1442 002cc2 46494e44
1442 002cc8 0d00
1443 002cca 12        mes5    .byt revers_on,'veraendern',cr,cr,null
1443 002ccb 56455241
1443 002cd5 0d0d00
1444 002cd8 12        mes6    .byt revers_on,'filename=?',cr,cr,null
1444 002cd9 46494c45
1444 002ce3 0d0d00
1445 002ce6 12        mes7    .byt revers_on,'insert',cr,cr,null
1445 002ce7 494e5345
1445 002ced 0d0d00
1446 002cf0 0d12      mes8    .byt cr,revers_on,'listen',cr,null
1446 002cf2 4c495354
1446 002cf8 0d00
1447 002cfa 0d12      mes9    .byt cr,revers_on,'block zu gross',cr,null
1447 002cfc 424c4f43
1447 002d0a 0d00
1448 002d0c 0d12      mes10   .byt cr,revers_on,'warmstart? j/n',cr,null
1448 002d0e 5741524d
1448 002d1c 0d00
1449 002d1e 12        mes12   .byt revers_on,'alles loeschen? j/cr',00
1449 002d1f 414c4c45
1449 002d33 00
1450 002d34 12        mes13   .byt revers_on,'springe zur Zeile Nr. ',cr,00
1450 002d35 53505249
1450 002d4b 0d00
1451 002d4d 0d12      mes14   .byt cr,revers_on,'speicher voll',cr,00
1451 002d4f 53504549
1451 002d5c 0d00
1453                          ;***** erlaubte befehlstasten
1454 002d5e 45544255 tasten   .byt 'etbud ',cr,'qkirlfwcmaspo<&h+-?',curs_down
                                    ,cursor_up,'#CK'
1454 002d64 0d
1454 002d65 514b4952
1454 002d78 1191
1454 002d7a 23c3cb
1456                          ;***** sprungverteiler fuer befehle
1457 002d7d 5f202521 jmptbl   .wor text_ein-1,top-1,bottom-1,up-1,down-1,retur
                                    n-1
1457 002d81 1f213121
1457 002d85 56216f21
1458 002d89 5621cb21          .wor down-1,quit-1,kill-1
1458 002d8d fa21
1459 002d8f 1522e022          .wor nsert-1,read-1,list-1,find-1,wait_such-1,ch
                                    ange-1
1459 002d93 bd231324
1459 002d97 56246d24
1460 002d9b f624ff2f          .wor monitor-1,assembler-1,loesch-1
1460 002d9f dd21
1461 002da1 9d223c22          .wor print-1,readnsert-1,lin_nsert-1,writ_flop-1
                                    ,hilfe-1
1461 002da5 41224822
1461 002da9 2e29
```

```
1462 002dab 78219121            .wor plus_scr-1,minus_scr-1,taboutj-1
1462 002daf 0230
1463 002db1 56213121            .wor down-1,up-1
1464 002db5 4f230f25            .wor nummer-1,cut-1,copy-1
1464 002db9 5c25
1466 002dbb             assembler =start+$1000        assembler schließt mit luecke
                                                       an
1467 002dbb             taboutj   =start+$1003        ausgabe der symboltafel
1468 002dbb                       .efi
1469 002dbb                       .end

fehlerzahl insgesamt:
0000
```

Roland Löhr

GEMDOS-Routinen

In Heft 46 dieser Zeitschrift wurde das Betriebssystem TOS der Atari-Rechner ST in einem ersten Überblick dargestellt. Es folgte auch ein Programm in Assembler für den seriellen Empfang von Dateien. Hier nun folgt der Quelltext für etwa 40 Funktionen des GEMDOS, mit denen die Ein- und Ausgabe zeichenweise und in Form von Strings abgedeckt wird sowie die Massenspeicherung auf der Floppy. Gegenüber den gleichartigen Routinen in meinem jetzt erscheinenden Buch (siehe Anzeige des te-wi Verlages) wurden Vereinfachungen vorgenommen, für deren Anregung ich Herrn Uwe Kornnagel aus Darmstadt danke. Alle Routinen sind wie bisher als Unterprogramme mit BSR oder JSR aufzurufen, sie führen aber auf eine gemeinsames Ende in den Stummeln GEMDOS2 bis GEMDOS12 am Anfang der Liste. Dazu wird der Befehl BRA benutzt (verzweige immer).

Zur Programmierung ist weiter auszuführen: Der Befehl JSR wurde innerhalb dieser Routinen vermieden, stattdessen wird BSR benutzt (Branch to Subroutine). Das ist zwar meistens kein Muß, weil der Code der Programmteile nie weiter als 32 KB auseinander liegt. Noch ein Hinweis zu Verzweigungsbefehlen: Wie z.B. der 6502, so kennt auch der 68000 "kurze" Verzweigungen von +127/-128 Bytes Distanz. Der Befehl hat dann die Länge eines Wortes. Man kann diese kurzen Verzweigungen bei der Assemblierungen im allgemeinen z.B. mit BRA.S (short) erzwingen. Daneben gibt es die "langen" Verweigungen. Der Befehl hat dann 2 Wort Länge, im 2. Wort steht die Distanz von +-32 KB. Bei der Assemblierung der nachfolgenden Quelltexte wird man an vielen Stellen den Hinweise erhalten: "Warning, Branch could be short", die Verzweigung dürfte auch eine kurze sein. Warnungen dieser Art sind nicht schädlich und man sollte ihretwegen den Quelltext nicht ändern, außer wenn man noch einige Taktzeiten herausschinden muß. Andererseits: Wenn man seine Quelltexte zunächst immer mit BRANCH.S schreibt, dann besorgt man sich unnötig echte Fehler vom Stil "out of range", Verzweigung zu weit. Man sollte also nicht zu geizig mit Zyklen und Platzbedarf sein.

Zur TOS-Programmierung
Aus zahlreichen Veröffentlichungen dürfte dem Leser bekannt sein, daß die neueren Betriebssysteme ihre Leistungen nicht mehr mit Unterprogrammaufrufen vom Stile "JSR absolute Adresse" erbringen, sondern innerhalb eines vorgezeichneten Protokolls der Parameterübergabe mit dem Aufruf einer Funktions-Nummer. Das hat den Vorteil, das aufeinanderfolgende Versionen der Betriebssysteme durch die Beibehaltung des gleichen Protokolls kompatibel bleiben können, obwohl sich absolute Adressen geändert haben. Beim Atari ST ist die Verwirklichung dieses Prinzips an den kompatiblen Versionen des TOS abzulesen, das als Betriebssystem entweder von der Floppy geladen werden kann

oder das in Festwertspeichern (ROMs) mit dem Einschalten des Rechners sofort zur Verfügung steht. Dieses TOS in ROMs (etwa DM 100) ist im übrigen sehr zu empfehlen, weil der Rechner schon nach Sekunden in Betrieb genommen werde kann.
Das Protokoll der Funktionsaufrufe unter TOS lautet wie folgt: Übergib die benötigten Parameter (wenn welche benötigt werden) in einer vorgezeichneten Abfolge als Worte oder Langworte auf dem Datenstack. Übergib dort als letzten Wortparameter die gewünschte Funktionsnummer und rufe dann einen der TRAP-Befehle auf. TOS führt nun die Funktion aus und verändert den zuletzt übergebenen Stackpointer nicht. Es steht jetzt in der Sorgfaltspflicht des Programmierers, den Stackpointer (Stapelzeiger) wieder auf den Stand zurückzubringen, der vor der Beschickung mit den Parametern und der Funktionsnummer galt. - Das besorgen unsere Programm-Enden bei den Labeln GEMDOS2 bis GEMDOS12. - Wenn TOS (bei den meisten Befehlen) ein Ergebnis abliefert, dann findet man es im Register D0 der CPU, also gelesene Zeichen, eine logische Kanal-Nummer (handle) bei Floppy-Befehlen oder einen Hinweis, ob die Operation erfolgreich war oder nicht. Diese Rückgabe wird bei GEMDOS2 ... GEMDOS12 gleich automatisch mit dem Befehl TST.W D0 abgeprüft. Mit Befehlen wie BEQ/BNE bzw. BPL/BMI können wir dann nach dem Funktionsaufruf meistens sofort feststellen, was gelaufen ist, denn der Befehl TST setzt die Flags N und Z im Status. - Bei den meisten Funktionen ist das Datenregister D0 ein zusätzliches Übergabe-Register sowohl im Moment des Aufrufes, als auch nach der Ausführung einer Funktion. Man wird es daher nicht als ein Register benutzen, in dem Daten über die Funktionsaufrufe hinweg aufbewahrt werden sollen.

Folgende TRAP-Befehle kommen zum Einsatz: Bei den vorliegenden GEMDOS-Funktionen TRAP #1, bei den BIOS-Funktionen TRAP #13 und bei den XBIOS-Funktionen TRAP #14 (erweiterte Funktionen für die Bedienung der Hardware des ST). Ein Hantierungshinweis noch: Für das TOS geschriebene Programme sollten wir unter dem Auswahl-Menue EXTRAS für TOS anmelden, wenn wir sie ausführen möchten. Wir erhalten dann einen weißen Bildschirm, einen blinkenden Cursor und schwarze Schrift. Programme, die bereits xyz.TOS als Namen haben, besorgen das automatisch. Liegt keine dieser eingestellten Bedingungen vor, so erhalten wir einen Überschriftsbalken des GEM und einen grauen Hintergrund mit der sichtbaren Biene der Maus. (In einer englischen AMIGA-Veröffentlichung fand ich kürzlich die lustige Frage: Bevorzugen Sie Mäuse gekocht oder roh?)

Zu den Unterprogrammen im einzelnen
Der Autor verweist zunaechst einmal sehr auf die Kommentare in der Liste (Kommentarzeilen beginnen mit einem Stern). Es sind die mit den Funktionen verbundenen Tätigkeiten beschrieben. Hier im Texttteil gehen wir noch auf einige Besonderheiten ein. - Bei den Funktionen wurde folgende Gliederung vorgenommen:

Prüfung der Bereitschaft von Geräten (ergibt einen Status)
Zeichenweise Eingabe, abgelieferte Zeichen in D0
Zeichenweise Ausgabe, Zeichen in D0 übergeben
Eingabe eines Strings, Bufferzeiger in D0.L übergeben
Ausgabe eines Strings, Bufferzeiger in D0.L übergeben
Eigene zusammengesetzte Funktionen wie crlf und writeln
Floppy-Funktionen, siehe weitere Erklärungen

Bei den Floppy-Funktionen wird bei einigem Nachdenken klar, daß sie eigentlich immer die Übergabe mehrerer Parameter erfordern, bzw. mehrere Parameter abliefern, für die man Speicherplatz in Form von Variablen bereithalten muß. Der Name einer Datei muß dabei z.B. in FILENAME stehen (mit 00 abgeschlossener String). In solchen Fällen wird nicht der String selber übergeben, sondern der

Zeiger auf den String, siehe z.B. bei FOPEN1. (Das in Heft 46 veröffentlichte Unterprogramm GETLINE ist geeignet, einen solchen String nach FILENAME zu bringen.) Bei Ein- und Ausgaben sind ferner die Adressen der Buffer und Längenangaben wichtige Parameter (Variablen BUFPTRI, BUFPTRO, BUFLENI, BUFLENO). Wichtig sind auch die Laufwerksnummer CURRDRV (0=A ... 15=P, Funktionen SETDRV und GETDRV u.a.) und der sog. "handle". Bei ihm handelt es sich um eine logische Kanal-Nummer, die man vom System bei der Anlage oder Eröffnung einer Datei zugeteilt erhält. Er wird bei den danach zu benutzenden Funktionen FREAD, FWRITE und FCLOSE zur verkürzten Ansprache der Datei weiterbenutzt.

Bei der Suche und Abfrage von Dateien kommen weitere Parameter ins Spiel: DFREE liefert vier Langworte in den Bereich ab DFREEBUF ab, aus denen man den noch freien Platz auf der Diskette errechnen kann: Zwei Sektoren zu je 512 Bytes bilden dabei eine Allocation Unit (steht im 4. Langwort). Das erste Langwort bei DFREEBUF enthält die Zahl der noch freien und das zweite die Gesamtzahl der Allocation Units. Das dritte die Zahl der Bytes pro Sektor. - Wenn man eine Datei mit Namen und Attribut sucht (Funktionen SFIRST und SNEXT), dann erhält man 44 Bytes Information zurück, die ab DTABUFF abgespeichert werden (siehe den Variablenbereich). Dem System muß allerdings mitgeteilt worden sein, wo der DTA-Buffer liegt (Funktion SETDTA).

Zu den Directory-Funktionen: MKDIR legt einen leeren Ordner an. Die Funktion wirkt damit ebemso, wie die im Desktop mit dem Auswahlmenue betätigte: Anlegen eines Ordners. Wenn man einen solchen Ordner im Desktop zweimal anklickt, dann wird ein Unter-Inhaltsverzeichnis aufgeschlagen, es ist dann das aktuelle. In Assembler erreichen wir das mit der Funktion DSETPATH. Mit DGETPATH stellen wir den hierarchischen Zugangsweg zu einem solchen Unterverzeichnis fest. Man nennt das auch Pfadnamen. Als Text schreibt man z.B. A:\ORDNER1\ . Der Pfadname bedeutet dann: Auf Laufwerk A das Inhaltsverzeichnis innerhalb des Ordners ORDNER1. Der Doppelpunkt und die Backslashes sind dabei notwendige Bestandteile der Formulierung. So gebaute Pfadnamen dürfen auch als FILENAME benutzt werden, so bei den Funktionen FOPEN, CREATE, SFIRST.

Zu Funktionen in einzelnen
CONINSTAT: Es wird der Tastaturpuffer geprüft. D0.W=0 kein Zeichen, D0.W=$FFFF ein Teichen ist verfügbar. Dito bei AUXINSTAT.

CONOUTSTAT, PRINTSTAT, AUXOUTSTAT: Wenn D0.W=$FFFF, dann ist das Gerät bereit, wenn D0.W=00, dann nicht.

Bei den zeichenorientierten Ein- und Ausgaben wird das Byte in den Bits 0...7 des Registers D0 übergeben. Bei CONIN findet man im vorderen Teil des Langwortes in D0 den sog. Scancode, die Nummer der gedrückten Taste. Mit ihr kann man feststellen, welche Taste gedrückt wurde, auch wenn sie als Steuertaste ein ASCII 00 als Zeichen abliefert.

Hinweise zu den Adressierungsarten
Ab Label GEMDOS10 fällt auf, daß die zuvor benutzte Adressierungsart ADDQ.L #x nicht mehr benutzt wird. Diese Adressierung Add Quick akzeptiert Argumente nur bis höchstens 8. Ein guter Assembler macht auf mögliche Fehler aufmerksam. - Um die Adresse einer Variablen zu übergeben, bedienen wir uns des Befehles MOVE.L #Adresse,Rx; Rx ist dabei eines der CPU-Register. Dieser Befehl wird hier häufig benutzt, um einen Zeiger z.B. auf FILENAME zu übergeben. - Demgegenüber würde MOVE.L Adresse,Rx den Inhalt unter Adresse in ein Register bringen.

Der vorliegende Quelltext deckt nicht alle Funktionen des GEMDOS ab, jedoch die wichtigsten. Vom Benutzer ist zu entscheiden, wie er sie möglichst arbeitssparend benutzen will. Er kann a) die Liste vollständig abtippen und

sie als DATEIx abspeichern. Bei ihrer späteren Benutzung in Quelltexten unter Assembler kann er dann einfach sagen: INCLUDE DATEIx. Der Code wird dann etwas umfangreicher, als eigentlich nötig, man braucht sich dann aber um nichts weiter zu kümmern. b) Man kann den Text vom Assembler vor-übersetzen lassen und ihn dem Linker zur Einbeziehung übergeben. c) Man kann die jeweils benötigten Teile per Textsystem in ein neues Quellfile mit Blockbefehlen einkopieren.

```
        GEMDOS-Funktionen als ein Paket von Unterprogrammen
*       Alle Funktionen sind mit jsr oder bsr aufzurufen
*       Ausnahme: jmp term oder bra term, Abschluss eines Prozesses

*       Ergebnisse im allgemeinen im Register D0.L, mit Ausnahmen.
*       Zur evtl. Fehlerpruefung D0.W auswerten

*       Version 10.9.86

 Section one

GEMDOS  equ     1                       Trap-Nummer

*       Routinen zum Abbruch des Prozesses

*       term kehrt zum aufrufenden Prozess zurueck
*       Aufzurufen mit jmp term

term    clr.w   -(sp)                   Funktions-Nummer 00
        trap    #GEMDOS

*       Gemeinsames Ende von Unterprogrammen mit Aufruf des GEMDOS
*       Berichtigung des Stacks entsprechend der Menge der
*       uebergebenen Argumente, Pruefung des Registers D0.W, dann RTS:

gemdos2 trap    #GEMDOS                 Funktionsaufruf
        addq.l  #2,sp                   Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
gemdos4 trap    #GEMDOS                 Funktionsaufruf
        addq.l  #4,sp                   Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
gemdos6 trap    #GEMDOS                 Funktionsaufruf
        addq.l  #6,sp                   Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
gemdos8 trap    #GEMDOS                 Funktionsaufruf
        addq.l  #8,sp                   Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
gemdos10 trap   #GEMDOS                 Funktionsaufruf
        adda.l  #10,sp                  Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
gemdos12 trap   #GEMDOS                 Funktionsaufruf
        adda.l  #12,sp                  Stackberichtigung
        tst.w   d0                      was wird zurueckgeliefert
        rts
```

```
*    Pruefung von Geraeten auf Bereitschaft:

coninstat move.w    #11,-(sp)          Funktionsnummer
          bra       gemdos2

*    auxinstat prueft, ob ein Zeichen am RS232-Eingang vorliegt
*    D0.W wird gesetzt, wie bei coninstat

auxinstat move.w    #18,-(sp)          Funktionsnummer
          bra       gemdos2

*    conoutstat prueft, ob die Standard-Ausgabe (Bildschirm)
*    empfangsbereit ist. Dann ist D0.W=$ff, sonst 00.

conoutstat move.w   #16,-(sp)
          bra       gemdos2

*    printstat prueft, ob die Centronics-Schnittstelle bereit ist,

printstat  move.w   #17,-(sp)
          bra      gemdos2

*    auxoutstat prueft die RS232-Ausgabe, ob bereit

auxoutstat move.w   #19,-(sp)
          bra       gemdos2

* Routinen der zeichenweisen 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 hoeherwertigen Wortes von D0.L ist der Scan-Code

conin     move.w    #1,-(sp)           Funktionsnummer
          bra       gemdos2

*    auxin wartet und holt ein Zeichen von der RS232-Schnittstelle

auxin     move.w    #3,-(sp)           Funktions-Nr.
          bra       gemdos2

*    conio ist eine gemischte E/A-Funktion fuer Tastatur und Bildschirm
*    Mit D0.W<>$ff schreibt man das Zeichen auf den Bildschirm,
*    der Aufruf hierfuer ist conio

conio     move.w    d0,-(sp)           Zeichen in d0
          move.w    #6,-(sp)           Funktions-Nr
          bra       gemdos4

*    Mit D0.W=$ff erhaelt man in D0.L eine 00. wenn keine Taste betaetigt,
*    bei betaetigter Taste holt man ein Zeichen wie bei conin.
conioin   move.w    #$ff,d0            Fuer Zeichenpruefung/Empfang
          move.w    #6,-(sp)
          bra       gemdos4

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

```
dircon    move.w    #7,-(sp)          Funktionsnummer
          bra       gemdos2

*    dirconctrl liest ein Zeichen wie dircon, es werden aber
*    CTRL-Zeichen durchgelassen, die sich am BS auswirken

dirconctrl move.w   #8,-(sp)
          bra       gemdos2

*    Funktionen zur Ausgabe eines einzelnen Zeichens:

*    conaut gibt ein in D0.W, Low Byte uebergebenes 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.
          bra       gemdos4

*    auxout gibt gibt ein Zeichen in D0.W auf den RS232-Kanal aus,
*    Bedingungen sonst wie bei conout

auxout    move.w    d0,-(sp)
          move.w    #4,-(sp)
          bra       gemdos4
*
*    prtout gibt ein Zeichen in D0.W auf den Printerport (Centronics) aus,
*    Bedingungen sonst wie bei conout

prtout    move.w    d0,-(sp)
          move.w    #5,-(sp)
          bra       gemdos4

* Funktionen zur Ein- und Ausgabe von Strings

*    readline: Lesen einer Zeile vom Keyboard mit Echo auf BS, bis CR
*    auftritt. Maximale Stringlaenge im 1. Byte des Buffers vorgeben,
*    readline liefert die erreichte Laenge im 2. Byte
*    und den empfangenen String ab 3. Byte des Buffers ab
*    Steuerzeichen koennen stoerend wirken

readline  move.l    d0,-(sp)          Zeiger auf den Eingabe-Buffer in d0
          move.w    #10,-(sp)         Funktions-Nr
          bra       gemdos6

*    printline gibt einen String Byte fuer Byte auf die Standard-Ausgabe
*    aus, bis eine 00 als Begrenzer gefunden wird.
*    In D0.L ist die Adresse des Stringbeginnes zu uebergeben

printline move.l    d0,-(sp)          Stringadresse
          move.w    #9,-(sp)     Funktion
          bra       gemdos6

*    crlf gibt ein CR und LF auf die Standard-Ausgabe (Bildschirm)

crlf      move.w    #$0d,d0           Das CR
          bsr       conout
          move.w    #$0a,d0           Das LF
          bra       conout            rts dort
```

```
*    writeln schreibt einen mit 00 begrenzten String auf die
*    Standard-Ausgabe und sendet ein cr/lf hinterher

writeln   bsr       printline          String-Ausgabe
          jmp       crlf               Zeilenabschluss, rts dort

* GEMDOS-Funktionen fuer Floppy

*    Einen Drive zum aktuellen machen (0=A, 1=B ... 15=P):
*    Drive-# muss als Hex-Zeichen in D0.W sein

setdrv    move.w    d0,-(sp)       Parameteruebergabe
          move.w    #$0e,-(sp)     Funktionsnummer 14
          bsr       gemdos4
          move.l    d0,drivemap    Bitmap der Drives kommt zurueck
          rts

*    Abfrage, welcher Drive der aktuelle ist (0=A usw.):

getdrv    move.w    #$19,-(sp)     Hole aktuellen Drive, Funktions-#
          bsr       gemdos2
          move.w    d0,currdrv     Current Drive merken
          rts

*    Definiere den DTA-Buffer fuer File-Parameter, der von der
*    Funktion SFIRST=$4E gefuellt wird, wenn etwas gefunden wurde
*    (DTA=Disk Transfer Address)

setdta    move.l    #dtabuff,-(sp) Adresse 44 Byte DTA-Buffer uebergeben
setdta1   move.w    #$1a,-(sp)     Funktions-#
          bra       gemdos6

*    Hole Adresse des aktuellen DTA-Buffers

getdta    move.w    #$2f,-(sp)     Funktionsnummer
          bsr       gemdos2
          move.l    d0,currdta     Adresse aktueller DTA merken
          rts

*    Hole Drive-Parameter, Berechnung der noch freien Bytes

dfree     move.w    #0,-(sp)       Freien Speicher Current Drive bestimmen
          move.l    #dfreebuf,-(sp) Adresse, wo die Information
          move.w    #$36,-(sp)     abzulegen ist, Funktions-#
          bra       gemdos8

*    Lege neues File mit Namen und Attribut an
*    Loesche ggfs. vorhandenes File gleichen Namens

createrw  move.w    #0,-(sp)       Setze Attribut Lesen/Schreiben
          bra.s     createl        Verzweige immer, kurz
createrd  move.w    #1,-(sp)       Setze File-Attribut nur Lesen
createl   move.l    #filename,-(sp) Zeiger auf zu verwendenden Dateinamen
          move.w    #$3c,-(sp)     Funktionsnummer
          bsr       gemdos8
          move.w    d0,handle      Kanal-# merken
          rts                      Fehler, wenn negativ
```

```
*    Lege einen neuen Ordner/Sub-Directory an (Pfadnamen schaffen)

mkdir     move.l    #filename,-(sp) Zeiger auf Namen des Ordners
          move.w    #$39,-(sp)      Funktions-Nr.
          bra       gemdos6

*    SETATTRIB setzt ein File-Attribut,
*    das beim Aufruf in D0.W enthalten sein muss

setattrib move.w    d0,-(sp)        Attribut aus D0
          move.w    #1,-(sp)        Funktion: Setzen
setattrl  move.l    #filename,-(sp) Zeiger auf den Pfadnamen
          move.w    #$43,-(sp)      Funktions-#
          bra       gemdos10

*    GETATTRIB holt das Attribut einer Datei mit Pfadnamen

getattrib move.w    #0,-(sp)        Parameter unbedeutend
          move.w    #0,-(sp)        Funktion: Holen
          bra.s     setattrl        Attribut am Ende in D0 oder negativ

*    Oeffne vorhandene Datei mit Namen und Attribut

fopenwrit move.w    #1,-(sp)        Oeffne zum Schreiben einer Datei
          bra.s     fopen1
fopenrw   move.w    #2,-(sp)        Oeffne zum Lesen und Schreiben
          bra.s     fopen1
fopenread move.w    #0,-(sp)        Oeffne zum Lesen
fopen1    move.l    #filename,-(sp) Zeiger auf Dateinamen
          move.w    #$3d,-(sp)      Funktions-# FOPEN
          bsr       gemdos8
          move.w    d0,handle       Kanal-# merken
          rts

*    Schliesse Datei mit handle-#

fclose    move.w    handle,-(sp)    Kanal nennen
          move.w    #$3e,-(sp)      Funktionsnummer
          bra       gemdos4

*    Einlesen von einer Datei mit handle-# ab Adresse bufptri
*    Die zu lesende Menge muss in BUFLENI uebergeben 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
          bra       gemdos12

*    Schreiben auf eine Datei mit handle-# ab Adresse bufptro
*    Die zu schreibende Menge muss in BUFLENO uebergeben 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.
          bra       gemdos12

*    Pruefen, ob eine Datei lt. FILENAME vorhanden ist
```

```
sfirst    clr.w     sattrib        Normale Files suchen
*    sfirstl = Einsprung fuer anders vorbesetztes sattrib
sfirstl   bsr       setdta         Setze Disk Transfer Adresse
          move.w    sattrib,-(sp)  Attribut des Files
          move.l    #filename,-(sp) Suchname
          move.w    #$4e,-(sp)     Funktions-#
          bra       gemdos8

*     Folge-File zu sfirst suchen, gleiche Namensbestandteile

snext     move.w    #$4f,-(sp)     Funktionsnummer
          bra       gemdos2

*    FSEEK positioniert innerhalb einer Datei
*     in D0.L muss +- der Offset uebergeben 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-#
          bra       gemdos10

*    Pathname zum aktuellen Inhaltsverzeichnis machen

dsetpath  move.l    #filename,-(sp) Zeiger auf den String mit Pathname
          move.w    #$3b,-(sp)     Funktions-#
          bra       gemdos6

*    Vollstaendigen Pfadnamen erfassen in PATHBUFF

dgetpath  move.w    #0,-(sp)       Default Aufruf, aktueller Drive
          move.l    #pathbuff,-(sp) Zeiger auf Empfangsbuffer
          move.w    #$47,-(sp)     Funktions-#
          bra       gemdos8

*    GETNSTDH holt vom GEMDOS einen Non-Standard-Handle
*    fuer eine System-Device (mit Standard-Handle)
*    Der Standard-Handle muss beim Aufruf in D0.W stehen

getnstdh  move.w    d0,-(sp)       Standard-Handle aus D0
          move.w    #$45,-(sp)     Funktions-#
          bsr       gemdos4
          move.w    d0,nstdhdl     Non-Standard-Handle merken
          rts

*    SETNSTDH laesst einen Standard-Handle auf dasselbe File
*    oder auf dieselbe Device zeigen,
*    wie den uebergebenen Non-Standard-Handle
*    Der Standard-Handle muss in D0.W uebergeben werden
*    Der Non-Standard-Handle wird NSTDHDL entnommen

setnsdth  move.w    d0,-(sp)       Standard-Handle in D0
          move.w    nstdhdl,-(sp)  Zuweisung Non-Standard-Handle
          move.w    #$46,-(sp)     Funktions-#
          bra       gemdos6
```

```
*    Arbeitsspeicher

dfreebuf  ds.l      1           Information Return Buffer/DFREE
totalun   ds.l      1           # Total Allocation Units
bytspsec  ds.l      1           # Bytes per Sector
physsec   ds.l      1           # Physical Sectors/Drive Allocation Unit

bufptri   ds.l      1           Zeiger auf aktuellen Eingabepuffer
bufptro   ds.l      1           Zeiger auf aktuellen Ausgabepuffer
bufleni   ds.l      1           Enthaelt Menge zu lesender Bytes
bufleno   ds.l      1           Enthaelt Menge zu schreibender Bytes
drivemap  ds.l      1           Bitmap der Drives im System
handle    ds.w      1           # des Kanals
nstdhdl   ds.w      1           Non-Standard-Handle
currdrv   ds.w      1           # Current Drive
currdta   ds.l      1           Adresse der aktuellen DTA

sattrib   ds.w      1           File-Attribute

* dtabuff enthaelt 44 Bytes mit File-Angaben
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           Dateigroesse, Langwort
fnamfound ds.b      14          Gefundener Filename + Extension, Ende DTA

pathbuff  ds.b      64          Pathname mit dgetpath erfasst
filename  ds.b      80          Filename-Arbeitsspeicher
```

□□

Roland Löhr

68000 Filter

Für die tägliche Assembler-Programmierung des 68000 sollen Quelltexte und Anregungen gegeben werden, wie man ein im Register D0.B übergebenes Zeichen auf so oft benötigte Merkmale wie Buchstabe, Zahlzeichen usw. prüfen kann. Im ersten Abschnitt beschreiben wir die konventionelle Methode, die mit dem Befehl CMPI.B # Durchlaßfilter bildet. Der zweite Teil beschreibt eine einfachere universale Methode, bei der für jedes Zeichen auf eine binäre Strichliste mit zutreffenden Merkmalen Rückgriff genommen wird. Sie wird als Tabelle geführt.

Bei der Programmierung halten wir uns vor Augen, daß der Vergleichsbefehl CMP bei Motorola den Status genau umgekehrt setzt, wie der 6502: Wenn geborgt wurde, weil der Zieloperand (bei uns im Register D0.B) kleiner als der Vergleichsoperand ist, dann wird das Carry-Bit=1, im anderen Falle =0.

Die mitgeteilten Filter stellen nach dieser Einleitung für geltende ASCII-Codierung also keine logische Besonderheit dar. Einzige Ausnahme sind die deutschen Sonderzeichen. Sie sind von Computer zu Computer anders verschlüsselt. Beim Atari sind sie nicht vom Typ DIN-ASCII, sondern haben Werte, die größer als hexadezimal $80 sind. Wir zählen auch diese Zeichen zu den Alpha-Zeichen (die Untersuchung findet im Unterprogramm DEUTSCH statt). Es wird Anwendungen geben, wo man anders verfahren möchte.

```
* Eine Reihe von Filter-Routinen
* Es wird vorausgesetzt, dass das zu pruefende Zeichen
* in D0.B als Byte enthalten ist

* Wenn die zu pruefende Bedingung zutrifft, wird
* die Variable FILTFLAG zu 00 gesetzt, sonst zu $FF
* Das aufrufende Programm kann mit BPL/BMI oder
* BEQ/BNE sofort weiter verzweigen

* HEXFILT  zeigt in filtflag an, ob ein Zeichen in Gross-
* oder Kleinschreibung eine Hexziffer ist

hexfilt   cmpi.b   #'g',d0        Kleines g und folgende Zeichen
          bcc.s    nothex         sind nicht hex
          cmpi.b   #'a',d0        a ... f
          bcc.s    ishex          sind zugelassen
          cmpi.b   #'G',d0        Grosses G ...
          bcc.s    nothex         sind nicht hex
          cmpi.b   #'A',d0        A ... F, Grossbuchstaben
          bcc.s    ishex          wieder zugelassen
          cmpi.b   #':',d0        Dazwischen liegen wieder keine
          bcc.s    nothex         Hexziffern
          cmpi.b   #'0',d0        Ziffernbereich 0 ... 9 erlaubt
          bcs.s    nothex         Zeichen liegt darunter
ishex     clr.b    filtflag       =00
          rts
nothex    move.b   #$ff,filtflag  Flag wird negativ
          rts

* ALPHA ist ein Filter fuer Buchstaben A...Z und a...z
* Deutsche Sonderzeichen werden als Buchstaben angesehen

alpha     jsr      deutsch        filtere deutsche Zeichen
          bpl.s    isalpha        Sonderzeichen erkannt
          cmpi.b   #$7b,d0        Groesser 'z'?
          bcc.s    notalpha       ja
          cmpi.b   #'a',d0        Buchstaben 'a'...'z'?
          bcc.s    isalpha        ja
          cmpi.b   #$5b,d0        Groesser 'Z'?
          bcc.s    notalpha       ja, Sonderzeichen
          cmpi.b   #'A',d0        Buchstaben 'A'...'Z'?
          bcs.s    notalpha       nein, kleiner
          clr.b    filtflag
isalpha   rts                     FILTFLAG=0, Zeichen ist alpha
notalpha  move.b   #$ff,filtflag
          rts                     FILTFLAG = negativ

* ALPHNUM ist ein Filter fuer Ziffern 0...9 und Buchstaben wie bei ALPHA

alphnum   cmpi.b   #$3a,d0        Groesser '9'?
          bcs.s    alpha          ja, keine Ziffer
          cmpi.b   #'0',d0        Die '0' gehoert dazu
          bcs.s    notalpha       ASCII-Wert ist geringer
          clr.b    filtflag
          rts                     Filtflag=0, Zeichen ist alphanumerisch
```

```
* DEZZIF ist ein Filter fuer Dezimalziffern 0...9

dezzif    cmpi.b    #$3a,d0        Groesser als '9'?
          bcc.s     notalpha       ja
dezzif1   cmpi.b    #'0',d0        Unter '0'?
          bcs.s     notalpha       ja, FILTFLAG=negativ
          clr.b     filtflag       FILTFLAG=0
          rts

* OCTZIF erkennt auf Oktalziffern 0...7

octzif    cmpi.b    #'8',d0        8 und groesser
          bcc.s     notalpha       ja, scheidet aus
          bra.s     dezzif1

* BINZIF erkennt auf Binarziffern 0...1

binzif    cmpi.b    #'2',d0        2 und groesser
          bcc.s     notalpha       ja, scheidet aus
          bra.s     dezzif1

* DRUCKBAR ist ein Filter fuer die Zeichen Space...Tilde ($7e)

druckbar  jsr       deutsch        Filtere deutsche Sonderzeichen
          beq.s     druckbz        erkannt
druckb1   cmpi.b    #$7f,d0        Rubout oder groesser?
          bcc.s     notalpha       ja, nicht druckbar
          cmpi.b    #' ',d0        Space?
          bcs.s     notalpha       kleiner, nicht druckbar
          clr.b     filtflag       =00
druckbz   rts

* Erkennung deutscher Sonderzeichen

deutsch   movem.l   a1/d1,-(sp)    Register retten
          lea.l     sonderz,a1
          move.w    #6,d1          7 Zeichen
deutschl  cmp.b     (a1)+,d0       Vergleich
          dbeq      d1,deutschl
          tst.w     d1
          bmi.s     deutschn       nichts gefunden
          clr.b     filtflag
deutschz  movem.l   (sp)+,a1/d1    Register zurueck
          tst.b     filtflag
          rts
deutschn  move.b    #$ff,filtflag
          bra.s     deutschz

* Tastaturcodes Atari 520 fuer deutsche Sonderzeichen
sonderz   dc.b      $84,$94,$81,$9e,$8e,$99,$9a

filtflag  ds.b      1
```

Klassifizierung mit Tabellen
Die nachfolgende Methode zur Erkennung von Merkmalen, die mit Eingabezeichen verbunden oder assoziert werden können, wurde hier vor allem für die schnelle syntaktische Zerlegung eines in Assemblerspache formulierten Eingabestromes entwickelt. Da gibt es Regeln etwa wie folgt: Ein Symbol muß mit einem

Alpha-Zeichenbeginnen. Man kann sie wie folgt erweitern: Es darf auch mit einem Unterstreichungsstrich oder mit einem deutschen Sonderzeichen beginnen. Zeichen, die am Beginn einer Zeile stehen dürfen, sind in der nachfolgenden Tabelle in Bit 13 mit einer "1" gekennzeichnet. Wie wir noch sehen werden, ist dieses Tabelle auf den Befehlssatz und die Adressierungsarten des 68000 besonders zugeschnitten.

Es gibt dann Zeichen, die ab zweiter oder höherer Stelle in einem String stehen dürfen. Das sind Buchstaben und Ziffern (alphanumerische Zeichen). Bei Symbolen seiner Assembler läßt der Autor hier zusätzlich das Fragezeichen zu. Dann gibt es das Merkmal "druckbar" schlechthin. Dazu gehört dann auch der Space.

Es gibt das Merkmal Steuerzeichen (in Bit 8 eine "1" für die ASCII-Codes $01...$1f). Steuerzeichen, auper CR und ggfs. LF gehören nicht in den Eingabestrom für Assembler-Sprache.

Sonderzeichen allgemein (Bit 10="1") umfassen die Zeichen der Interpunktion, Klammern, Dollar Anführungsstriche usw. Man kann sie in weitere Gruppen aufteilen: a) eine Zahlenbasis kennzeichnend durch $, % oder (Bit3="1"), b) einen Operator der Grundrechenarten (Bit 2="1"), c) eine Zuweisung oder einen Vergleich begleitend (Bit 0="1"), d) runde Klammern (Bit 8="1") oder die Adressierungsart kennzeichnend (Bit 9="1").

Je nach Zahlensystem sind alphanumerische Zeichen groß oder klein geschrieben entweder reine Dezimalziffern 0...9 (Bit7="1"), als Hexzeichen zugelassen (Bit 6="1"), Oktalziffern (Bit 5="1") oder Binärziffern (Bit 4="1").

Aus Gründen der Aufschlüsselung oder Übersetzung werden hier außerdem die Merkmale "deutsches Sonderzeichen" (in Bit 12) und "Underline oder Fragezeichen" (in Bit 11) geführt. .

Aus dem Gesagten geht hervor, daß für jedes Zeichen ein Merkmalswort mit 16 Bit Information angelegt wird, und zwar in der Tabelle ZCHNTAB (s.u.). Das mag auf den ersten Blick als zu aufwendig und redundant aussehen. Der in der nachfolgenden Liste enthaltene Zerlegungscode möge den Leser jedoch überzeugen, daß man nicht nur beim 68000 mit einer solchen für den jeweiligen Zweck angelegten Tabelle mit wenigen Programzeilen sehr schnell zu Entscheidungen kommt. - Bei der am Beginn dieses Artikels zunächst angesprochenen Filter-Methode mit aufeinanderfolgenden Vergleichen muß man, wenn zunächst einmal die grobe Gruppenzugehörigkeit erkannt wurde (z.B. alphanumerisch) im allgemeinen noch viele Einzelabfragen bis zur Entscheidung folgen lassen, die bei der tabellarischen Methode zumeist entfallen können. Und die konventionelle Methode der Filterung hat den Nachteil, daß sie mit zahlreichen Aufrufen von Unterprogrammen arbeitet, was relativ zeitaufwendig ist.

Viele Zeichen sind mehrdeutig und erst aus dem Zusammenhang zuzuordnen. Der Buchstabe "A" kann z.B. Bestandteil eines Bezeichners sein aber auch eine hexadezimale Ziffer. In einer binären Strichliste wird er daher an mehreren Positionen eingetragen sein (auch: druckbar, darf am Zeilenanfang stehen usw.). Es kommt also nicht darauf an, daß es im Wort oder Langwort des Merkmales (ZCHNTAB) immer nur einen Strich geben darf. Es kommt vielmehr auf die hierarchische Abfragestruktur an, die die Merkmale stufenweise eingrenzt. So schreibt man Ausdrücke für Zahlen in Assembler z.B. $AAFF. Nach dem Dollar werden also hexadezimale Ziffern erwartet. Ihre Bewertung ist abzubrechen, wenn plötzlich ein Fremdzeichen auftaucht, das nicht mehr das Merkmalsbit "hexadezimal" hat, wie z.B. in $AAFF, oder in $AAFFG. Das fremde Zeichen führt zu einer anderen Logik der Weiterbehandlung.

Man kann auch die Merkmale eines Strings zusammenODERn. Wenn man dabei feststellt, daß deutsche Sonderzeichen, Underline oder ein Fragezeichen enthalten waren, so kann man sagen: Das ist bestimmt kein mnemonischer Befehl. Je nach Assemblersprache kann man weiter sagen: Wenn eine Ziffer im String auftaucht, dann kann es auch kein mnemonischer Befehl sein. So beim 68000, nicht jedoch, wenn man Mnemonics wie SMB5 zuläßt (setze Memory Bit 5 an Adresse). Entsprechende Prüfungen werden in der folgenden Liste zwischen den Labeln KN3 und KN4 durchgeführt.

Die Filterung mit der Tabelle beruht auf folgender Programmiertechnik: Beim 68000 gibt es die hier benutzte Adressierungsart "Adreßregister indirekt mit Index" in den Schreibweisen (An,Rn) und d(An,Rn). An ist dabei ein Adreßregister, Rn irgendein Adreß- oder Datenregister als Index und d ein möglicher Offset dazu, von dem wir hier keinen Gebrauch machen. Wenn wir nun das zu prüfende Byte ins Register D0.W geladen haben und wenn durch die Vorbereitung seine Bits 8...15 Null sind, dann kann sich das ASCII-Byte selbst in eine Tabelle indizieren. Das geschieht hier in den ersten Zeilen nach PRUEFUNG: Es wird eine Kopie des Bytes in D7 angefertigt und mit ASL.W #1,D7 mit zwei multipliziert, weil wir eine Tabelle mit Merkmals-Worten adressieren wollen. Das Register A6 zeigt fest auf die Basis der Tabelle ZCHNTAB. Beim Label PR2 wird zu dieser Basis der in D7 berechnete Offset als Index hinzuaddiert. Das mit dieser effektiven Adresse geladene Wort wird nach D7 gebracht. Also: D7 wirkt an dieser Stelle als Index für die Quelladresse und zugleich auch als Ziel der Datenbewegung.

Im Anschluß an dieses Laden des Merkmalwortes ins Register D7 können nun die Befehle des 68000 zur Bitprüfung bequem benutzt werden, hier in der Form BTST #n,D7; n ist dabei ein Direktoperand, der eine Bit-Nummer 0...15 (bei Langwortprüfung auch bis 31 zulässig) enthält. Bei KN2 wissen wir spontan, ob das Zeichen am Beginn einer Zeile stehen darf, bei KN3, ob es ein Sonderzeichen ist, das am Zeilenbeginn stehen darf. Zwischen KN3 und KN7 haben wir mit wenigen weiteren Befehlen festgestellt, ob eine eingelesene Textkette mit Sicherheit ein Symbol ist (weil die Zeichen Underline, Fragezeichen oder Dezimalziffern enthalten sind) oder ob wir zunächst die Tabelle der mnemonischen Befehle abgrasen müssen.

Die logische Zerlegung am Zeilenbeginn umfaßt für diese Umgebung nur etwas mehr als zwanzig Befehlszeilen. In der Summe werden wohl auch weniger Befehlszeilen durchlaufen, als beim Aufruf von Unterprogrammen zur konventionellen Art der Filterung. Und es entfallen die Zeitverluste durch JSR und RTS. - Es steht im Belieben des Anwenders, für seine Zwecke andere und auch umfangreichere Tabellen für die logische Aufschlüsselung von Merkmalen anzulegen.

```
* Filterung mit Merkmals-Tabelle
* Rev. 15.7.86 by Roland Loehr

* Am Beginn dieser Programm-Sequenz sei das zu pruefende Zeichen
* bereits im Datenregister D0.B geladen, Bits 8...15 seien 0
* Benoetigte Register seien schon gesichert

pruefung  lea.l    zchntab,a6      Zeige auf die Tabelle der Merkmale
          clr.w    d7              setze Register zu 00
pr1       move.b   d0,d7           ASCII-Zeichen uebertragen
          asl.w    #1,d7           *2 = Vektor in Wort-Tabelle
pr2       move.w   (a6,d7.w),d7    Merkmal holen und Status

kn1       beq      nextline        leere Zeile, naechste Zeile holen
```

```
weg2    btst    #13,d7          als 1. Zeichen der Zeile zulaessig?
kn2     beq     fehler          nein
        btst    #10,d7          Test auf Sonderzeichen
kn3     bne     weg4            ja, trifft zu
        clr.w   d6              Register leeren
        or.w    d7,d6           Merkmal odern
        jsr     lieskett        Lies Rest vom String und odere Zeichen
*                               fuer Zeichen das Merkmal in D7 nach D6
*                               als Sammler der Merkmale

        btst    #11,d6          Underline/Fragezeichen enthalten?
kn4     beq     weg6            nein, dann Mnemonic oder Symbol
weg5    bra     symbol          ja, dann kann es nur Symbol sein

* String ist alphnum ohne _ und ?, ggfs. numerische Zeichen enthaltend,
* also Mnemonic oder Symbol

weg6    tst.b   d6              Pruefung auf numerisch
kn5     bmi.s   symbol          ja, darf nicht in Mnemonic sein
        cmpa.w  #3,a4           Laenge unter 3?
kn6     bcs.s   symbol          ja, kann nicht Mnemonic sein
        cmpa.w  #6,a4           oder 6 und mehr?
kn7     bcc.s   symbol          dito

* Jetzt Laenge 3 und 4 uebrig. Kann Mnemonic oder Symbol sein

weg7    bra mnempars            Mnemonics parsen!

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

* weg4 ist Sammelpunkt fuer Sonderzeichen,
* die am Zeilenanfang stehen duerfen naemlich / ; . # *

weg4    cmpi.b  #'/',d0         Beginn Kommentarblock?
kn8     beq     kblock          ja, kann sein
        cmpi.b  #';',d0         Kommentarzeile?
kn9     beq     commline        ja, definitiv
        cmpi.b  #'.',d0         Direktive mit Punkt?
kn10    beq     dirpars         ja, Tabelle absuchen
        cmpi.b  #'#',d0         INCLUDE?
        beq     includen         ja

stern   jsr     nextchar        Folgezeichen untersuchen
      usw.

*************** Ende des Parsens am Zeilenanfang *************

*************** Bereich der Konstanten ***********************

* Tabelle zum direkten Umsetzen von ASCII- und sonstigen
* Zeichen im Merkmale, die ein Parsen beschleunigen
*
* Je Zeichen wird ein Merkmalswort benoetigt, dessen gesetzte Bits
* folgende Bedeutung haben:
* Bit-Nr.
* 15 alphanumerisch im weitesten Sinne, inkl. Underline, Fragezeichen
*    und deutschen Sonderzeichen. Haufigste Abfrage beim Suchen von Symbolen
* 14 druckbares Zeichen, einschl. Space, dt. Sonderzeichen
* 13 reines alpha-Zeichen, inkl. dt. Sonderzeichen
```

```
* 12 deutsches Sonderzeichen
* 11 Underline oder Fragezeichen
* 10 Sonderzeichen allgemein
*  9 Punktzeichen .,;: wenn druckbares Zeichen
*    Grafikzeichen, wenn Bit 14=0
*  8 Klammerzeichen, wenn Bit 14=1. Runde, eckige Klammern, Akkoladen
*    Steuerzeichen $01..$1f, wenn Bit 14=0
*  7 Dezimalziffer 0...9, numerisches Zeichen
*  6 Hexziffer 0...9, A...F, a...f
*  5 Oktalziffer 0...7
*  4 Binarziffer 0...1
*  3 Zahlzeichen $, % oder Klammeraffe @
*  2 arithmetischer Operator + - * /
*  1 Logischer Operator & ! ^ oder
*  0 Zuweisungs- oder Vergleichsoperator = < >

zchntab ;Tabelle der Merkmale von Zeichen:
          dc.w      $0000,$0100,$0100,$0100  Zeilenende, Steuerzeichen
          dc.w      $0100,$0100,$0100,$0100
          dc.w      $0100,$0100,$0100,$0100
          dc.w      $0100,$0100,$0100,$0100

          dc.w      $0100,$0100,$0100,$0100  noch Steuerzeichen
          dc.w      $0100,$0100,$0100,$0100  $10-$1f
          dc.w      $0100,$0100,$0100,$0100
          dc.w      $0100,$0100,$0100,$0100

          dc.w      $4000,$4402,$4400,$6600  ab Space
          dc.w      $4408,$4408,$4402,$4400  bis Hochkomma
          dc.w      $4500,$4500,$6404,$4404  ab runde Klammer auf
          dc.w      $4600,$4404,$6400,$6404  bis Slash

          dc.w      $c0f0,$c0f0,$c0e0,$c0e0  Ziffern ab Null
          dc.w      $c0e0,$c0e0,$c0e0,$c0e0  bis 7,
          dc.w      $c0c0,$c0c0              bis 9
          dc.w      $4400,$6400,$4601,$4401  Doppelpunkt
          dc.w      $4601,$c800              bis Fragezeichen

          dc.w      $4408,$e040,$e040,$e040  Klammeraffe
          dc.w      $e040,$e040,$e040        bis F
          dc.w      $e000,$e000,$e000,$e000  G
          dc.w      $e000,$e000,$e000,$e000
          dc.w      $e000                    bis O

          dc.w      $e000,$e000,$e000,$e000  P
          dc.w      $e000,$e000,$e000,$e000
          dc.w      $e000,$e000,$e000        bis Z
          dc.w      $4400,$4400,$4400,$4402  eckige Klammer auf
          dc.w      $e800                    bis Underline

          dc.w      $4400,$e040,$e040,$e040  Akzent
          dc.w      $e040,$e040,$e040        bis f
          dc.w      $e000,$e000,$e000,$e000  kleines g
          dc.w      $e000,$e000,$e000,$e000
          dc.w      $e000                    bis kleines o

          dc.w      $e000,$e000,$e000,$e000  kleines p
          dc.w      $e000,$e000,$e000,$e000
          dc.w      $e000,$e000,$e000        bis kleines z
```

```
         dc.w     $4400,$4402,$4400,$4400  Akkolade auf
         dc.w     $0100                    bis Rubout
* Zeichen groesser gleich hex 80:
         dc.w     $0200,$f000,$0200,$0200  zumeist Grafik, sonst
         dc.w     $f000,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$0200,$0200,$0200  $80-8f
         dc.w     $0200,$0200,$f000,$0200

         dc.w     $0200,$0200,$0200,$0200  zumeist Grafik, sonst
         dc.w     $f000,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$f000,$f000,$0200  $90-9f
         dc.w     $0200,$0200,$f000,$0200


         dc.w     $0200,$0200,$0200,$0200  zumeist Grafik, sonst
         dc.w     $0200,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$0200,$0200,$0200  $a0-af
         dc.w     $0200,$0200,$0200,$0200

         dc.w     $0200,$0200,$0200,$0200  zumeist Grafik, sonst
         dc.w     $0200,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$0200,$0200,$0200  $b0-bf
         dc.w     $0200,$0200,$0200,$0200

         dc.w     $0200,$0200,$0200,$0200  zumeist Grafik, sonst
         dc.w     $0200,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$0200,$0200,$0200  $c0-cf
         dc.w     $0200,$0200,$0200,$0200

         dc.w     $0200,$0200,$0200,$0200  zumeist Grafik, sonst
         dc.w     $0200,$0200,$0200,$0200  deutsche Sonderzeichen
         dc.w     $0200,$0200,$0200,$0200  $d0-df
         dc.w     $0200,$4408,$0200,$0200  $dd=Paragraph
```

□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O

Roland Löhr

CBM-Eprommer

Heft 45 enthielt unter dem Titel 'Eproms ab Typ 2764' eine Schaltung nebst Betriebsprogramm für den AIM 65 zum Brennen und Prüfen von EPROMS ab Typen 2764. Mit der nachstehenden sehr einfachen Schaltung und mit dem Betriebsprogramm kann man nunmehr auch die Commodore-Rechner 610...720 als Eprommer benutzen, und zwar mit erhöhtem Komfort. - Geräte dieses Typs werden bekanntlich z.Zt. sehr preiswert (ab DM 218) als Ausverkaufs-Posten durch die Fa. Völkner angeboten und inzwischen für viele ernsthafte Anwendungen benutzt. In ihrer Leistungsfähigkeit sind sie durchaus mit einem einfachen PC vergleichbar. - Das Betriebsprogramm dürfte sich auch auf andere Commodore-Rechner übertragen lassen (z.B. PC-128), weil es dort eine gleichmäßige Sprungleiste zu den Hauptroutinen gibt. Man wird allerdings Abwandlungen in Betracht ziehen müssen, wenn die Zahl der Pins am Userport nicht ausreicht.

Es ist folgende Umgebung in der Hardware zu beachten: Die Hilfsspannungen +6 und +21 Volt sollten einem Labornetzgerät entnommen und durch Schalter abschaltbar gemacht werden. Zur Platine des CBM 710 wurde von hier aus bereits früher berichtet, daß die Pins 1 und 3 am Userport keine Verbindung zu Masse

haben. Man muß sie nachlöten. Weiterhin ist das Programm für die Systembank 15 geschrieben. Es setzt damit eine Erweiterungsplatine am Cartridge-Port voraus (siehe Heft 46). Bei Verwendung der Banking-Routinen (Heft 33) wird man das Programm aber auch in einer anderen Bank lauffähig machen können. - Bei der Eprobung von Programm und Schaltung stellte sich am Oszilloskop heraus, daß der mit angeblich 2 MHz betriebene CBM in Wirklichkeit einen Quarz mit 1,8432 MHz hat, also etwas langsamer ist. Das war dann bei der Programmierung der Timer für die Dauer der Impulse zu berücksichtigen.

Beim Aufbau der Schaltung war folgendes zu bedenken: Der Userport des CBM hat weniger als 26 freie Pins. Aus diesem Grund sind zum EPROM-Sockel führende Signale zu multiplexen. Dazu wird hier ein 12-stufiger asynchroner Binärzähler vom Typ CMOS 4040 benutzt. Er legt die Adreßsignale A0...A11 an den Sockel. Vor einer Programmierung ist der Stand des Zählers auf eine definierte Anfangsadresse zu bringen. Das geschieht mit der Routine PULS UP in Zeile 612. Vom Anfangsstand aus wird er später um jeweils 1 in PULS PLUS hochgezählt. Der Zählimpuls wird dabei am Portpin PB6 des CIA 6526 erzeugt. PB5 beherrscht den Reset-Eingang des Zählers.

Die Adreß-Signale A12 und A13 werden auf die Portpins PB0 und PB1 dieses CIA gelegt. PB4 beherrscht das Output-Enable des EPROMs und PB7 steuert den Programmierpuls, dessen Weite vom Brenn-Algorithmus abhängig ist. - Die an das EPROM anzulegenden bzw. aus ihm auszulesenden Daten laufen über den Datenport A der CIA. Hier hat es nun eine kleine Häßlichkeit in der Beschaltung des CBM und in seiner Interrupt-Routine: Man hat den Datenport nicht für sich allein. Er ist vielmehr auch mit dem Datenbustreiber 75160 für den IEEE-Bus verbunden. Und weiter: Die Routine IOINIT des Interruptes schaltet diesen Baustein beharrlich immer wieder auf Dateneingang (Signal TE am 75160, gesteuert vom Triport-Interfacebaustein 6525, Pin PA1). Wenn wir den 75160 hochohmig sperren wollen, dann müssen wir zunächst mit SEI den Systeminterrupt ausschließen, was immer bei der Benutzung des Datenports geschieht.

Zum Programm: Es berücksichtigt die o.a. Gegebenheiten. Nach der Initialisierung bei MAIN wird bei RE ENTER ein interaktives Menue gezeigt, das folgende Funktionen und Routinen zur Verfügung stellt: EINLESEN eines EPROMs, LISTEN eines Bereiches, Prüfen, ob leer (CHECKEN), Verifizieren (Verify), Abweichungen anzeigen (ABWEICH) und Programmieren (PROGRAM). Weiterhin kann ein RAM-Bereich zur Anzeige gebracht werden (RAMLIST), natürlich auch der Inhalt der Festwertspeicher. Auch beim Einlesen kann jede Speicherbank und jeder Bereich benutzt werden.
Folgende Algorithmen stehen zur Wahl: 50 ms-Impulse (alte Methode), Anfangsimpulse "intelligent" von jeweils 1 ms. Sobald die Information im EPROM festsitzt, erfolgt dann ein Nachbrennen (OVER BURN). Hier gibt es dann in der interaktiven Abfrage zwei Wahlmöglichkeiten: a) maximal 25 Impulse von 1 ms und danach ein Nachbrennen mit der dreifachen Dauer aus der Summe der Anfangsimpulse, b) maximal 15 Anfangsimpulse und vierfache Nachbrenndauer. In beiden Fällen liegt die Programmierspannung maximal 75 ms an einer Zelle an. Im praktischen Betrieb wird viel weniger Zeit gebraucht. Zellen, die nach 25 bzw. 15 Anfangsimpulsen die Information noch nicht akzeptiert haben, werden zur Menge der "schwachen Zellen" addiert. Diese Menge wird am Schluß angezeigt und dient zur Beurteilung des EPROMs. - Zellen, die die Information gar nicht annehmen wollen, werden zwischenzeitlich immer mit ihrer Adresse angezeigt (Zahl der fatalen Fehler). Am Schluß der Programmierung werden beide genannten Mengen zur Anzeige gebracht. Das Programm wartet dann auf das Abschalten der Spannungen und auf einen Tastendruck des Programmierers. Jetzt erfolgt noch eine Schlußprüfung. Der gesamte Bereich wird verglichen (Vorlage gegen EPROM) und Abweichungen nochmals mit Adresse und Soll/Ist angezeigt. - Nach jeder Funktion kommt wieder das Menue zur Anzeige.
Die Schaltung des Epromers befindet sich am Schluß des Artikels

```
0001                                    ;Ç$ko"promsubmit"
0002                                    ;submitfile fuer cbm-prommer
0004 000000             start           =$4000
0005 000000                             .fil f1:cbm-prommer1
0006                                    ;$ko"cbm-prommer1"
0007                                    ;programm f. den eprommer f. cbm 610/720
0008                                    ;brennen der typen 2764-27128
0009                                    ;datum 4.7.86. teilung des textes in cbm-promme
0009                                    r1 und ...2
0011                                    ;hardware-adressen und bit-muster in den regist
0011                                    ern:
0012 000000             i6509           =$01                    datenbank-register der cpu
0013                                    ; cia = complex interface adapter
0014 000000             cia             =$dc00                  cia des userports und des iec=
                                                                -bus
0015 000000             porta           =cia                    dient als datenbus
0016 000000             portb           =cia+1                  dient als steuerbus mit folge
                                                                nden belegungen:
0017                                    ;                       pb0=a12 adress-signal
0018                                    ;                       pb1=a13 dito
0019                                    ;                       pb2=nc
0020                                    ;                       pb3=nc
0021                                    ;                       pb4=oe output enable eprom
0022                                    ;                       pb5=reset fuer zaehler 4040
0023                                    ;                       pb6=clock-puls fuer dito, er
0023                                    zeugt adressen a0...a11
0024                                    ;                       pb7=pgm, programmierpuls fue
0024                                    r eprom
0026 000000             ddra            =cia+2                  datenrichtung
0027 000000             ddrb            =cia+3
0028 000000             ta_low          =cia+4                  timer a latch low
0029 000000             ta_high         =cia+5                  counter high
0030 000000             icr             =cia+$d                 interrupt control register
0031 000000             ta_bit          =$01                    interupt-bit des timer a im i
                                                                cr
0032 000000             cra             =cia+$e                 control register a/steuerung
                                                                timer a
0033                                    ;tpi = tri-port interface
0034                                    ; pa1 steuert te, die datenrichtung des 75160 i
0034                                    m cbm
0035                                    ; te=0 = 0 lesen, te=1 = schreiben
0036 000000             tpi             =$de00                  basisadresse
0037 000000             tpi_porta       =tpi+0                  porta
0038 000000             te_contr        =$02                    steuerung der datenrichtung 7
                                                                5160
0040                                    ;konstanten:
0041 000000             puls_1ms        =$0824                  1 ms-puls bei 2 mhz
0042                                    ;puls_50ms =$19708  50000 taktzyklen 2 mhz
0043 000000             cr              =$0d
0044 000000             space           =$20
0045 000000             bit6            =%01000000              bit fuer clock
0046                                    ;variablen im ram
0047 000000                             *=$43                   zero page-breich, ohne steuru
                                                                ng des cbm
0048 000043             temp            *=*+1                   hilfsregister
0049 000044             counter         *=*+2                   zuordnungszaehler fueer pulsu
                                                                ng
0050 000046             promlow         *=*+2                   prom-adresse, ab der gebrannt
                                                                 wird
0051 000048             promhigh        *=*+2                   letzte prom-adresse, die gebr
                                                                annt wird
0052 00004a             impulse         *=*+1                   zaehler fuer brennpulse
0053 00004b             ramlow          *=*+2                   ram-adresse low, brennen, les
                                                                en usw.
0054 00004d             codebank        *=*+1                   bank, in der der code steht
0055 00004e             ramhigh         *=*+2                   ram-endadresse,brennen, vergl
                                                                eichen
0056 000050             ramlfp          *=*+2                   laufende ram-adresse der zu b
                                                                rennenden vorlage
0057 000052             mode            *=*+1                   zeigt befehlsart an
0058 000053             errors          *=*+2                   fehlerzahl
0059 000055             schwach         *=*+2                   zahl schwacher zellen, die da
                                                                s byte schlecht annehmen
```

```
0060 000057            algo     *=*+1              algoritmus des brennens
0061 000058                     *=$62
0062 000062            jump     *=*+2              ablage indirekter sprungadres
                                                   sen
0063 000064            status   =$9c               e/a systemstatus
0064 000064                     *=$00b9
0065 0000b9            temp1    *=*+2              einlesen von adressen
0066 0000bb                     *=$fa
0067 0000fa            mesptr   *=*+2              pointer f. messages
0068 0000fc            bank15   =15                systembank
0069 0000fc            bank1    =1
0070 0000fc            bank2    =2                 bank des textspeichers
0071 0000fc            bank3    =3                 bank des objectcodes
0072 0000fc            bank4    =4                 bank der symboltafel
0074                            ;********* systemroutinen cbm 720
0075 0000fc            kbsout   =$ffd2             zeichenausgabe auf aktiven ka
                                                   nal
0076 0000fc            sequen   =$e668             aufruf escape-funktionen
0077 0000fc            wrob     =$f0fb             akku als 2 hexbytes ausgeben
0078 0000fc            rdoa     =$f123             4 hexziffern einlesen nach $b
                                                   9/ba
0079 0000fc            rdob     =$f130             2 hexziffern in den akku einl
                                                   esen
0080 0000fc            kgetin   =$ffe4             hole 1 zeichen aus tastatur-b
                                                   uffer
0081 0000fc            kbasin   =$ffcf             zeichen von aktivem kanal hol
                                                   en
0082 0000fc            kopen    =$ffc0             file oeffnen
0083 0000fc            kclrch   =$ffcc             alle kanäle schließen
0084 0000fc            kbsout   =$ffd2
0085 0000fc            kstop    =$ffe1             stop-key gedrueckt?
0086 0000fc            kclall   =$ffe7             allo files schliessen
0088 0000fc                     *=$03a0            systemvariable cbm
0089 0003a0            pagsav   *=*+1              rettung der urspruengl. daten
                                                   bank
0091                            ;***** hauptprogramm
0092 0003a1                     *=start            zuweisung per submitfile
0093 004000 a501       main     lda i6509          alte datenbank retten
0094 004002 8da003              sta pagsav
0095 004005 201145              jsr initall        interfaces zuruecksetzen
0096 004008 a900                lda #0             system-meldung ausgeben
0097 00400a 207f45              jsr text_out
0098 00400d 205245     main1    jsr getkey
0099 004010 c90d                cmp #cr            bestaetigt?
0100 004012 d0f9                bne main1
0102 004014 205845     re_enter jsr stopkey?       abbruch gewuenscht?
0103 004017 201145              jsr initall        zur sicherheit re-init.
0104 00401a a901                lda #1             reentry nach ausfuehrenden ro
                                                   utinen
0105 00401c 207f45              jsr text_out       hauptmenue
0106 00401f 205245     re_enter1 jsr getkey       befehlstaste holen
0107 004022 20d2ff              jsr kbsout
0108 004025 48                  pha                zeichen retten
0109 004026 204d45              jsr crlf
0110 004029 68                  pla                befehl zurueck
0111 00402a a207                ldx #7             8 befehle kommen in betracht
0112 00402c dd474a     befloop  cmp befehle,x
0113 00402f f006                beq executo
0114 004031 ca                  dex
0115 004032 10f8                bpl befloop
0116 004034 4c1f40              jmp re_enter1
0118 004037 8a         execute  t=a                befehlsentschluesselung
0119 004038 0a                  asl a              a mal 2
0120 004039 aa                  tax
0121 00403a bd4f4a              lda beftab,x
0122 00403d 8562                sta jump
0123 00403f bd504a              lda beftab+1,x
0124 004042 8563                sta jump+1
0125 004044 a900                lda #0
0126 004046 8552                sta mode           neutralisiere befehlsart
0127 004048 8553                sta errors         fehlerzahl
0128 00404a 8554                sta errors+1
0129 00404c 6c6200              jmp (jump)         zur ausfuehrung
```

```
0131 00404f a9c0     abweich  lda #$c0          abweichungen auslisten
0132 004051 8552              sta mode.
0133 004053 20d243            jsr open_rd2
0134 004056 4c6040            jmp einlesen0
0135 004059 a980     verify   lda #$80          befehlsart
0136 00405b 8552              sta mode
0137 00405d 20c443   einlesen jsr open_read     interface etc. oeffnen
0138 004060 78       einlesen0 sei              interrupt sperren, er zerschl
                                                aegt te_contr im tpi
0139 004061 20bd44   lesen2   jsr wait_abit     konsolidieren
0140 004064 ad00de            lda tpi_porta     schalte te_contr auf ausgang
0141 004067 0902              ora #te_contr
0142 004069 8d00de            sta tpi_porta
0143 00406c ad00dc            lda porta         daten holen
0144 00406f 85b9              sta temp1
0145 004071 a000              ldy #0
0146 004073 2452              bit mode
0147 004075 300e              bmi lesen2a
0148 004077 a64d              ldx codebank      bank steuern
0149 004079 8601              stx i6509
0150 00407b 9150              sta (ramlfp),y    byte ablegen
0151 00407d aea003            ldx pagsav
0152 004080 8601              stx i6509
0153 004082 4cac40            jmp lesen2b
0154 004085 a64d     lesen2a  ldx codebank
0155 004087 8601              stx i6509         bank stouern
0156 004089 b150              lda (ramlfp),y
0157 00408b aea003            ldx pagsav
0158 00408e 8601              stx i6509         alte bank herrichten
0159 004090 c5b9              cmp temp1         vergleich der bytes
0160 004092 f018              beq lesen2b
0161 004094 8543              sta temp          zeichen f. ausgabe merken
0162 004096 f8                sed               fehler dezimal addieren
0163 004097 18                clc
0164 004098 a901              lda #1
0165 00409a 6553              adc errors
0166 00409c 8553              sta errors
0167 00409e a554              lda errors+1
0168 0040a0 6900              adc #0
0169 0040a2 8554              sta errors+1
0170 0040a4 d8                cld
0171 0040a5 2452     lesen2aa bit mode          fehler anzeigen?
0172 0040a7 5003              bvc lesen2b
0173 0040a9 202b45            jsr zeige_abw
0174 0040ac a545     lesen2b  lda counter+1     endbedingung erreicht?
0175 0040ae c549              cmp promhigh+1
0176 0040b0 d02e              bne lesen3
0177 0040b2 a544              lda counter
0178 0040b4 c548              cmp promhigh
0179 0040b6 d028              bne lesen3
0180 0040b8 58                cli               interrupt wieder erlaubt
0181 0040b9 a554              lda errors+1
0182 0040bb 0553              ora errors
0183 0040bd d00c              bne lesen2d
0184 0040bf a552              lda mode          wird nur eingelesen?
0185 0040c1 f005              beq lesen2f       ja
0186 0040c3 a904              lda #4            meldung:keine fehler
0187 0040c5 207f45   lesen2e  jsr text_out
0188 0040c8 4c1440   lesen2f  jmp re_enter      fertig, zum menue
0189 0040cb a905     lesen2d  lda #5
0190 0040cd 207f45            jsr text_out      fehleranzeige
0191 0040d0 204d45            jsr crlf
0192 0040d3 a554              lda errors+1
0193 0040d5 20fbf0            jsr wrob
0194 0040d8 a553              lda errors
0195 0040da 20fbf0            jsr wrob
0196 0040dd 4c1440            jmp re_enter      fertig
0198 0040e0 204b44   lesen3   jsr ram_plus      zeiger erhoehen
0199 0040e3 20a944            jsr puls_plus     neue adresse anlegen
0200 0040e6 20f343            jsr gen_a12
0201 0040e9 4c6140            jmp lesen2
0203 0040ec 201c44   ramlist  jsr ram_von       anzeige eines ram-bereiches
0204 0040ef 205244            jsr ram_bis       abfragen
```

```
0205 0040f2 204244            jsr ramlfpint
0206 0040f5 a000              ldy #0
0207 0040f7 a20f      ramlist1 ldx #15          16 items pro zeile
0208 0040f9 8643              stx temp
0209 0040fb a551              lda ramlfp+1      zunaechst adresse anzeigen.
0210 0040fd 20fbf0            jsr wrob          ausgabe 2 hexziffern
0211 004100 a550              lda ramlfp
0212 004102 20fbf0            jsr wrob
0213 004105 a920              lda #space
0214 004107 20d2ff            jsr kbsout
0215 00410a a64d      ramlist2 ldx codebank     auf entsprechende Bank switch
                                                en
0216 00410c 8601              stx i6509
0217 00410e b150              lda (ramlfp),y
0218 004110 aea003            ldx pagsav
0219 004113 8601              stx i6509
0220 004115 20fbf0            jsr wrob          hexbyte ausgeben
0221 004118 a920              lda #space
0222 00411a 20d2ff            jsr kbsout
0223 00411d 204b44            jsr ram_plus      pointer +1
0224 004120 c643              dec temp
0225 004122 10e6              bpl ramlist2      immer 16 items ausgeben
0226 004124 204d45            jsr crlf
0227 004127 38                sec
0228 004128 a550              lda ramlfp        ende?
0229 00412a e54e              sbc ramhigh
0230 00412c a551              lda ramlfp+1
0231 00412e e54f              sbc ramhigh+1
0232 004130 90c5              bcc ramlist1
0233 004132 4c1440            jmp re_enter      fertig
0235 004135 a980      listen  lda #$80          merke modus, prom listen
0236 004137 8552              sta mode
0237 004139           checken                   sehen, ob prom leer ist
0238 004139 206144            jsr prom_von
0239 00413c 207044            jsr prom_bis      anfang und ende?
0240 00413f 207f44            jsr puls_up
0241 004142 20de43            jsr read_int
0242 004145 78                sei               interrupt unterbinden
0243 004146 ad00de            lda tpi_porta     schalte te_contr auf ausgang
0244 004149 0902              ora #te_contr
0245 00414b 8d00de            sta tpi_porta
0246 00414e a000              ldy #0            relativer zeiger
0247 004150 a200      checke  ldx #0            spaltenzaehler
0248 004152 8643              stx temp
0249 004154 2452              bit mode
0250 004156 100f              bpl checken0      nein, wenn auf leer geprueft
                                                wird
0251 004158 a545              lda counter+1     adresse ausgeben
0252 00415a 20fbf0            jsr wrob
0253 00415d a544              lda counter
0254 00415f 20fbf0            jsr wrob
0255 004162 a920              lda #space
0256 004164 20d2ff            jsr kbsout
0257 004167 20bd44    checken0 jsr wait_abit    konsolidieren
0258 00416a ad00dc            lda porta         eingabewert
0259 00416d 2452              bit mode
0260 00416f 3008              bmi checken1      listen, nicht vergleichen
0261 004171 c9ff              cmp #$ff          leer?
0262 004173 f00c              beq checken1
0263 004175 58                cli               interrupt wieder zulassen
0264 004176 4ccb40            jmp losen2d       fehler
0265 004179 20fbf0    checken1 jsr wrob         ausgabe byte
0266 00417c a920              lda #space
0267 00417e 20d2ff            jsr kbsout
0268 004181 a545      checken1 lda counter+1     endbedingung erreicht?
0269 004183 c549              cmp promhigh+1
0270 004185 d013              bne checken2
0271 004187 a544              lda counter
0272 004189 c548              cmp promhigh
0273 00418b d00d              bne checken2
0274 00418d 2452              bit mode
0275 00418f 3005              bmi checken1a
0276 004191 a90b              lda #11
```

```
0277 004193 207f45            jsr text_out        prom ist leer
0278 004196 58       checkenia cli
0279 004197 4c1440            jmp re_enter
0280 00419a 20a944   checken2  jsr puls_plus
0281 00419d 2452              bit mode            bei leerpruefung keine anzei
                                                  ge
0282 00419f 10c6              bpl checken0
0283 0041a1 e643              inc temp
0284 0041a3 a643              ldx temp
0285 0041a5 e010              cpx #16             16 zeichen dargestellt?
0286 0041a7 d0be              bne checken0
0287 0041a9 204d45            jsr crlf
0288 0041ac 4c3041            jmp checke
0289 0041af a90c     fehler   lda #12             adressbereich unklar
0290 0041b1 207f45            jsr text_out
0291 0041b4 4c1440   fehler1  jmp re_enter        eingaben wiederholen
0293 0041b7 201c44   program  jsr ram_von         eprom programmieren
0294 0041ba 205244            jsr ram_bis
0295 0041bd 201244            jsr ram_err?
0296 0041c0 90ed              bcc fehler
0297 0041c2 206144            jsr prom_von
0298 0041c5 207044            jsr prom_bis        bereichsabfragen
0299 0041c8 200844            jsr prom_err?
0300 0041cb 90e2              bcc fehler
0301 0041cd 38                sec                 pruefung gleicher mengen
0302 0041ce a54e              lda ramhigh
0303 0041d0 e54b              sbc ramlow
0304 0041d2 85fa              sta mesptr
0305 0041d4 a54f              lda ramhigh+1
0306 0041d6 e54c              sbc ramlow+1
0307 0041d8 85fb              sta mesptr+1        ablage zum vergleich
0308 0041da 90d3              bcc fehler          falsch adressiert
0309 0041dc 38                sec                 nun menge prom-bytes
0310 0041dd a548              lda promhigh
0311 0041df e546              sbc promlow
0312 0041e1 8550              sta ramlfp          ablage zum vergleich
0313 0041e3 a549              lda promhigh+1
0314 0041e5 e547              sbc promlow+1
0315 0041e7 8551              sta ramlfp+1        z. vergleich
0316 0041e9 90c4              bcc fehler
0317 0041eb c5fb              cmp mesptr+1
0318 0041ed d0c0              bne fehler
0319 0041ef a550              lda ramlfp
0320 0041f1 c5fa              cmp mesptr          die andere menge
0321 0041f3 d0ba              bne fehler
0322 0041f5 a90a              lda #10             abfrage, ob sicher
0323 0041f7 207f45            jsr text_out
0324 0041fa 205245            jsr getkey
0325 0041fd c94a              cmp #'j'            ja?
0326 0041ff d0b3              bne fehler1         nein
0327 004201 207f44            jsr puls_up         zaehlerverlauf
0328 004204 204244            jsr ramlfpint       laufpointer ram einstellen
0329 004207 a90f              lda #15             abfrage ob normal oder intell
                                                  igent brennen
0330 004209 207f45            jsr text_out
0331 00420c 205245            jsr getkey
0332 00420f c949              cmp #'i'
0333 004211 f005              beq progr0          intelligent brennen
0334 004213 c652              dec mode            negativ machen als flag
0335 004215 4c2a42            jmp progr1
0336 004218 a911     progr0   lda #17
0337 00421a 207f45            jsr text_out        abfrage ob overburn x3 oder x
                                                  4
0338 00421d 205245            jsr getkey
0339 004220 c935              cmp #'5'
0340 004222 b0f4              bcs progr0          nur 3 und 4 zugelassen
0341 004224 c933              cmp #'3'
0342 004226 90f0              bcc progr0
0343 004228 8557              sta algo            algorithmus merken
0344 00422a a902     progr1   lda #2              21 volt einschalten
0345 00422c 207f45            jsr text_out
0346 00422f 205245            jsr getkey          bestaetigung?
0347 004232 c90d              cmp #cr
```

```
0348 004234 d0f4            bne progr1
0349 004236 a90d            lda #13          anzeige, dass gebrannt wird
0350 004238 207f45          jsr text_out
0351 00423b 78              sei              interrupt unterbunden
0352 00423c ad00de          lda tpi_porta    schalte te_contr auf ausgang
0353 00423f 0902            ora #te_contr
0354 004241 8d00de          sta tpi_porta
0355 004244 a000            ldy #0
0356 004246 ad04dc          lda ta_low       interrupt-flag vorsorglich lo
                                             eschen
0357 004249 844a            sty impulse      =00
0358 00424b 8453            sty errors
0359 00424d 8454            sty errors+1     fehlerzahl=0
0360 00424f 8455            sty schwach      zahl schwacher zellen
0361 004251 8456            sty schwach+1
0362 004253 2452            bit mode         welcher algorithmus?
0363 004255 1003            bpl progr2       intelligent
0364 004257 4c9943          jmp nprogr       normal
0365 00425a 20f343  progr2  jsr gen_a12       hohe adresse anlegen
0366 00425d a900            lda #0
0367 00425f 8d02dc          sta ddra         abkuerzen, wenn byte schon gl
                                             eich
0368 004262 ad01dc          lda portb
0369 004265 29ef            and #$ef         output enable low
0370 004267 8d01dc          sta portb
0371 00426a a64d            ldx codebank
0372 00426c 8601            stx i6509
0373 00426e b150            lda (ramlfp),y   byte in der ram-bank
0374 004270 aea003          ldx pagsav
0375 004273 8601            stx i6509
0376 004275 cd00dc          cmp porta        byte im eprom
0377 004278 d003            bne progr2a
0378 00427a 4c2643          jmp prog04       brennen uebergehen, wenn glei
                                             ch
0379 00427d 20be44  progr2a jsr burn_1ms     brennimpuls erzeugen
0380 004280 f021            beq over_burn    data sitzt schon fest
0381 004282 a54a            lda impulse      pruefen, ob eprom gut ist
0382 004284 a657            ldx algo         limit pruefen
0383 004286 e034            cpx #'4'
0384 004288 f006            beq progr3
0385 00428a c919            cmp #25          bis zu 25 + 3x 25 impulse bei
                                              algo=3
0386 00428c 90cc            bcc progr2       + 1 schuss
0387 00428e b004            bcs progr3a
0388 004290 c90f    progr3  cmp #15          limit noch nicht erreicht
0389 004292 90c6            bcc progr2       + 1 schuss
0390 004294 18      progr3a clc
0391 004295 f8              sed              dezimal: zahl schwacher zelle
                                             n addieren
0392 004296 a555            lda schwach
0393 004298 6901            adc #1
0394 00429a 8555            sta schwach
0395 00429c a556            lda schwach+1
0396 00429e 6900            adc #0
0397 0042a0 8556            sta schwach+1
0398 0042a2 d8              cld
0399 0042a3         over_burn                ; nun 3x-4x summe bisheriger
                                             zeit brennen
0400 0042a3 a54a            lda impulse
0401 0042a5 a657            ldx algo         limit pruefen
0402 0042a7 e034            cpx #'4'
0403 0042a9 f007            beq overba
0404 0042ab 0a              asl a            a impulse x 2
0405 0042ac 18              clc
0406 0042ad 654a            adc impulse      = impulse x 3
0407 0042af 4cb442          jmp overbc
0408 0042b2 0a      overba  asl a            a x2
0409 0042b3 0a              asl a            a nun x4
0410 0042b4 8543    overbc  sta temp         menge merken
0411 0042b6 a9ff            lda #$ff         porta zum ausgang machen
0412 0042b8 8d02dc          sta ddra
0413 0042bb a64d            ldx codebank
0414 0042bd 8601            stx i6509
```

```
0415 0042bf b150          lda (ramlfp),y    byte holen zum brennen
0416 0042c1 aea003        ldx pagsav
0417 0042c4 8601          stx i6509
0418 0042c6 8d00dc        sta porta         datenport
0419 0042c9 48            pha               byte merken
0420 0042ca a924          lda #<puls_1ms    timer 1 vorladen, low
0421 0042cc 8d04dc        sta ta_low        ins latch
0422 0042cf a908          lda #>puls_1ms    nun pulsausgabe starten
0423 0042d1 8d05dc        sta ta_high       timer high laden
0424 0042d4 ad01dc        lda portb
0425 0042d7 297f          and #$7f          burn-pin auf low ziehen
0426 0042d9 0910          ora #$10          output enable high machen
0427 0042db 8d01dc        sta portb
0428 0042de a9d9    overb1 lda #$d9
0429 0042e0 8d0edc        sta cra           start des timers
0430 0042e3 a501          lda ta_bit        abfragemuster
0431 0042e5 2c0ddc  overb2 bit icr          wann kommt das interrupt-flag
                                              ?
0432 0042e8 f0fb          beq overb2        noch nicht
0433 0042ea ad04dc        lda ta_low        interrupt-flag beseitigen
0434 0042ed c643          dec temp          menge
0435 0042ef d0ed          bne overb1
0436 0042f1 ad01dc        lda portb         burn-pin wieder high
0437 0042f4 0980          ora #$80
0438 0042f6 29ef          and #$ef          output enable auf low
0439 0042f8 8d01dc        sta portb
0440 0042fb 20bd44        jsr wait_abit
0441 0042fe a900          lda #0            porta lesbar machen
0442 004300 8d02dc        sta ddra
0443 004303 68            pla               byte-vorlage zurueckholen
0444 004304 20bd44        jsr wait_abit
0445 004307 cd00dc        cmp porta         holen, was schon erzeugt wurd
                                              e
0446 00430a f01a          beq prog04        schlusspruefung
0447 00430c a905          lda #5            prom ist ungleich
0448 00430e 207f45        jsr text_out
0449 004311 202b45        jsr zeige_abw     adresse zeigen
0450 004314 204d45        jsr crlf
0451 004317 f8            sed               dezimal rechnen
0452 004318 18            clc
0453 004319 a553          lda errors        fehler +1
0454 00431b 6901          adc #1
0455 00431d 8553          sta errors
0456 00431f a554          lda errors+1
0457 004321 6900          adc #0
0458 004323 8554          sta errors+1
0459 004325 d8            cld
0460 004326 38      prog04 sec              nun endbedingung pruefen
0461 004327 a550          lda ramlfp
0462 004329 e54e          sbc ramhigh
0463 00432b a551          lda ramlfp+1
0464 00432d e54f          sbc ramhigh+1
0465 00432f 9041          bcc prog_fort     weiter brennen
0466 004331 d03f          bne prog_fort     dito
0467 004333 a900          lda #0
0468 004335 201145        jsr initall       fertig, impulsausgabe unterbi
                                              nden
0469 004338 a903    prog4 lda #3            meldung an bediener
0470 00433a 207f45        jsr text_out      :1 volt abschalten
0471 00433d 58            cli
0472 00433e 205245        jsr getkey
0473 004341 c90d          cmp #cr
0474 004343 d0f3          bne prog4
0475 004345 78            sei
0476 004346 a554          lda errors+1      fehlerzahl ausgeben
0477 004348 20fbf0        jsr wrob
0478 00434b a553          lda errors
0479 00434d 20fbf0        jsr wrob
0480 004350 a920          lda #space
0481 004352 20d2ff        jsr kbsout
0482 004355 a910          lda #16           fehlertext
0483 004357 207f45        jsr text_out
0484 00435a a900          lda #0
```

```
0485 00435c 8553            sta errors
0486 00435e 8554            sta errors+1
0487 004360 a556            lda schwach+1
0488 004362 20fbf0          jsr wrob
0489 004365 a555            lda schwach
0490 004367 20fbf0          jsr wrob
0491 00436a a913            lda #19
0492 00436c 207f45          jsr text_out        zahl schwacher zellen anzeige
                                                n
0493 00436f 4c4f40          jmp abweich         fertig, nun nocheinmal alles
                                                checken
0495 004372 20a944  prog_fort jsr puls_plus     prom-adresse +1
0496 004375 204b44          jsr ram_plus
0497 004378 a900            lda #0
0498 00437a 854a            sta impulse
0499 00437c a544            lda counter         lebenszeichen alle 256 bytes
0500 00437e d012            bne prog_fa         nein
0501 004380 a545            lda counter+1       adresse ausgeben
0502 004382 20fbf0          jsr wrob
0503 004385 a544            lda counter
0504 004387 20fbf0          jsr wrob
0505 00438a a90d            lda #13             anzeige, dass gebrannt wird
0506 00438c 207f45          jsr text_out
0507 00438f 204d45          jsr crlf            neue zeile
0508 004392 2452    prog_fa bit mode            normal/intelligent?
0509 004394 3003            bmi nprogr          normal
0510 004396 4c5a42  prog_fi jmp progr2          weiteres byte brennen
0511 004399 20bd44  nprogr  jsr wait_abit       normal 50 ms programmieren
0512 00439c 20f343          jsr gen_a12         hohe adresse anlegen
0513 00439f a900            lda #0
0514 0043a1 8d02dc          sta ddra            abkuerzen, wenn byte schon gl
                                                eich
0515 0043a4 ad01dc          lda portb
0516 0043a7 29ef            and #$ef            output enable low
0517 0043a9 8d01dc          sta portb
0518 0043ac a64d            ldx codebank
0519 0043ae 8601            stx i6509
0520 0043b0 b150            lda (ramlfp),y      byte in der ram-bank
0521 0043b2 aea003          ldx pagsav
0522 0043b5 8601            stx i6509
0523 0043b7 cd00dc          cmp porta           byte im eprom
0524 0043ba d003            bne nprogr1
0525 0043bc 4c2643          jmp prog04          byte ist schon gleich, skip o
                                                ver
0526 0043bf a932    nprogr1 lda #50
0527 0043c1 4cb442          jmp overbc          50 ms brennen
0528 0043c4 201c44  open_read jsr ram_von       eprom ins ram lesen/vergleich
                                                en
0529 0043c7 206144  open_rd1 jsr prom_von
0530 0043ca 207044          jsr prom_bis
0531 0043cd 200844          jsr prom_err?       bereich pruefen.
0532 0043d0 90f5            bcc open_rd1        fehler
0533 0043d2 207f44  open_rd2 jsr puls_up        adressen einstellen
0534 0043d5 204244          jsr ramlfpint       ram-zeiger initialisieren
0535 0043d8 20de43          jsr read_int        daten-lesen einrichten
0536 0043db a000            ldy #0
0537 0043dd 60              rts
0538 0043de a900    read_int lda #0             datenport wird eingang
0539 0043e0 8d02dc          sta ddra            datenrichtung
0540 0043e3 ad00de          lda tpi_porta
0541 0043e6 29fd            and #$ff-te_contr   umsteuerung auf lesen
0542 0043e8 8d00de          sta tpi_porta
0543 0043eb ad01dc          lda portb
0544 0043ee 29ef            and #$ef            output enable low ziehen
0545 0043f0 8d01dc          sta portb
0546 0043f3 a545    gen_a12 lda counter+1       lege high address auf portb
0547 0043f5 2930            and #%00110000      a13, a12 isolieren
0548 0043f7 4a              lsr a
0549 0043f8 4a              lsr a
0550 0043f9 4a              lsr a
0551 0043fa 4a              lsr a
0552 0043fb 8543            sta temp            zwischenspeichern
0553 0043fd ad01dc          lda portb
```

```
0554 004400 29fc              and #%11111100     alte a13, a12 loeschen
0555 004402 0543              ora temp           ,neue uebernehmen
0556 004404 8d01dc            sta portb
0557 004407 60                rts
0558 004408 38       prom_err? sec
0559 004409 a548              lda promhigh       adresseingaben pruefen
0560 00440b e546              sbc promlow        promhigh muss >= promlow sein
0561 00440d a549              lda promhigh+1
0562 00440f e547              sbc promlow+1
0563 004411 60                rts
0564 004412 38       ram_err?  sec               adressen pruefen
0565 004413 a54e              lda ramhigh
0566 004415 e54b              sbc ramlow
0567 004417 a54f              lda ramhigh+1
0568 004419 e54c              sbc ramlow+1
0569 00441b 60                rts
0570 00441c a912     ram_von   lda #18            quellbank abfragen
0571 00441e 207f45            jsr text_out
0572 004421 2030f1            jsr rdob           tastaturabfrage
0573 004424 854d              sta codebank
0574 004426 204d45            jsr crlf
0575 004429 a908              lda #8             prompt ausgeben
0576 00442b 207f45            jsr text_out
0577 00442e a900              lda #0
0578 004430 85b9              sta temp1
0579 004432 85ba              sta temp1+1
0580 004434 2023f1            jsr rdoa           hole ram-startadresse
0581 004437 a5b9              lda temp1
0582 004439 a6ba              ldx temp1+1
0583 00443b 854b              sta ramlow
0584 00443d 864c              stx ramlow+1
0585 00443f 4c4d45            jmp crlf           rts dort
0586 004442 a54b     ramlfpint lda ramlow
0587 004444 8550              sta ramlfp         ram-laufpointer auf anfang se
                                                 tzen
0588 004446 a54c              lda ramlow+1
0589 004448 8551              sta ramlfp+1
0590 00444a 60                rts
0591 00444b e650     ram_plus  inc ramlfp
0592 00444d d002              bne ramlfpad1
0593 00444f e651              inc ramlfp+1
0594 004451 60       ramlfpad1 rts
0595 004452 a909     ram_bis   lda #9
0596 004454 207f45            jsr text_out
0597 004457 207745            jsr getadres1      begrenzer-adresse f. ram hole
                                                 n
0598 00445a 854e              sta ramhigh
0599 00445c 864f              stx ramhigh+1
0600 00445e 4c4d45            jmp crlf
0601 004461 a906     prom_von  lda #6
0602 004463 207f45            jsr text_out
0603 004466 207745            jsr getadres1      startadresse prom holen
0604 004469 8546              sta promlow
0605 00446b 8647              stx promlow+1
0606 00446d 4c4d45            jmp crlf
0607 004470 a907     prom_bis  lda #7             prompt
0606 004472 207f45            jsr text_out
0607 004475 207745            jsr getadres1      adresse prom holen
0608 004478 8548              sta promhigh
0609 00447a 8649              stx promhigh+1
0610 00447c 4c4d45            jmp crlf
0612 00447f          puls_up                     impulse ausgeben, bis prom-st
                                                 artadresse erreicht
0613 00447f ad01dc            lda portb          zaehler an reset des 4040 ein
                                                 schalten
0614 004482 29df              and #$df
0615 004484 8d01dc            sta portb
0616 004487 38       puls_up1  sec
0617 004488 a544              lda counter
0618 00448a e546              sbc promlow
0619 00448c a545              lda counter+1
0620 00448e e547              sbc promlow+1
0621 004490 f016              beq puls_end
```

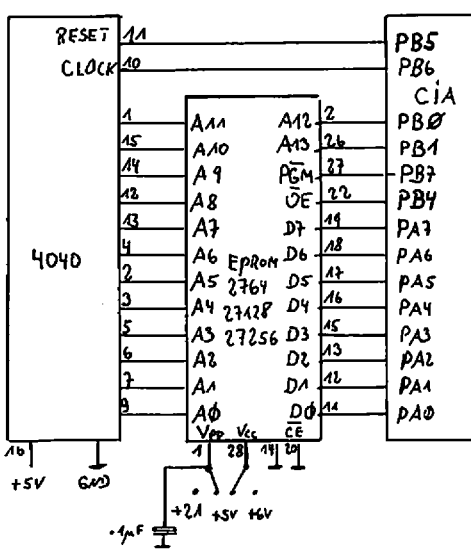
```
0622 004492 ad01dc            lda portb
0623 004495 4940              eor #bit6          abfallende flanke erzeugen
0624 004497 8d01dc            sta portb
0625 00449a e644              inc counter        ,nutze zwischenzeit
0626 00449c d002              bne puls_up2
0627 00449e e645              inc counter+1
0628 0044a0 4940    puls_up2  eor #bit6          aufsteigende flanke
0629 0044a2 8d01dc            sta portb
0630 0044a5 4c8744            jmp puls_up1
0631 0044a8 60      puls_end  rts
0633 0044a9 ad01dc  puls_plus lda portb
0634 0044ac 4940              eor #bit6          abfallende flanke erzeugen
0635 0044ae 8d01dc            sta portb
0636 0044b1 e644              inc counter        ,nutze zwischenzeit
0637 0044b3 d002              bne puls_pl2
0638 0044b5 e645              inc counter+1
0639 0044b7 4940    puls_pl2  eor #bit6          aufsteigende flanke
0640 0044b9 8d01dc            sta portb
0641 0044bc 60                rts
0642 0044bd 60      wait_abit rts                konsolidierung
0644 0044be a9ff    burn_1ms  lda #$ff           porta zum ausgang machen
0645 0044c0 8d02dc            sta ddra
0646 0044c3 a64d              ldx codebank
0647 0044c5 8601              stx i6509
0648 0044c7 b150              lda (ramlfp),y     byte holen zum brennen
0649 0044c9 aea003            ldx pagsav
0650 0044cc 8601              stx i6509
0651 0044ce 8d00dc            sta porta          datenport
0652 0044d1 48                pha                byte fuer vergleich merken
0653 0044d2 20bd44            jsr wait_abit      signale konsolidieren
0654 0044d5 a924              lda #<puls_1ms     timer 1 vorladen, low
0655 0044d7 8d04dc            sta ta_low         ins latch
0656 0044da a908              lda #>puls_1ms     nun pulsausgabe starten
0657 0044dc 8d05dc  burn0     sta ta_high        timer high laden
0658 0044df a9d9              lda #$d9
0659 0044e1 8d0edc            sta cra            start des timers
0660 0044e4 ad01dc            lda portb
0661 0044e7 297f              and #$7f           burn-pin auf low ziehen
0662 0044e9 0910              ora #$10           output enable high machen
0663 0044eb 8d01dc            sta portb
0664 0044ee a901              lda #ta_bit
0665 0044f0 2c0ddc  burn1     bit icr            wann kommt das interrupt-flag
                                                 ?
0666 0044f3 f0fb              beq burn1          noch nicht
0667 0044f5 ad04dc            lda ta_low         interrupt-flag beseitigen
0668 0044f8 ad01dc            lda portb          burn-pin wieder high
0669 0044fb 0980              ora #$80
0670 0044fd 29ef              and #$ef           output enable auf low

0671 0044ff 8d01dc            sta portb
0672 004502 e64a              inc impulse
0673 004504 20bd44            jsr wait_abit      konsolidieren
0674 004507 a900              lda #0             porta lesbar machen
0675 004509 8d02dc            sta ddra
0676 00450c 68      burn2.    pla                byte-vorlage zurueckholen
0677 00450d cd00dc            cmp porta          holen, was schon erzeugt wurd
                                                 e
0678 004510 60      burnend   rts
0679                          ;initialisierung der interfaces
0680 004511 a9ff    initall   lda #$ff
0681 004513 8d03dc            sta ddrb           datenrichtung=ausgang
0682 004516 8d01dc            sta portb
0683 004519 a900              lda #0
0684 00451b 8544              sta counter        zaehler zuruecksetzen
0685 00451d 8545              sta counter+1
0686                          ;mach' datenport zu eingang
0687 00451f 8d02dc            sta ddra
0688 004522 ad00de            lda tpi_porta
0689 004525 29fd              and #$ff-te_contr
0690 004527 8d00de            sta tpi_porta
0691 00452a 60                rts
0693 00452b 204d45  zeige_abw jsr crlf           abweichungen anzeigen
0694 00452e a545              lda counter+1
```

```
0695 004530 20fbf0          jsr wrob            adresse ausgeben
0696 004533 a544            lda counter
0697 004535 20fbf0          jsr wrob
0698 004538 a920            lda #' '            space ausgeben
0699 00453a 20d2ff          jsr kbsout
0700 00453d a543            lda temp            das zeichen im ram zeigen
0701 00453f 20fbf0          jsr wrob
0702 004542 a920            lda #' '            space ausgeben
0703 004544 20d2ff          jsr kbsout
0704 004547 a5b9            lda temp1           das zuerst im eprom gelesene
                                                zeichen zeigen
0705 004549 20fbf0          jsr wrob
0706 00454c 60              rts
0707 00454d a90d      crlf  lda #cr             output cr/lf to active device
0708 00454f 4cd2ff          jmp kbsout
0709 004552 20e4ff    getkey jsr kgetin         hole zeichen vom akt. kanal,
                                                cbm kgetin
0710 004555 f0fb            beq getkey          noch kein zeichen
0711 004557 60              rts
0712 004558 20e1ff    stopkey? jsr kstop        programmabbruch mit stop-key
0713 00455b f001            beq monitor
0714 00455d 60        rchek2 rts
0715 00455e           monitor                   ;monitorprogramm des cbm ansp
                                                ringen
0716 00455e 20e7ff          jsr kclall
0717 004561 00              brk
0718 004562 48        pagres pha                alte datenbank wiederherstell
                                                en
0719 004563 ada003          lda pagsav
0720 004566 8501            sta i6509
0721 004568 68              pla
0722 004569 60              rts
0723 00456a a912      getadress lda #18         quellbank abfragen
0724 00456c 207f45          jsr text_out
0725 00456f 2030f1          jsr rdob            tastaturabfrage
0726 004572 854d            sta codebank
0727 004574 204d45          jsr crlf
0728 004577 2023f1    getadres1 jsr rdoa        4 bytes = 2 hexziffern
0729 00457a a5b9            lda temp1
0730 00457c a6ba            ldx temp1+1
0731 00457e 60              rts
0733                        ;ausgabe eines textes, die text-nummer muss im
0733                        akku sein
0734                        ;textbegrenzung durch 00
0736 00457f 0a        text_out asl a            a multiplikation nummer im ak
                                                ku mit 2
0737 004580 a8              tay
0738 004581 b91f4a          lda mestaf,y        pointer zur message zurechtle
                                                gen
0739 004584 85fa            sta mesptr
0740 004586 b9204a          lda mestaf+1,y
0741 004589 85fb            sta mesptr+1        message-pointer
0742 00458b a90f            lda #bank15
0743 00458d 8501            sta i6509
0744 00458f a000            ldy #0
0745 004591 b1fa      txt1  lda (mesptr),y      zeichen holen
0746 004593 f009            beq txtend          null als blockbegrenzer
0747 004595 297f            and #$7f            reines ascii herstellen
0748 004597 20d2ff          jsr kbsout          zeichenausgabe
0749 00459a c8              iny                 positionierung auf folgendes
                                                byte
0750 00459b 4c9145          jmp txt1
0751 00459e a000      txtend ldy #0             vorsorglich
0752 0045a0 60              rts
0754                        ;interaktive meldungen
0755 0045a1 0d        mes1  .byt cr,'prommer fuer 2764 und 27128, r. leahr',
                                                cr
0755 0045a2 50524f4d
0755 0045c7 0d
0756 0045c8 5330414e        .byt 'spannung +5 volt eingeschaltet?',cr
0756 0045e7 0d
0757 0045e8 32312056        .byt '21 volt ausgeschaltet? cr=ja',cr,00
0757 004604 0d00
```

```
0758 004606 0d        mes2      .byt cr,'hauptmenue, wahl mit taste',cr
0758 004607 48415550
0758 004621 0d
0759 004622 45202045            .byt 'e  einlesen prom-inhalt ins ram',cr
0759 004641 0d
0760 004642 4c20204c            .byt 'l  listen eines prom-bereiches',cr
0760 004660 0d
0761 004661 52202052            .byt 'r  ram-bereich listen',cr
0761 004676 0d
0762 004677 56202056            .byt 'v  verifizieren prom/ram',cr
0762 00468f 0d
0763 004690 43202043            .byt 'c  checken ob leer',cr
0763 0046a2 0d
0764 0046a3 50202050            .byt 'p  programmieren',cr
0764 0046b3 0d
0765 0046b4 41202041            .byt 'a  abweichungen listen?',cr
0765 0046cb 0d
0766 0046cc 51202051            .byt 'q  quit zum monitorprogramm',cr,00
0766 0046e7 0d00
0767 0046e9 57454e4e mes3       .byt 'wenn intelligente programmierung, dann ...
                                     ',cr,cr
0767 004713 0d0d
0768 004715 5a554552            .byt 'zuerst +6V einschalten, erst danach',cr
0768 004738 0d
0769 004739 32312056            .byt '21 volt brennspannung zuschalten',cr,cr
0769 004759 0d0d
0770 00475b 51554954            .byt 'quittieren mit cr, wenn eingeschaltet',cr,
                                     00
0770 004780 0d00
0771 004782 46455254 mes4       .byt 'fertig, 21 volt brennspannung ausschalten
                                     cr=aus',cr,cr
0771 0047b2 0d0d
0772 0047b4 41554620            .byt 'auf 5 volt zurueckschalten !',cr,00
0772 0047d0 0d00
0773 0047d2 2050524f mes5       .byt ' prom ist gleich',cr,cr,00
0773 0047e2 0d0d00
0774 0047e5 0d        mes6      .byt cr,' ***** achtung: prom ist ungleich ****'
                                     ,cr,00
0774 0047e6 202a2a2a
0774 00480c 0d00
0775 00480e 41422050 mes7       .byt 'ab prom-adresse?',cr,00
0775 00481e 0d00
0776 004820 42495320 mes8       .byt 'bis prom-adresse?',cr,00
0776 004831 0d00
0777 004833 41422052 mes9       .byt 'ab ram-adresse?',cr,00
0777 004842 0d00
0778 004844 42495320 mes10      .byt 'bis ram-adresse?',cr,00
0778 004854 0d00
0779 004856 49535420 mes11      .byt 'ist das sicher? j=ja',cr,00
0779 00486a 0d00
0780 00486c 0d        mes12     .byt cr,'prom ist leer',cr,00
0780 00486d 50524f4d
0780 00487a 0d00
0781 00487c 0d        mes13     .byt cr,'adressbereiche unklar !',cr,00
0781 00487d 41445245
0781 004894 0d00
0782 004896 0d        mes14     .byt cr,' ### ich brenne, bitte warten ###',cr,0
                                     0
0782 004897 202a2a2a
0782 0048b8 0d00
0783 0048ba 2a2a2a20 mes15      .byt '### eprom ist schadhaft ###',cr,00
0783 0048d5 0d00
0784 0048d7 0d        mes16     .byt cr,'50 ms brennen = cr, intelligent = i',cr
                                     ,0
0784 0048d8 3530204d
0784 0048fb 0d00
0785 0048fd 20204641 mes17      .byt '  fatale fehler, jetzt schlusspruefung:',c
                                     r,00
0785 004924 0d00
0786 004926 0d        mes18     .byt cr,'welcher programmieralgorithmus?',cr
```

```
0786 004927 57454c43
0786 004946 0d
0787 004947 42495320            .byt 'bis zu 15 x 1 ms anfangsimpulse + 4fache m
                                     enge overburn oder',cr
0787 004983 0d
0788 004984 42495320            .byt 'bis zu 25 x 1 ms "              + 3fache "
                                                       "        ?',cr
0788 0049bd 0d
0789 0049be 42495454            .byt 'bitte quittieren mit 4 oder mit 3',cr,00
0789 0049df 0d00
0790 0049e1 0d          mes19   .byt cr,'in welcher bank steht der code?',cr,00
0790 0049e2 494e2057
0790 004a01 0d00
0791 004a03 20202053    mes20   .byt '   schwache Zellen im prom',cr,00
0791 004a1d 0d00
0793                            ;wortadressen der textbeginne
0794 004a1f             mestaf          ;message-tafel
0795 004a1f a1450646            .wor mes1,mes2,mes3,mes4,mes5,mes6
0795 004a23 e9468247
0795 004a27 d247e547
0796 004a2b 0e482048            .wor mes7,mes8,mes9,mes10,mes11,mes12,mes13,mes1
                                     4,mes15
0796 004a2f 33484448
0796 004a33 56486c48
0796 004a37 7c489648
0796 004a3b ba48
0797 004a3d d748fd48            .wor mes16,mes17,mes18,mes19,mes20
0797 004a41 2649e149
0797 004a45 034a
0799 004a47 454c5256 befehle    .byt 'elrvcpaq'     erlaubte befehlstasten
0801 004a4f 5d403541 beftab     .wor einlesen,listen,ramlist,verify,checken,prog
                                     ram
0801 004a53 ec405940
0801 004a57 3941b741
0802 004a5b 4f405e45            .wor abweich,monitor adressen der teilprogramme
0803 004a5f             wor             abweich,monitor adressen der
                                        teilprogramme
0803 004a5f                             wor abweich,monitor adressen
                                        der teilprogramme
```



Beschaltung des Eprommers

Userport CBM 6xx/7xx

Pin	Signal
1	GND
3	GND
7	PB7
8	PB6
9	PB5
10	PB4
11	PB3
12	PB2
13	PB1
14	PB0
15	PA7
16	PA6
17	PA5
18	PA4
19	PA3
20	PA2
21	PA1
22	PA0
24	+5 V DC

Assembler-Praxis

auf Atari ST

Ende September soll nun endlich das Buch mit obigem Titel aus der Feder des Berichterstatters lieferbar sein (siehe auch Anzeige des te-wi Verlages in diesem Heft). Die im Buch abgedruckten Quelltexte können schon jetzt auf Diskette hier beim Autor bezogen werden (bitte Brief mit Scheck DM 42 einsenden oder auf die Überweisung schreiben: Bestellung Atari-Sources). Zur Orientierung der Leser möchte ich die enthaltenen Abschnitte nennen und auf die Zielsetzung des Buches hinweisen:

Abschnitte: Merkmale der CPU, Die Exceptions, Assembler und Syntax, Vom Text zum Programm, Leistungen eines Textprogrammes, Die typische Befehlszeile. Unter der Hauptüberschrift 'Das Sprachreservoir' folgen: Die Befehlsgruppen, Datenübertragungen, Spezialbefehle für Adreßregister, Statusbefehle, Status prüfen und merken, Binäre Rechenarten, Dezimale Rechenarten, Logische Befehle, Vergleiche, Befehle zur Bitmanipulation und Bitprüfung, Schiebe- und Rotierbefehle, Sprünge, Unterprogramme und numerierte Traps, Bedingte Verzweigungen und Schleifenkonstrukte, Stackverwaltung, Adressierungsarten - die effektive Adresse, Die wichtigen Direktiven, Sonstige Direktiven.

Im Hauptabschnitt 'Nun wird programmiert' folgen: Das Betriebssystem, Ein erstes Programm, Die BIOS-Funktionen, Eine BIOS-Toolbox, Ein GEMDOS-Toolkit, Das erweiterte XBIOS, Benutzung des Toolkits. Der Hauptabschnitt 'Mit Texten arbeiten' enthält: Fallentscheidungen, Filter, Ein Editor, Ein Parser, Strings suchen. Bei 'Spezielle Ein- und Ausgaben' findet man: Tastatur-Dekodierung, Memory Dump, Floppy-Test, Floppy-Funktionen, Mit Dateien arbeiten, Serielle Datenübertragung. Der letzte Hauptabschnitt 'Rechenroutinen und Datenfelder' enthält: Binäres Rechnen, Dezimales Rechnen, Zahllenwandlung und Arrays.

Ziel des Buches ist es, speziell dem Besitzer eines Atari ST ein sicheres Fundament für die Benutzung der Assemblersprache und für die TOS-Programmierung zu schaffen. Der Schwerpunkt der Darstellungen liegt in der Software. Der Hauptabschnitt mit den Befehlsgruppen, ordnenden Tabellen und Hinweisen auf Besonderheiten der Befehle dient dabei der Vorbereitung und dem späteren Nachschlagen. Die weiteren Abschnitte bauen Schritt um Schritt Routinen für die Benutzung des Betriebssystems auf, die man als Bibliotheksroutinen anlegen sollte. Auf ihnen aufbauend werden vollständige einfache Programme entwickelt, mit denen man interaktiv die Leistungen der Assemblerprogrammierung überprüfen kann, so bei den Rechenfunktionen, den Anfängen eines Editors und den Anfängen einer Datenbank im Abschnitt Arrays. Es kam dabei auch auf eine reiche Kommentierung und die didaktische Vorführung der möglichen Adressierungsarten an.

Es gibt natürlich bereits mehrere Bücher zum 68000 und auch zum Atari ST. In der Sicht des Autors gehen sie zunächst zu sehr auf die Hardware oder auf die Katalogisierung des Befehlssatzes ein. Wenn sie Vorschläge zur Programmierung bringen, dann sind es oft nur exemplarische Anwendungen weniger gerade betrachteter Befehle. Dieses Buch ist insofern anders, als es einerseits am konkreten Computer eine Basis für Experimente schafft und andererseits alltäglich vorkommende Aufgabestellungen anfaßt, wie Texteingabe, Suchen in Texten, Rechenroutinen mit erhöhter Genauigkeit, Eingabe von Datensätzen und Suchen in diesen.

Die Beobachtung zeigt, daß die großen Programme für das Publikum wohl vorwiegend in der Sprache 'C' und unter Einbeziehung der Benutzeroberfläche GEM geschrieben werden. Sie schleppen damit oft Ballast mit sich, der die Reaktionsgeschwindigkeit des Systemes unter diejenige unter z.B. 6502 (in Assemblerprogrammierung) herabsetzt. Assemblerprogrammierung für den 68000 ist mit der Leistungsfähigkeit der CPU schön und effektiv. Sie ist notwendig, um die heute gegebene Preiswürdigkeit der Hardware in eine akzeptable Umgebung zu führen. Das Buch möchte dafür Basis und Anregung sein.

Martin Albrecht, 4350 Recklinghausen

Patches

für die C-Bibliothek des Atari ST

Wie alle größeren Programmpakete, so sind auch die C-Bibliotheken GEMLIB und VDIBIND des Atari-Entwicklungspaketes nicht frei von Fehlern. Die hier gezeigten Korrekturen beheben zwei Probleme, die beim Rechnen mit vorzeichenbehafteten 32-Bit Zahlen (signed long) auftreten.

Hat man mit dem Linker die Bibliothek VDIBIND eingebunden, dann erzeugt der C-Operator % für den Divisionsrest unerklärliche Resultate. In VDIBIND ist das Modul lrem.o zuständig. Bild 1 zeigt ein Listing der Routine und die erforderliche Korrektur. Der Fehler ist offensichtlich. Statt des Inhalts von –ldivr (Divisionsrest) wird immer dessen Adresse als Konstante übergeben.

Das zweite Problem ist etwas komplizierter. Es betrifft das Modul ldiv der Bibliotheken VDIBIND und GEMLIB. ldiv ist für die (signed) long-Division zuständig (C-Operator /), kann aber die negative Zahl $80000000 nicht dividieren. Das Ergebnis ist immer gleich Null. Bild 2 zeigt das Listing von ldiv mit den Verbesserungen. Die Routine arbeitet intern mit positiven Zahlen. Wenn Dividend oder Divisor negativ sind (oberstes Bit=1), dann wird zuerst das Zweierkomplement gebildet, um die jeweilige Zahl positiv zu machen. Das klappt bei allen negativen Zahlen bis auf $80000000, denn der Befehl neg.l #$80000000 ergibt wieder $80000000!. Zur Abhilfe werden einige Verzweigungen auf 'unsigned'-Abfragen umgestellt.

Wie wird nun gepatcht? Alle Korrekturen ersetzen glücklicherweise die alten Befehle 1:1. Das Vorgehen ist für alle Module gleich und wird hier nur am Beispiel lrem.o gezeigt: Man benötigt das Archivprogramm AR68.PRG und den Debugger SID.PRG. AR68 muß vorher auf .TTP umbenannt werden. Mit ar68 xv vdibind lrem.o legt man eine Kopie des Moduls lrem.o an. Die eigentliche Änderung wird mit dem Debugger SID erledigt, wie in Bild 3 gezeigt wird. Das verbesserte Modul muß nun das alte lrem.o in VDIBIND ersetzen. Das erreicht man mit dem Befehl ar68 rv vdibind lrem.o. In gleicher weise ist mit ldiv.o in VDIBIND zund GEMLIB zu verfahren. - Natürlich sollte man die Änderungen nicht gleich am Original vornehmen und möglichst gründlich überprüfen.

```
* Divisionsrest zweier long-Zahlen. Entspricht der Routine lrem.o in der
* 'C'-Bibliothek VDIBIND. Korrektur angegeben

        .globl  lrem,_lrem    * lrem,_lrem sind global

        .text
lrem:
_lrem:  link    a6,#-2        * lokalen Stack einrichten
        move.l  $c(a6),-(a7)  * Divisor auf den Stack fuer ldiv
        move.l  $8(a6),-(a7)  * Dividend auf den Stack fuer ldiv
        jsr     ldiv          * Division
        cmpa.l  (a7)+,(a7)+
        move.l  #_ldivr,d0    * Divisionsrest in d0
* ersetze move.l #_ldivr,d0 durch move.l _ldivr,d0
* Code alt: 20 3C 00 00 00 00   Code neu: 20 39 00 00 00 00
        unlk    a6            * Stack bereinigen
        rts

        .end
```

Bild 1: Die Routine lrem.o

```
* Division zweier long-Zahlen. Entspricht der Routine ldiv.o in den
* 'C'-Bibliotheken GEMLIB und VDIBIND.
* Die Routine kann in dieser Form die Zahl $80000000 nicht dividieren,
* das Ergebnis ist immer gleich 0. Korrekturen in den Kommentaren.

      .globl  ldiv,_ldiv,_ldivr   * ldiv,_ldiv und _ldivr sind global

      .text
```

```
ldiv:
_ldivs: link    a6,#-2          * lokalen Stack einrichten
        movem.l d2-d7,-(a7)
        clr.w   d3              * Vorzeichen Flag
        clr.l   d5              * Quotient = 0
        move.l  $8(a6),d7       * Dividend in d7
        move.l  $c(a6),d6       * Divisor in d6
        bne.s   ld1
        move.l  #$80000000,_ldivr   * Divisor = 0
        move.l  ##80000000,d0
        divs    #0,d0           * Division durch 0, Bomben!
        bra     ld11            * fertig
ld1:    bge.s   ld2             * Divisor >= 0?
        neg.l   d6              * ist negativ, Zweierkomplement bilden
        addq.w  #$1,d3          * in d3 vermerken
ld2:    tst.l   d7              * Dividend >= 0?
        bge.s   ld3             * ja
        neg.l   d7              * ist negativ, Zweierkomplement bilden
        addq.w  #$1,d3          * d3+1, Quotient ist neg. wenn Bit0=1 !
ld3:    cmp.l   d7,d6
        bgt.s   ld9             * Divisor > Dividend, Ergebnis = 0!
* ersetze bgt.s ld9 durch bhi.s ld9, verzeigt dann auch bei d6=$80000000
* bgt arbeitet signed (Zweierkomplement), bhi arbeitet unsigned!
* Code alt: 6E 38  Code neu: 62 38
        bne.s   ld4             * Divisor <> Dividend, weiter!
        moveq.l #$1,d5          * Divisor = Dividend, Ergebnis = 1!
        clr.l   d7              * Divisionsrest auf 0
        bra.s   ld9             * fertig
ld4:    cmp.l   #$10000,d7  * 16-Bit Divisor ?
* ersetze cmp.l #$10000,d7 durch cmp.l #$ffff,d7, kann dadurch folgendes
* bge durch bhi ersetzen (Gruende siehe oben)
* Code alt: BE BC 00 01 00 00  Code neu: BE BC 00 00 FF FF
        bge.s   ld5             * nein
* ersetze bge.s ld5 durch bhi.s ld5 (Gruende siehe oben)
* Code alt: 6C 0A  Code neu: 62 0A
        divu    d6,d7           * ja, nutze divu-Befehl
        move.w  d7,d5           * Quotient in d5
        swap    d7              * Divisionsrest in d7
        ext.l   d7              * Vorzeichen auf 32-Bit erweitern
        bra.s   ld9             * fertig
ld5:    moveq.l #$1,d4          * unterstes Bit in d4 setzen
ld6:    cmp.l   d6,d7
        bcs.s   ld7
* ersetze bcs.s ld7 durch bls.s ld7, damit der Divisor bei Dividend
* $80000000 nicht zu weit nach links geschoben wird
* Code alt: 65 06  Code neu: 63 06
        asl.l   #1,d6           * Disisor linksschieben, bis > Dividend
        asl.l   #1,d4           * d4 ebenso, für Resultatbit
        bra.s   ld6             * Schleife
ld7:    tst.l   d4
        beq.s   ld9             * Resultatbit verschwunden, fertig
        cmp.l   d6,d7
        bcs.s   ld8             * d6 > d7, vor Subtraktion erst rechtsschieben
        or.l    d4,d5           * Resultatbit in d5 'odern'
        sub.l   d6,d7           * Divisor abziehen
ld8:    lsr.l   #1,d4           * und Divisor und Resultatbit rechtsschieben
        lsr.l   #1,d6
        bra.s   ld7             * bis d4=Resultatbit verschwunden
ld9:    cmp.w   #$1,d3          * au8 Resultat negativ sein ? (teste Bit0)
        bne.s   ld10            * nein
        neg.l   d7              * ja, Zweierkomplement des Rests
        move.l  d7,_ldivr       * Divisionsrest ablegen
        move.l  d5,d0           * Quotient in d0
        neg.l   d0              * auch Zweierkomplement bilden
        bra.s   ld11
ld10:   move.l  d7,_ldivr       * Divisionsrest ablegen
        move.l  d5,d0           * Quotient in d0
ld11:   tst.l   (a7)+           * Stack bereinigen
        movem.l (a7)+,d3-d7
        unlk    a6
        rts

        .bss
        .even
```

```
_ldivr: ds.l    1          * Divisionsrest (global)

        .end
```

Bild 2: Die Routine ldiv.o

```
-r lrem.o<Return>                         (lrem einlesen)
Start = 000D408E     End = 000D41DD
-l d40aa<Return>                          (ab Start+$1C disassemblieren)
  000D40AA      link  A6,#$FFFE
  000D40AE      move.l  $C(A6),-(A7)
  000D40B2      move.l  $8(A6),-(A7)
  000D40B6      jsr  $0
  000D40BC      cmpm.l  (A7)+,(A7)+
  000D40BE      move.l  #$0,D0
  000D40C4      unlk  A6
  000D40C6      rts
  000D40C8      subq.w  #$7,$6469(a4)
  000D40CC      moveq.l  #$72,D3
  000D40CE      ori.b  #$8800,D0
  000D40D2      ori.b  #$0,D0
-s d40be<Return>                          (Speicher ändern mit -s)
000D40BE 20 <Return>
000D40BF 3C 39<Return>                    (D40BF zu 39 ändern)
000D40C0 00 .<Return>                     (-s beenden (.<Return>))
-w lrem.o                                 (geändertes lrem.o zurückschreiben)
-<Control C>Bye                           (SID verlassen mit Control C)
```

Bild 3: SID-Protokoll, fett: Benutzereingabe, kursiv: Kommentare

□○□

Editorial

Am Ende dieses Heftes ist der Platz für begleitende Bemerkungen knapp geworden. Daher in Kürze: Bei der Fa. Völkner läuft noch immer die Ausverkaufsaktion für Commodore-Rechner der Typen 610/710/720. Viele Firmen und Institute haben da zugegriffen, um ein preiswertes Entwicklungssystem betreiben zu können oder um die Rechner einer speziellen Aufgabe zuzuteilen. Der Eprommer in diesem Heft ist ein weiterer Beitrag für diese Verwendung. Einen Abzug zum RAM/ROM/I-O-Listing des CBM 710 kann man erhalten bei: Roland Prinz, Adalber-Stifter-Str. 2, 6930 Eberbach, Tel. 06271-2830. Beiträge aus der Leserschaft zu diesen Rechnern sind weiterhin willkommen.

Bei der Entwicklung des Editors für den PC-128 zeigte sich, daß man mit diesem Gerät besser noch als mit dem CBM 720 in Assembler umgehen kann, der Durchgriff in andere Bänke ist einfacher. Und man hat viel Platz für eigene Anwendungen. Bei künftigen Entwicklungen für 65xxx sollte man lieber den PC-128 als den AIM 65 nehmen, der an vielen Stellen schon stillgelegt worden ist.

Atari-Rechner der Serie ST sind ein großer Erfolg geworden, nicht zuletzt wegen ihres Preises. Wer in diesem Heft die Beiträge zum Atari und zur Programmierung des 68000 studiert, wird bemerken, daß man mit ihnen auf eine sehr einfache Weise umgehen kann. Aus der eigenen Erfahrung möchte ich hinzufügen, daß man auf dem 68000 mehr Code je Arbeitssitzung schreibt, als z.B. auf dem 6502, weil man eine Zweiadreß-Maschine mit einem sehr leistungsfähigen Befehlssatz programmiert. - Der AMIGA, auch mit 68000, ist im Betriebssystem noch wieder anders konzipiert, weil mehrere Tasks zugleich abgearbeitet werden können. Im nächsten Heft (Anfang November) findet man weitere Beiträge zu 68000 und auch zum AMIGA. Erwähnen Sie bitte in Ihrem Bekanntenkreis die thematische Erweiterung des MICRO MAG. Und schreiben Sie gerne auch einen Beitrag zu eigenen Arbeiten.

Bücher

Thies, K.-D.: Das 8086 Systembuch, te-wi Verlag, München 1986, 208 S., DM 49,-, ISBN 3-921803-61-6. Realisierungen von Multiprozessorsystemen mit der CPU 8086 werden konkret und für die Ansprüche von Entwicklern beschrieben. Teil 1 des reich bebilderten Bandes beschreibt das I/O-Interfacing, Teil 2 das Multiprocessing mit dem Arbiter 8289. Wer für diese CPU oder für 86186/86286 entwickelt, wird auf dieses Buch zurückgreifen.

Lien, David. A.: MACINTOSH Programmier-Handbuch: MS BASIC 2.1, te-wi Verlag, München 1986, 450 S., DM 59,-, ISBN 3-921803-48-9. Übersetzung eines Kurstextes aus dem Amerikanischen (Lien ist Professor in San Diego). Das lebendig geschriebene Buch fängt bei den einfachsten Dingen des BASIC an und reicht bis zum Gebrauch der mehr abstrakten Funktionen. Wer sich auf dem MAC mit kaum Vorkenntnissen einarbeiten oder die vollen Möglichkeiten des Befehlssatzes ausschöpfen will, der sollte zu diesem Buch greifen.

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

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

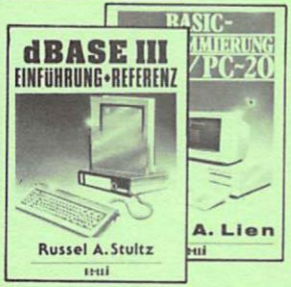
DIE ASM86/ASM286 MAKROASSEMBLER
Thies, 300 Seiten
Assemblerprogrammierung von 8086, 8087, 80186, 80286: ASM-Konzepte wie Segmente, Module, Prozeduren usw. werden über CPU/Speicherbilder funktionell erklärt.
Softcover, DM 69,– (Mitte 86)

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

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

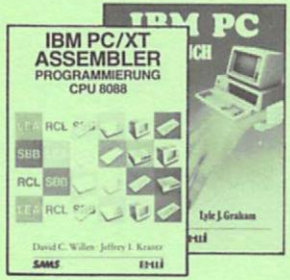
UNIX
(Yates/Thomas) US-Standardwerk der UNIX-Promoterin Yates. Eine sachkundige Übersicht und Einführung in die Anwendung. 550 Seiten, Softcover, DM 79,–

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



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

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



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

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

tm 4582

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

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

In Vorbereitung:
Das 80286 Buch

MICRO MAG

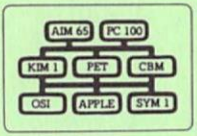
HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☎ (04102) 5 58 16

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

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

Leser-Service des Herausgebers

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM

Jetzt Preisvergünstigung bei Nachlieferungen,
solange der Vorrat reicht:

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

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

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

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