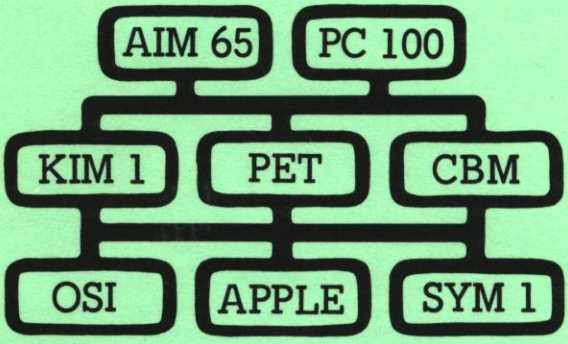
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 9,50 Nr. 31 Juni 1983



Inhaltsverzeichnis

EXEDIT - Extended Editor für AIM 65 3
Erweiterung des EXEDIT 18
AIM Spezial (14) 22
FORTH (7) 25
FORTH Mnemonics? 28
Adreßkartei unter FORTH 31
FORTH-Disassembler 35
INPUT Window (CBM) 42
32 KB CMOS-RAM 46
Der 68000 VMEbus 49
Supertape - Kassettenaufzeichnung mit 600 Byte/Sek. 52
Bücher 58
Aus der Branche 59
Editorial 61

DUO Plott Interface

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

Paralleles IEEE 488 - Interface, genormter IEEE-Stecker und Kabel
kompletter Zeichensatz des CBM-Computers
zwei Geräteadressen für Groß-/Grafikmodus und Textmodus
alle Funktionen der Drucker bleiben erhalten, Floppy-Kompatibilität
Deutsche Umlaute, ß und Paragraph
Problemloser Einbau, inklusive deutsches EPSON-Handbuch
Komplettpreis einschl. Kabel, CBM-Grafiksatz und Handbuch incl. MWSt
für EPSON DM 298,--
für ITOH DM 298,--

Umrüstsatz MX80 F/T auf MX80 Typ 3

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

Für COMMODORE

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

Speichererweiterung auf 96 K

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

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

Drucker:

RX 80 DM 998,-- + MWSt

ITOH 8510 DM 1495,-- + MWSt

EPSON-Druck-Computer FX 80 DM 1495,-- + MWSt

Stellberg Computer-Systeme

COMMODORE EPSON C. ITOH SOFTWARE INTERFACE

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

Dipl.-Phys. A. Redder und Dipl.-Ing. R. Wollenberg

EXEDIT - Extended Editor für AIM 65

1. Einleitung

Auf der Grundlage des geänderten Betriebssystems REMON /1/ wurde eine Befehlserweiterung für den AIM-Editor entwickelt. Es sollten sowohl die Möglichkeit, das Display ein- und auszuschalten, als auch die Einrichtung einer CAPS-LOCK-Taste auf der Editorebene durch entsprechende Kommandos unterstützt werden. Zum anderen wurden Routinen entwickelt, die einen Block von Zeilen löschen, kopieren oder verschieben. Aber die wohl mützlichste Routine ist die XCHANG-Routine, welche einen eingegebenen String im gesamten Textspeicher durch einen anderen ersetzt. Der in Heft 30 des 65xx MICRO MAG vorgestellte Screen-Editor /2/ wird durch einen entsprechenden Befehlsaufruf angebunden.

Durch diese Implementation von neuen Befehlen wird die Leistungsfähigkeit des AIM-Editors erheblich gesteigert, und es wurde ein weiterer Schritt auf dem Weg zu einem komfortablen Textverabeitungssystem getan. Es lassen sich bestimmt noch weitere nützliche Utilities finden, aber mit den hier vorgestellten Routinen ist der Hauptteil der nötigen Software für die Textherstellung nun vorhanden. Was noch fehlt, ist ein Wordprozessor, der die erstellten Texte im Blocksatz formschön auf einen Drucker oder eine Schreibmaschine ausgibt.

2. Programmaufbau

Der nachstehende Quelltext kann in vier Bereiche eingeteilt werden. Diese sind:

a) Initialisierung,
b) Befehlsdekodierung,
c) Befehlsroutinen und
d) Unterprogramme.

a) Initialisierung

Um das Programm an den vorhandenen Editor anzuhängen, ist es notwendig, den EDLINK-Vektor ($A40C, A40D) auf den Anfang des Programms zu richten. An dieser Stelle werden auch die Startwerte für die Tabulatorfunktion gesetzt, die im Screen-Editor benötigt werden. Es wird die Kontrollmeldung

' Extended Editor V2.3 '

ausgegeben, um zu signalisieren, daß jetzt der erweiterte Befehlssatz zur Verfügung steht. Anschließend wird automatisch der normale AIM-Editor aufgerufen.

b) Befehlsdekodierung

Die Dekodierung gültiger Befehle geschieht nach dem gleichen Prinzip, wie es auch im AIM-Editor praktiziert wird. Weitere Befehle werden angehängt, indem COMCNT (=Anzahl der Befehle- 1) richtig gesetzt wird, in COMTBL das gültige ASCII-Zeichen für den neuen Befehl und in JMPTBL die hexadezimale Startadresse oder das Label angefügt wird.

c) Befehlsroutinen

Insgesamt sind, wenn man den Screeneditor mitzählt, acht neue Befehle implementiert. Nachfolgend eine Übersicht über die neuen Kommandos.

A – Die Routine DSONOF schaltet das Display an oder aus, vgl. /1/.

Z – In CAPLCK wird die Umschaltung der Groß-/Kleinschreibung vorgenommen /1/.

E – Es wird ein Block von Zeilen gelöscht. Die Bereichsgrenzen werden über eine Positionierung angegeben. Diese erfolgt über die Up- und Down-Kommandos.

Y – Ein Block von Zeilen wird vor eine beliebige Zeile kopiert. Der Originalblock bleibt erhalten.

M -- Ein Zeilenblock wird von einer Stelle an eine andere transportiert.

X – Die XCHANG-Routine ist eine Verbesserung des C-Kommandos. Es wird im gesamten Textbereich ein Old-String in einen New-String umgewandelt. Dies geschieht auch mehrfach in einer Zeile, wenn der Old-String mehrfach vorkommt.

W – Komprimiert einen Assembler-Quelltext, indem alle Kommentare gelöscht werden. Ausnahmen:

- Die erste Textzeile bleibt unverändert.
- Geht einem Semikolon ein ' voran, wird dort keine Löschung vorgenommen, um z.B. `CMP #';'` zuzulassen.

Rechts stehende Blanks werden zusätzlich gelöscht, ebenso wie reine Kommentarzeilen.

S – Aufruf des Screen-Editors /2/.

Es gibt drei Programme, die auf einen Block von Zeilen wirken, und zwar die Routinen ERASE, COPY und MOVE. Ihnen ist gemeinsam, daß man den Bereich durch eine FROM- und TO-Zeile markieren muß. Und zwar wird, nachdem die Aufforderung 'FROM ?', 'TO ?' bzw. 'DESTIN. ?' erschienen ist, eine Befehlserkennungsroutine durchlaufen, die folgende Möglichkeiten zuläßt:

– CR die Current-Line wird als FROM- oder TO-Line übernommen.

-- ESC die Routine wird abgebrochen, ohne daß der Editor verlassen wird.

-- U, D, T, B, Space
Es wird das entsprechende Kommando des AIM-Editors ausgeführt. Sie dienen zur Positionierung auf eine bestimmte Zeile.

Bei COPY und MOVE wird zusätzlich noch mit 'DESTIN. ?' nach einer Zeile gefragt, vor die der Block eingefügt werden soll. Danach erfolgt zur Sicherung die Abfrage 'SHURE (Y/N)?. Die Routine wird nur abgearbeitet, wenn die Frage mit 'Y' beantwortet wurde.

Die XCHANG-Routine verlangt zuerst die Eingabe des zu ändernden Strings und nach Ausgabe von 'TO=' den neuen String. Wird der Old-String gefunden, so wird die entsprechende Zeile angezeigt. Man hat nun die Möglichkeit, durch Eingabe von CR den Austausch durchführen zu lassen oder durch Betätigen der Space-Taste die Suche nach dem nächsten Auftreten von Old-String zu starten. Bei Eingabe von ESC wird die Routine abgebrochen. Falls Old-String in einer Zeile mehrfach vorkommt, wird die Zeile für jedes Auftreten angezeigt und kann verändert werden. Dies ist unabdingbar, wenn man Textverarbeitung betreiben will. Ein Nachteil ist nur, daß kein Cursor die Position des Strings markiert, der als nächster geändert werden soll.

Bei Anbindung des Screeneditors ist zu beachten, daß einige Subroutinen in beiden Programmen vorkommen und daher an einer Stelle gelöscht werden können.

d) Unterprogramme

Viele Funktionen sind, obwohl sie nur einmal gebraucht werden, als Unterprogramme ausgeführt, um sowohl die Lesbarkeit der Befehlsroutinen zu erhöhen, als auch eine mögliche spätere Erweiterung zu vereinfachen.

3. Literaturhinweise

/1/ Wollenberg, Redder: REMON - Redigierter Monitor, 65xx MICRO MAG, Nr. 28, Dez. 1982
/2/ Wollenberg, Redder: SCREEN – Bildschirmeditor für AIM 65, MICRO MAG 30, April 1983
/3/ Rockwell International: AIM 65 - Monitor-Programmlisting

```
0000          ;*********************************************************
0000          ;*
0000          ;*    EXED-Extended Editor  V2.3  20.4.83  REDDER/WOLLENBERG
0000          ;*
0000          ;*********************************************************
0000
0000
0000          ;*********************************************************
0000          ;*  Definition of Editor and Monitor-Routines
0000          ;*********************************************************
0000
0000 OUTPUT         =$E97A          ;Outputs ASCII in A to D/P
0000 QM             =$E7D4          ;Outputs A '?'
0000 CRLF           =$E9F0          ;Outputs CR,LF & NUL to AOD
0000 ADDA           =$F92A          ;Add acc. to NOWLN
0000 SAVNOW         =$F934          ;Save current-line to ADDR
0000 ATTOP          =$F8D8          ;Current-line at top ? C-flag set if so
0000 DOW1           =$F6E3          ;1 line down in memory
0000 ATBOT          =$F8E9          ;Current-line at bottom ?
0000 ATEND          =$F8F9          ;Current-line at end of text ?
0000 AD1            =$F928          ;Add 1 to NOWLN
0000 UPNO           =$F709          ;1 line up in memory with bottom-check
0000 UP1            =$F713          ;1 line up in memory
0000 RESNOW         =$F8D0          ;Restore current-line from ADDR
0000 REPLAC         =$F93F          ;Replaces new line for old one
0000 TOPNO          =$F8BC          ;Set current-line to top
0000 SETBOT         =$F8C5          ;Set current-line to bottom
0000 SUB            =$F91D          ;Substract 1 from NOWLN
0000 REENTR         =$F6CF          ;Reentry of AIM-editor
0000 CFLG           =$F882          ;Set flag for C-command
0000 KIFLG          =$F8B6          ;Clear flag for K- or I-command
0000 FCH            =$F81E          ;Get string and find it
0000 FC8            =$F85A          ;Find string
0000 GETKEY         =$EC40          ;Get 1 char. from keyboard
0000 RDRUB          =$E95F          ;Get 1 char. with delete possible
0000 CLR            =$E844          ;Clears D/P pointers
0000 PLNE           =$F727          ;Displays current-line
0000 EDIT           =$F641          ;Start of editor
0000 KEP            =$E7AF          ;Outputs system-messages
0000 BRK4           =$E6FA          ;Outputs "ON" to D/P
0000 BRK3           =$E6F1          ;Outputs "OFF" to D/P
0000
0000 TABMAX         =8              ;Number of tabulators
0000 CR             =$0D            ;Carriage-return
0000 ESC            =$1B            ;Escape
0000
0000
0000          ;*********************************************************
0000          ;*    Monitorvariablen
0000          ;*********************************************************
0000
0000 JUMP           =$A47D          ;Used for indirekt JUMP
0000 STIY           =$A427          ;Used by FCH
0000 NAME           =$A42E          ;Dummy variable used by COPY & ERASE
0000 CURPOZ         =$A415          ;Curserposition
0000 EDLINK         =$A40C          ;Link-vector for editor (MIKRO MAG 28)
0000 ADDR           =$A41C          ;Used by SAVNOW & RESNOW
```

```
0000 DIBUFF         =$A438          ;Displaybuffer
0000 FLGREG         =$A408          ;Flag-register (see MIKRO MAG 28)
0000
0000
0000          ;*************************************************************
0000          ;*   Zero-page-variablen
0000          ;*************************************************************
0000
0000                *=$B6
00B6 PUFFER         *=*+25          ;New string (used by XCHANG)
00CF SAVU1          *=*+2           ;Dummy variable
00D1 TABNR          *=*+1           ;Number of current tabulators
00D2 TABBUF         *=*+8           ;Space for tabulator-values
00DA TVLIN          *=*+1           ;Current TV-line
00DB CURADR         *=*+2           ;Absolute curser-address
00DD CURREL         *=*+1           ;Relative curser-address
00DE IFLG           *=*+1           ;Insert-flag
00DF                *=$DF
00DF NOWLN          *=*+2           ;Current-line
00E1 BOTLN          *=*+2           ;Last active, so far
00E3 TEXT           *=*+2           ;Start of text
00E5 END            *=*+2           ;End of text
00E7 SAVE           *=*+2           ;Used by REPLAC
00E9 OLDLEN         *=*+1           ;Original length
00EA LENGTH         *=*+1           ;New length
00EB STRING         *=*+20          ;Old string used by C-command & XCHAN
00FF
00FF          ;*************************************************************
00FF          ;*     Start of EXEDIT (Initialization)
00FF          ;*************************************************************
00FF
00FF                *=$C000
C000
C000 EDINI  A927    LDA #(EXEDIT    ;Let EDLINK point to beginning of
C002        8D0CA4  STA EDLINK      ; the program
C005        A9C0    LDA #)EXEDIT
C007        8D0DA4  STA EDLINK+1
C00A        A008    LDY #TABMAX     ;Initialization of the tabulator
C00C        84D1    STY TABNR
C00E        88      DEY
C00F TABZE1 B91FC0  LDA TABPOS,Y
C012        99D200  STA TABBUF,Y
C015        88      DEY
C016        10F7    BPL TABZE1
C018        C8      INY
C019        20DAC3  JSR KEPU        ;Output message
C01C        4C41F6  JMP EDIT        ;Jump to start of editor
C01F
C01F TABPOS 00      .BYT 0,7,11,22,29,39,49,59
C020        07
C021        0B
C022        16
C023        1D
C024        27
C025        31
C026        3B
C027
```

```
C027        ;************************************************************
C027        ;* Command-Interpreter-Routine            REGS. ALT.: A,X,Y
C027        ;*
C027        ;* ( see also monitor-listing )
C027        ;************************************************************
C027
C027 EXEDIT A207    LDX #COMCNT     ;Set value for loop
C029 CD02   DD49C0  CMP COMTBL,X    ;Compare A with valid ASCII-codes
C02C        F006    BEQ CFND1       ;Valid; branch
C02E        CA      DEX             ;Dekrement loop-counter
C02F        10F8    BPL CD02        ;ready ? no,branch
C031        4CD4E7  JMP QM          ;Unvalid ASCII-character
C034 CFND1  20F0E9  JSR CRLF        ;Valid command
C037        8A      TXA             ;Decode it
C038        0A      ASL A
C039        AA      TAX
C03A        BD51C0  LDA JMPTBL,X    ;Get address of subroutine
C03D        8D7DA4  STA JUMP
C040        BD52C0  LDA JMPTBL+1,X
C043        8D7EA4  STA JUMP+1
C046        6C7DA4  JMP (JUMP)      ;Execute it
C049
C049 COMCNT         =7              ;Number of impl. commands-1
C049
C049
C049 COMTBL 415A    .BYT 'AZEYMXWS'  ;Command-table
C051
C051 JMPTBL 61C0    .WOR CAPLCK,DSONOF,ERAS,COPY
C053        6EC0
C055        77C0
C057        BDC0
C059        00C2    .WOR MOVE,XCHANG,WRING,SCREEN
C05B        85C1
C05D        1CC2
C05F        30C4
C061
C061
C061        ;************************************************************
C061        ;*   Capital letters ON/OFF               REGS. ALT.: A,X,Y
C061        ;************************************************************
C061
C061 CAPLCK 20ABC2  JSR GROSKL      ;Switch caps-lock-flag
C064        2940    AND #$40
C066        F003    BEQ *+5
C068        4CF1E6  JMP BRK3        ;Output "OFF"
C06B        4CFAE6  JMP BRK4        ;Output "ON"
C06E
C06E
C06E        ;************************************************************
C06E        ;*    DISPLAY ON/OFF                    REGS. ALT.: A
C06E        ;************************************************************
C06E
C06E DSONOF AD08A4  LDA FLGREG      ;Switch display-flag ( OFF/ON )
C071        4980    EOR #$80
C073        8D08A4  STA FLGREG
C076        60      RTS
C077
C077
```

```
C077          ;*   Clear from line 'FROM' to line 'TO'   REGS. ALT.: A,X,Y
C077          ;*************************************************************
C077
C077 ERAS     20C6C2 JSR FROMTO       ;Get range 'FROM' to 'TO'
C07A          20F0E9 JSR CRLF
C07D          A01F   LDY #MES4-MES1
C07F          20DAC3 JSR KEPU         ;Outputs "SHURE (Y/N) ?"
C082          2040EC JSR GETKEY       ;Get character
C085          20D3C3 JSR FILT1        ;Make capital letters
C088          C959   CMP #'Y'         ;If 'Y' then go on
C08A          D024   BNE ERAS3        ;Other key, so return
C08C          20F0E9 JSR CRLF
C08F          A000   LDY #0
C091 ERAS1    B1DD   LDA (CURREL),Y   ;Move text around
C093          91DB   STA (CURADR),Y
C095          F009   BEQ ERAS2        ;At end of text ? yes, branch
C097          2096C2 JSR INC1         ;Inkrement CURADR by 1
C09A          2090C2 JSR INC2         ;Inkrement CURREL by 1
C09D          4C91C0 JMP ERAS1        ;End of loop
C0A0 ERAS2    2092C3 JSR SBFRTO       ;Compute length ( CURREL-CURADR )
C0A3          38     SEC
C0A4          A5E1   LDA BOTLN
C0A6          E5DD   SBC CURREL       ;Compute new last active line
C0A8          85E1   STA BOTLN
C0AA          A5E2   LDA BOTLN+1
C0AC          E5DE   SBC CURREL+1
C0AE          85E2   STA BOTLN+1
C0B0 ERAS3    AD2EA4 LDA NAME         ;Set current-line to line before
C0B3          85DF   STA NOWLN        ;  'FROM'
C0B5          AD2FA4 LDA NAME+1
C0B8          85E0   STA NOWLN+1
C0BA          4C27F7 JMP PLNE         ;Display current-line
C0BD          ;*************************************************************
C0BD          ;*   Copy a block of lines to new position
C0BD          ;*                                          REGS. ALT.: A,X,Y
C0BD          ;*************************************************************
C0BD
C0BD COPY     2034F9 JSR SAVNOW
C0C0          20C6C2 JSR FROMTO       ;Get the range
C0C3 COPY1    A02C   LDY #MES5-MES1
C0C5          20DAC3 JSR KEPU         ;Outputs "DESTIN. ?"
C0C8 COPY2    2040EC JSR GETKEY       ;Get character
C0CB          C90D   CMP #CR          ;If CR, go on
C0CD          F00D   BEQ COPY3
C0CF          C91B   CMP #ESC         ;If ESC, force a reentry
C0D1          D003   BNE *+5
C0D3          4CCFF6 JMP REENTR
C0D6          201FC3 JSR COMAND       ;Up,down,top or bottom-command ?
C0D9          4CC8C0 JMP COPY2
C0DC COPY3    20A0C3 JSR COFRTO       ;Is 'FROM' less than 'TO'
C0DF          B0E2   BCS COPY1        ;No,branch to beginning
C0E1          20F0E9 JSR CRLF
C0E4          A01F   LDY #MES4-MES1
C0E6          20DAC3 JSR KEPU         ;Outputs "SHURE (Y/N) ?"
C0E9          2040EC JSR GETKEY       ;Get character
C0EC        • 20D3C3 JSR FILT1        ;Make capital letters
C0EF          C959   CMP #'Y'         ;If 'Y', go on
C0F1          F003   BEQ *+5
```

```
C0F3          4C7FC1 JMP COPY8
C0F6          20B4C2 JSR SANOWU      ;Save current-line at SAVU1
C0F9          2092C3 JSR SBFRTO      ;Compute length; CURREL=CURREL-CURADR
C0FC          2034F9 JSR SAVNOW      ;Save current-line at ADDR
C0FF          20C5F8 JSR SETBOT      ;Set current-line to last line
C102          18     CLC
C103          A5E1   LDA BOTLN       ;Compute new last line
C105          65DD   ADC CURREL      ;BOTLN=BOTLN+length
C107          85E1   STA BOTLN
C109          85DF   STA NOWLN       ;Save new last line at NOWLN too
C10B          A5E2   LDA BOTLN+1
C10D          65DE   ADC CURREL+1
C10F          85E2   STA BOTLN+1
C111          85E0   STA NOWLN+1
C113          20F9F8 JSR ATEND       ;Bottom-line at end of textbuffer ?
C116          900E   BCC COPY3A      ;No, branch
C118          20C2C3 JSR RESBOT      ;Restore old bottom-line
C11B          20F0E9 JSR CRLF
C11E          A031   LDY #$31
C120          20AFE7 JSR KEP         ;Output "MEM FAIL"
C123          4CCFF6 JMP REENTR      ;Reenter of the AIM-Editor
C126 COPY3A   2082C3 JSR COMP1       ;Is 'DEST.' greater than 'TO' ?
C129          9024   BCC COPY4       ;Yes, branch
C12B          18     CLC
C12C          A5DB   LDA CURADR      ;Compute new 'FROM' ( CURADR )
C12E          65DD   ADC CURREL      ;CURADR=CURADR+length
C130          85DB   STA CURADR
C132          8D2EA4 STA NAME        ;Save it in NAME,NAME+1 too
C135          A5DC   LDA CURADR+1
C137          65DE   ADC CURREL+1
C139          85DC   STA CURADR+1
C13B          8D2FA4 STA NAME+1
C13E          18     CLC
C13F          AD30A4 LDA NAME+2      ;Compute new 'TO' (stored in NAME+2
C142          65DD   ADC CURREL      ; ,NAME+3)
C144          8D30A4 STA NAME+2      ;NAME+2,NAME+3='TO'+length
C147          AD31A4 LDA NAME+3
C14A          65DE   ADC CURREL+1
C14C          8D31A4 STA NAME+3
C14F COPY4    A000   LDY #0          ;For indirekt LDA & STA
C151          4C5AC1 JMP COPY6
C154 COPY5    20DDF9 JSR SUB         ;Dekrement NOWLN by 1
C157          20B8C2 JSR DECSAV      ;Dekrement SAVE by 1
C15A COPY6    B1E7   LDA (SAVE),Y    ;Move text around to have space
C15C          91DF   STA (NOWLN),Y   ; for the block of lines to copy
C15E          A5E8   LDA SAVE+1
C160          C5D0   CMP SAVU1+1     ;Done ?
C162          D0F0   BNE COPY5       ;No, branch
C164          A5E7   LDA SAVE
C166          C5CF   CMP SAVU1       ;Done ?
C168          D0EA   BNE COPY5       ;No, branch
C16A COPY7    B1D3   LDA (CURADR),Y  ;Move text from 'FROM' ( CURADR )
C16C          91CF   STA (SAVU1),Y   ; to 'DEST.' ( SAU1 )
C16E          2096C2 JSR INC1        ;Inkrement CURADR by 1
C171          20A4C2 JSR INC3        ;Inkrement SAVU1 by 1
C174          2080C2 JSR DEC2        ;Dekrement CURREL (length) by 1
C177          A5DE   LDA CURREL+1    ;Has Length reached zero ?
C179          D0EF   BNE COPY7       ;No, branch
```

```
C17B          A5DD    LDA CURREL      ;Length=0 ?
C17D          D0EB    BNE COPY7       ;No, branch
C17F COPY8    20D0F8  JSR RESNOW      ;Restore current-line
C182          4C27F7  JMP PLNE        ;Display current-line
C185
C185
C185          ;*************************************************************
C185          ;*    Change old-string to new-string in the whole text
C185          ;*      buffer. It changes more than one occurences in
C185          ;*      a line.                              REGS. ALT.: A,X,Y
C185          ;*************************************************************
C185
C185 XCHANG   20B4C2  JSR SANOWU      ;Save current-line
C188          201EF8  JSR FCH         ;Get old-string and find it
C18B          AD29A4  LDA STIY+2      ;Length of old-string
C18E          85DB    STA CURADR      ;Save it
C190          AD15A4  LDA CURPO2      ;Curserposition
C193          85DD    STA CURREL      ;Save it
C195          20F0E9  JSR CRLF
C198          A005    LDY #5
C19A          20AFE7  JSR KEP         ;Output "TO="
C19D          A000    LDY #0
C19F CHANG1   205FE9  JSR RDRUB       ;Get a char. with delete possible
C1A2          C90D    CMP #CR         ;End of new-string ?
C1A4          F008    BEQ CHANG2      ;Yes, branch
C1A6          99B600  STA PUFFER,Y    ;Save new-string
C1A9          C8      INY
C1AA          C019    CPY #25         ;Allows 25 characters
C1AC          D0F1    BNE CHANG1
C1AE CHANG2   84DA    STY TVLIN       ;Save length of new-string
C1B0 CHANG3   2044EB  JSR CLR
C1B3          2027F7  JSR PLNE        ;Display current-line
C1B6          2040EC  JSR GETKEY      ;Get a key
C1B9          C90D    CMP #CR         ;If CR, change old- to new-string
C1BB          D035    BNE CHANG7      ;No CR,branch
C1BD          A5DD    LDA CURREL      ;Curserposition
C1BF          48      PHA             ;Save it
C1C0          202AF9  JSR ADDA        ;Add it to NOWLN
C1C3          A4DA    LDY TVLIN       ;Length of new-string
C1C5          84EA    STY LENGTH
C1C7          F00A    BEQ *+12        ;If LENGTH=0; no exchange
C1C9          88      DEY
C1CA CHANG4   B9B600  LDA PUFFER,Y    ;Move new-string to display buffer
C1CD          9938A4  STA DIBUFF,Y    ; for REPLAC
C1D0          88      DEY
C1D1          10F7    BPL CHANG4
C1D3          20B2F8  JSR CFLG        ;Set C-flag for REPLAC
C1D6          A5DB    LDA CURADR      ;Length of old-string
C1D8          85E9    STA OLDLEN
C1DA          203FF9  JSR REPLAC      ;Replace old-string by new-string
C1DD          68      PLA             ;Curserposition
C1DE          AA      TAX
C1DF          F006    BEQ CHANG6
C1E1 CHANG5   201DF9  JSR SUB         ;Restore NOWLN where it was
C1E4          CA      DEX
C1E5          D0FA    BNE CHANG5
C1E7 CHANG6   20CBC3  JSR FNDSTR      ;Find next old-string
```

```
C1EA          AD15A4 LDA CURPO2      ;Save curser position
C1ED          85DD   STA CURREL
C1EF          4CB0C1 JMP CHANG3
C1F2 CHANG7   C920   CMP #' '        ;If space, find next occurences
C1F4          F0F1   BEQ CHANG6      ; of old-string (could be same line)
C1F6          C91B   CMP #ESC        ;If escape, we are ready
C1F8          D0B6   BNE CHANG3
C1FA          20BDC2 JSR RSNOWU      ;Restore current-line
C1FD          4C27F7 JMP PLNE        ;Display current-line
C200
C200
C200          ;*************************************************************
C200          ;*   Move a block of lines to a new position
C200          ;*                                          REGS. ALT.: A,X,Y
C200          ;*************************************************************
C200
C200 MOVE     20BDC0 JSR COPY        ;See COPY routine
C203          AD2EA4 LDA NAME        ;Restore CURADR (FROM) and CURREL
C206          85DB   STA CURADR      ; (TO) for ERASE routine
C208          AD2FA4 LDA NAME+1
C20B          85DC   STA CURADR+1
C20D          AD30A4 LDA NAME+2
C210          85DD   STA CURREL
C212          AD31A4 LDA NAME+3
C215          85DE   STA CURREL+1
C217          A000   LDY #0
C219          4C91C0 JMP ERAS1       ;See ERASE routine
C21C
C21C
C21C          ;*************************************************************
C21C          ;*  Wring Assembler-Source-Text and delete comments
C21C          ;*                                          REGS. ALT.: A,X,Y
C21C          ;*************************************************************
C21C
C21C WRING    A01F   LDY #MES4-MES1
C21E          20DAC3 JSR KEPU        ;Outputs "SHURE (Y/N) ?"
C221          2040EC JSR GETKEY      ;Get character
C224          20D3C3 JSR FILT1       ;Capital letters
C227          C959   CMP #'Y'        ;If 'Y' then go on
C229          D04F   BNE WRING6
C223          20F0E9 JSR CRLF
C22E          20BCF8 JSR TOPNO       ;Start at top
C231 WRING1   A000   LDY #0          ;Go one line up in memory
C233          2013F7 JSR UP1
C236 WRING2   20E9F8 JSR ATBOT       ;At bottom ?
C239          B03F   BCS WRING6      ;Yes
C23B          A000   LDY #0          ;No, move character to DIBUFF
C23D WRING3   B1DF   LDA (NOWLN),Y
C23F          F039   BEQ WRING6
C241          9938A4 STA DIBUFF,Y
C244          C8     INY
C245          C90D   CMP #CR         ;End of line ?
C247          F003   BEQ WRING4      ;Yes, reduce blanks
C249          C93B   CMP #';'        ;';' ?
C24B          D0F0   BNE WRING3      ;No, next character
C24D          B936A4 LDA DIBUFF-2,Y
C250          C927   CMP #'''        ;Yes, lock for previous character
C252          F0E9   BEQ WRING3      ;Next character when '';'
```

```
C254 WRING4 88      DEY             ;Reduce left hand blanks
C255        F015    BEQ WRING5      ;Empty line ?
C257        B937A4 LDA DIBUFF-1,Y
C25A        C920    CMP #' '        ;Blank ?
C25C        F0F6    BEQ WRING4      ;Yes, lock for more
C25E        84EA    STY LENGTH      ;No, do the replace
C260        20B2F8 JSR CFLG
C263        2020C4 JSR CHARME
C266        203FF9 JSR REPLAC
C269        4C31C2 JMP WRING1       ;Next line
C26C
C26C WRING5 84EA    STY LENGTH      ;Kill empty line
C26E        20B6F8 JSR KIFLG
C271        2020C4 JSR CHARME
C274        203FF9 JSR REPLAC
C277        4C36C2 JMP WRING2       ;Next line is on
C27A
C27A WRING6 20BCF8 JSR TOPNO        ;Top of text
C27D        4C27F7 JMP PLNE         ;Show line
C280
C280
C280
C280        ;************************************************************
C280        ;*                      SUBROUTINES
C280        ;************************************************************
C280
C280
C280        ;************************************************************
C280        ;*  Dekrement pointer CURREL by 1          REGS. ALT.: A
C280        ;************************************************************
C280
C280 DEC2   C6DD    DEC CURREL
C282        A5DD    LDA CURREL
C284        C9FF    CMP #$FF
C286        D002    BNE *+4
C288        C6DE    DEC CURREL+1
C28A        60      RTS
C28B
C28B
C28B        ;************************************************************
C28B        ;*  Dekrement pointer SAVE by 1            REGS. ALT.: A
C28B        ;************************************************************
C28B
C28B DECSAV C6E7    DEC SAVE
C28D        A5E7    LDA SAVE
C28F        C9FF    CMP #$FF
C291        D002    BNE *+4
C293        C6E8    DEC SAVE+1
C295        60      RTS
C296        ;************************************************************
C296        ;*  Inkrement pointer CURADR by 1          REGS. ALT.: A
C296        ;************************************************************
C296
C296 INC1   E6DB    INC CURADR
C298        D002    BNE *+4
C29A        E6DC    INC CURADR+1
C29C        60      RTS
```

```
C29D            ;************************************************************
C29D            ;*  Inkrement pointer CURREL by 1          REGS. ALT.: A
C29D            ;************************************************************
C29D
C29D INC2   E6DD    INC CURREL
C29F        D002    BNE *+4
C2A1        E6DE    INC CURREL+1
C2A3        60      RTS
C2A4
C2A4
C2A4            ;************************************************************
C2A4            ;*  Inkrement pointer SAVU1 by 1           REGS. ALT.: A
C2A4            ;************************************************************
C2A4
C2A4 INC3   E6CF    INC SAVU1
C2A6        D002    BNE *+4
C2A8        E6D0    INC SAVU1+1
C2AA        60      RTS
C2AB
C2AB
C2AB            ;************************************************************
C2AB            ;*  Switch CAPS-LOCK-flag ON and OFF       REGS. ALT.: A
C2AB            ;************************************************************
C2AB
C2AB GROSKL ADD3A4 LDA FLGREG
C2AE        4940    EOR #$40
C2B0        8DD3A4 STA FLGREG
C2B3        60      RTS
C2B4
C2B4
C2B4            ;************************************************************
C2B4            ;*  Save current-line at SAVU1             REGS. ALT.: A
C2B4            ;************************************************************
C2B4
C2B4 SAVNOWU A5DF   LDA NOWLN
C2B6        85CF    STA SAU1
C2B8        A5E0    LDA NOWLN+1
C2BA        85D0    STA SAVU1+1
C2BC        60      RTS
C2BD
C2BD
C2BD            ;************************************************************
C2BD            ;*  Restore current-line from SAVU1        REGS. ALT.: A
C2BD            ;*********************+**************************************
C2BD
C2BD RSNOWU A5CF    LDA SAVU1
C2BF        85DF    STA NOWLN
C2C1        A5D0    LDA SAVU1+1
C2C3        85E0    STA NOWLN+1
C2C5        60      RTS
C2C6
C2C6
C2C6            ;************************************************************
C2C6            ;*  This routine asks for a line 'FROM' and a line 'TO'
C2C6            ;*  and stores the addresses in CURADR and CURREL.If CURADR
C2C6            ;*  is not smaller than CURREL you have to repeat the input.
C2C6            ;*                                         REGS. ALT.: A,X,Y
C2C6            ;************************************************************
```

```
C2C6 FROMTO A015    LDY #MES2-MES1
C2C8        20DAC3  JSR KEPU        ;Outputs "FROM ?"
C2CB FROTO1 2040EC  JSR GETKEY      ;Get a key
C2CE        C90D    CMP #CR         ;If CR, go on
C2D0        F00D    BEQ FROTO2
C2D2        C91B    CMP #ESC        ;If ESC, force a reentry
C2D4        D003    BNE *+5
C2D6        4CCFF6  JMP REENTR
C2D9        201FC3  JSR COMAND      ;Space,U,D,T,B-Command ?
C2DC        4CCBC2  JMP FROTO1
C2DF FROTO2 A5DF    LDA NOWLN       ;Save the current-line at CURADR
C2E1        85DB    STA CURADR
C2E3        8D2EA4  STA NAME        ;and also at NAME,NAME+1
C2E6        A5E0    LDA NOWLN+1
C2E8        85DC    STA CURADR+1
C2EA        8D2FA4  STA NAME+1
C2ED        A01B    LDY #MES3-MES1
C2EF        20DAC3  JSR KEPU        ;Outputs "TO ?"
C2F2 FROTO3 2040EC  JSR GETKEY      ;Get a key
C2F5        C90D    CMP #CR         ;;If CR, go on
C2F7        F00D    BEQ FROTO4
C2F9        C91B    CMP #ESC        ;If ESC, do a reentry
C2FB        D003    BNE *+5
C2FD        4CCFF6  JMP REENTR
C300        201FC3  JSR COMAND      ;Space,U,D,T,B command ?
C303        4CF2C2  JMP FROTO3
C306 FROTO4 A000    LDY #0
C308        2013F7  JSR UP1         ;1 line up in memory
C30B        A5DF    LDA NOWLN       ;Save current-line at CURREL
C30D        85DD    STA CURREL
C30F        8D30A4  STA NAME+2      ;and also at NAME+2,NAME+3
C312        A5E0    LDA NOWLN+1
C314        85DE    STA CURREL+1
C316        8D31A4  STA NAME+3
C319        2070C3  JSR GREAT       ;IS CURADR )= CURREL ?
C31C        B0A8    BCS FROMTO      ;Yes, branch to beginning
C31E        60      RTS
C31F
C31F
C31F        ;*************************************************************
C31F        ;*  This routine checks for and executes the (space),(up),
C31F        ;*  (down),(top) and (bottom) command.
C31F        ;*                                          REGS. ALT.: A,X,Y
C31F        ;*************************************************************
C31F
C31F COMAND C920    CMP #' '        ;Space-command ?
C321        D003    BNE COM0        ;No, branch
C323        4C27F7  JMP PLNE        ;Display current-line
C326 COM0   C955    CMP #'U'        ;Up-command ?
C328        D00B    BNE COM1        ;No, branch
C32A        20DBF8  JSR ATTOP       ;At beginning of text ?
C32D        B040    BCS COM5        ;Yes, branch to RTS
C32F        20E3F6  JSR DOW1        ;1 line down in memory
C332        4C6CC3  JMP COM4
C335 COM1   C944    CMP #'D'        ;Down-command ?
C337        D01F    BNE COM2        ;No, branch
C339        A000    LDY #0          ;Necessary for UP1-Routine
C33B        20E9F8  JSR ATBOT       ;At end of text ?
```

```
C33E        9008    BCC *+10        ;No, execute command
C340        A072    LDY #$72
C342        20AFE7  JSR KEP         ;Outputs "END"
C345        4CF0E9  JMP CRLF
C348        2013F7  JSR UP1         ;1 line up in memory
C34B        20E9F8  JSR ATBOT
C34E        901C    BCC COM4
C350        A072    LDY #$72

C352        20AFE7  JSR KEP
C355        4CF0E9  JMP CRLF
C358 COM2   C954    CMP #'T'        ;Top-command ?
C35A        D006    BNE COM3        ;No, branch
C35C        20BCF8  JSR TOPNO       ;Set current-line to top of text
C35F        4C6CC3  JMP COM4
C362 COM3   C942    CMP #'B'        ;Bottom-command ?
C364        D009    BNE COM5        ;No, branch
C366        20C5F8  JSR SETBOT      ;Set current-line to end of text
C369        20E3F6  JSR DOW1        ;1 line down in memory
C36C COM4   4C27F7  JMP PLNE        ;Display the current-line
C36F COM5   60      RTS
C370
C370
C370        ;*************************************************************
C370        ;*      This routine checks whether CURADR is greater or
C370        ;*      equal CURREL. Carry-Flag is set if so.
C370        ;*                                              REGS. ALT.: A
C370        ;*************************************************************
C370
C370 GREAT  A5DC    LDA CURADR+1
C372        C5DE    CMP CURREL+1    ;CURADR+1 ( CURREL+1 ?
C374        9008    BCC GREAT1      ;Yes, branch to clear carry
C376        D008    BNE GREAT2      ;No, branch to set carry
C378        A5DB    LDA CURADR      ;Compare lower halfs
C37A        C5DD    CMP CURREL      ;CURADR ( CURREL ?
C37C        B002    BCS GREAT2      ;No, branch to set carry
C37E GREAT1 18      CLC
C37F        60      RTS
C380 GREAT2 38      SEC
C381        60      RTS
C382
C382
C382        ;*************************************************************
C382        ;*   Compare CURADR and SAVU1. Set carry-flag if CURADR IS
C382        ;*   greater or equal than SAVU1.           REGS. ALT.: A
C382        ;*************************************************************
C382
C382 COMP1  A5DC    LDA CURADR+1    ;Compare upper halfs
C384        C5D0    CMP SAVU1+1     ;CURADR+1 ( SAVU1+1 ?
C386        90F6    BCC GREAT1      ;Yes
C388        D0F5    BNE GREAT2      ;No
C38A        A5DB    LDA CURADR      ;Compare lower halfs
C38C        C5CF    CMP SAVU1       ;CURADR ( SAVU1 ?
C38E        B0F0    BCS GREAT2      ;No, branch to set carry-flag
C390        18      CLC
C391        60      RTS
C392
C392
```

```
C392        ;*************************************************************
C392        ;*   Does a 16-bit subtraction ( CURREL - CURADR )
C392        ;*                                              REGS. ALT.: A
C392        ;*************************************************************
C392
C392 SBFRTO 38      SEC
C393        A5DD    LDA CURREL
C395        E5DB    SBC CURADR      ;Lower halfs
C397        85DD    STA CURREL
C399        A5DE    LDA CURREL+1
C39B        E5DC    SBC CURADR+1    ;Upper halfs

C39D        85DE    STA CURREL+1
C39F        60      RTS
C3A0        ;*************************************************************
C3A0        ;*   This routife checks whether the current-line lies bet-
C3A0        ;*   ween the line 'FROM' and line 'TO'. Carry-Flag is set
C3A0        ;*   if so.                                    REGS. ALT.: A
C3A0        ;*************************************************************
C3A0
C3A0 COFRTO A5E0    LDA NOWLN+1     ;Compare upper halfs
C3A2        C5DC    CMP CURADR+1    ;NOWLN+1 ( CURADR+1 ?
C3A4        9008    BCC COFRN1      ;Yes, branch to clear carry
C3A6        D008    BNE COFRN2      ;No, go on
C3A8        A5DF    LDA NOWLN       ;If equal, compare lower halfs
C3AA        C5DB    CMP CURADR      ;NOWLN ( CURADR ?
C3AC        B002    BCS COFRN2      ;No, go on
C3AE COFRN1 18      CLC             ;Yes, set carry-flag
C3AF        60      RTS
C3B0 COFRN2 A5DE    LDA CURREL+1    ;Compare upper halfs
C3B2        C5E0    CMP NOWLN+1     ;CURREL+1 ( NOWLN ?
C3B4        90F8    BCC CORN1      ;Yes, branch to clear carry
C3B6        D008    BNE COFRN3      ;If not equal branch to set carry
C3B8        A5DD    LDA CURREL      ;If equal, compare lower halfs
C3BA        C5DF    CMP NOWLN       ;CURREL ( NOWLN ?
C3BC        B002    BCS COFRN3      ;No, set carry-flag
C3BE        18      CLC             ;Yes, clear carry-flag
C3BF        60      RTS
C3C0 COFRN3 38      SEC
C3C1        60      RTS
C3C2        ;*************************************************************
C3C2        ;*   Restore Bottom-Line from SAVE             REGS. ALT.: A
C3C2        ;*************************************************************
C3C2
C3C2 RESBOT A5E7    LDA SAVE
C3C4        85E1    STA BOTLN
C3C6        A5E8    LDA SAVE+1
C3C8        85E2    STA BOTLN+1
C3CA        60      RTS
C3CB        ;*************************************************************
C3CB        ;*   This little routine allows to find more than one
C3CB        ;*   occurences of the old-string in one line.
C3CB        ;*                                              REGS. ALT.: A,X,Y
C3CB        ;*************************************************************
C3CB
C3CB FNDSTR A5DD    LDA CURREL      ;Last curser-position
C3CD        8D15A4 STA CURPO2       ;Beginning of next search
C3D0        4C62F8 JMP FC8+3        ;Find next occurence of old-string
```

```
C3D3          ;************************************************************
C3D3          ;*  Makes capital letters os ASCII-Code    REGS. ALT.: A
C3D3          ;************************************************************
C3D3
C3D3 FILT1    C940    CMP #$40        ;Letter ?
C3D5          3002    BMI FILT2

C3D7          29DF    AND #$DF
C3D9 FILT2    60      RTS
C3DA
C3DA
C3DA          ;************************************************************
C3DA          ;*  Output Messages                          REGS. ALT.: A,Y
C3DA          ;************************************************************
C3DA
C3DA KEPU     B9EBC3  LDA MES1,Y      ;Which message ?
C3DD          48      PHA             ;Save it
C3DE          297F    AND #$7F        ;Convert to ASCII
C3E0          207AE9  JSR OUTPUT      ;Output to D/P
C3E3          C8      INY             ;Next one
C3E4          68      PLA             ;Restore original value
C3E5          10F3    BPL KEPU        ;End of text ? (if MSB=1)
C3E7          2044EB  JSR CLR         ;Clears D/P pointers
C3EA          60      RTS
C3EB
C3EB MES1     4558    .BYT 'EXTENDED EDITOR V2.3',$A0
C3FF          A0
C400 MES2     4652    .BYT 'FROM ',$BF
C405          BF
C406 MES3     544F20  .BYT 'TO ',$BF
C409          BF
C40A MES4     5348    .BYT 'SHURE (Y/N) ',$BF
C416          BF
C417 MES5     4445    .BYT 'DESTIN. ',$BF
C41F          BF
C420
C420          ;************************************************************
C420          ;*    Computes length of current-line (without CR)
C420          ;*                                          REGS. ALT.: A,Y
C420          ;************************************************************
C420
C420 CHARME   A000    LDY #0          ;Init counter
C422 CHARM1   B1DF    LDA (NOWLN),Y   ;Get a character
C424          F007    BEQ CHARM2      ;End of text ?
C426          C90D    CMP #CR         ;End of line ?
C428          F003    BEQ CHARM2      ;Yes, branch
C42A          C8      INY             ;Inkrement counter
C42B          D0F5    BNE CHARM1      ;Branch always
C42D CHARM2   84E9    STY OLDLEN      ;Save length
C42F          60      RTS
C430
C430
C430          ;************************************************************
C430          ;*************       SCREEN-EDITOR     **********************
C430          ;************************************************************
C430
C430 SCREEN           *=*             ;Not implemented here
C430                                  ;see MIKRO MAG 30                  □□
```

Erweiterung des EXEDIT

Die vorstehende Arbeit von Wollenberg und Redder (EXEDIT), war für den Autor Anlaß, sich mit der Implementierung weiterer Dienstleistungen zu befassen, die hier zur Anregung abgedruckt werden. Unglücklicherweise ging das erste Quellfile für die Erweiterungen verloren, so daß es ohne Kommentare rekonstruiert werden mußte. Es ist gleichwohl voll arbeitsfähig und benutzt die Symbole von EXEDIT sowie weitere bekannte aus dem Monitor-Listing des AIM 65.

Wie ersichtlich, wurden die gültigen Kommandos in COMTBL erweitert, dementsprechend auch die Zahl der Kommandos in COMCNT und die Vektoren in JMPTBL. Es stehen damit folgende weitere Dienstleistungen zur Verfügung:

Taste H: HELP. Es wird die Message MES6 zur Anzeige gebracht, um den Benutzer zu orientieren.

Taste V: Der Find-Befehl kann repetiert werden, indem man CR betätigt. Jede andere Taste führt zum Verlassen der Funktion.

Taste F3: Finde repetiert aufwärts Label an den Zeilenanfängen, solange (wie vor) CR beätigtwird. Diese Funktion ist ggfs. so zu erweitern, daß sie auch zuverlässig in der Zeile sucht.

TASTE /: Concatenierung, Zusammenschweißung zweier Zeilen zu einer. Es wird ein Space eingeschoben (sinnvollerweise). Achtung, wenn die resultierende Zeile mehr als 60 Zeichen erhält!

Taste S: Swap Editors. Es kann ein zweiter Textspeicher aufgemacht werden. Und hernach geht man mit 'S' zwischen diesen hin und her. Für das erstmalig richtige Funktionieren ist Voraussetzung, daß die 6 Zellen ab A474 zu 0 initialisiert waren (z.B. durch weitere Befehle bei EDINI. Das Arbeiten mit 2 Textspeichern ist durchaus angenehm, mit dem einen kann man dauerhaft etwas festhalten, mit dem anderen experimentiert man.

Taste # : Anzeige der Pointer für den Textpuffer in BOTLN, TEXT und END.

Taste . (Punkt): Der gefüllte Textpuffer wird ab angezeigter (und einschließlich) angezeigter Zeile nach unten abgeschnitten, womit die Vorzeile die letzte wird.

Taste ; (Semikolon): Es werden alle Zeilen ab Textbeginn bis vor die angezeigte abgeschnitten. Der verbleibende Text aus dem Rest des Textbuffers wird auf den Anfang des Textpuffers gezogen. Mit den Befehlen Punkt und Komma lassen sich also bequem Segmente aus einem vorhandenen Quelltext für eine anderweitige Verwendung herausschneiden, z.B. auch für eine Unterprogramm-Bibliothek.

Die Tasten 5, 6 und N haben die gleiche Wirkung wie im Monitorprogramm: Man springt direkt aus dem Editierprogramm in eine Sprache (COLD oder WARM) oder in den Assembler.

Es gäbe noch weitere nützliche Funktionen, die hier noch nicht implementiert sind, für die es in dieser Zeitschrift jedoch schon Vorschläge gegeben hat: Virtuelle Numerierung der Zeilen, Aufschlagen einer Zeile mit Nummer X, Weiterblättern (auch unsichtbar) um n Zeilen in beide Richtungen, Concatenierung zweier kompletter Textpuffer, Hinzufügen von Kommentaren mit ; in die offene Zeile von Assembler-Texten, Trennen einer offenen Zeile zu 2 Zeilen usw..

Roland Löhr

```
EXTENSIONS TO EXEDIT BY ROLAND LöHR
8949 5A41    0126 COMTBL .BYT "ZAEYMXWV^56N/HS#.;"
895B 7F89    0127 JMPTBL .WOR CAPLCK,DSONOF,ERAS,COPY
895D 8C89    0127
895F 9589    0127
8961 DB89    0127
8963 1E8B    0128        .WOR MOVE,XCHANG,WRING,VIND,SRCHUP
8965 A38A    0128
8967 3A8B    0128
8969 158E    0128
896B 2B8E    0128
896D 00B0    0129        .WOR COLD,WARM,ASSEM,CONCAT,HELP,SWAP,STATUS
```

```
896F 03B0     0129
8971 00D0     0129
8973 8E8E     0129
8975 988E     0129
8977 9D8E     0129
8979 C78E     0129
897B EC8E     0130          .WOR CUTOFF
897D 028F     0131          .WOR LIFTUP
8D30          0556          .OPT LIS
8D30 452D     0557 MES6     .BYT 'E-ERASE',$D
8D37 0D       0557
8D38 4D2D     0558          .BYT 'M-MOVE',$D
8D3E 0D       0558
8D3F 592D     0559          .BYT 'Y-COPY',$D
8D45 0D       0559
8D46 412D     0560          .BYT 'A-DIPL ON/OFF',$D
8D53 0D       0560
8D54 5A2D     0561          .BYT 'Z-CAPLCK',$D
8D5C 0D       0561
8D5D 562D     0562          .BYT 'V-REP-FIND',$D
8D67 0D       0562
8D68 572D     0563          .BYT 'W-WRING',CR
8D6F 0D       0563
8D70 2E43     0564          .BYT '.CUTOFF',CR
8D77 0D       0564
8D78 3B2D     0565          .BYT ';-ERASE BLOCK TO',CR
8D88 0D       0565
8D89 5E2D     0566          .BYT '^-FIND UPW',$D
8D93 0D       0566
8D94 352D     0567          .BYT '5-COLD',$D
8D9A 0D       0567
8D9B 362D     0568          .BYT '6-WARM',$D
8DA1 0D       0568
8DA2 4E2D     0569          .BYT 'N-ASSEMBLER',$D
8DAD 0D       0569
8DAE 2F2D     0570          .BYT '/-CONCAT',$D
8DB6 0D       0570
8DB7 532D     0571          .BYT 'S-SWAP EDITORS',$D
8DC5 0D       0571
8DC6 232D     0572          .BYT '#-STATUS',$8D
8DCE 8D       0572
8DCF 5357     0573 MES8     .BYT 'SWAP EDITORS SHURE Y/N ?',$8D
8DE7 8D       0573
8DE8 5354     0574 MES9     .BYT 'STATUS OF ED',CR,'BOTLN START END',$8D
8DF4 0D       0574
8DF5 424F     0574
8E04 8D       0574
8E15          0587          .OPT LIS
```

```
REPEAT FIND FUNCTION WITH KEY V
8E15 200CF8   0589 VIND     JSR FCHAR
8E18 2040EC   0590          JSR GETKEY
8E1B C90D     0591          CMP #CR
8E1D F003     0592          BEQ VIND3
8E1F 4C78FA   0593 H8F1B    JMP ERRO
8E22 202EF8   0594 VIND3    JSR FC2
8E25 200FF8   0595          JSR FCHA1
8E28 4C188E   0596          JMP VIND+3
```

```
SEARCH UPWARRDS A LABEL WITH KEY F3
8E2B            0598 SRCHUP
8E2B 203B8E     0599 H8F27  JSR H8F37
8E2E 200FF8     0600        JSR FCHA1
8E31 2040EC     0601        JSR GETKEY
8E34 C90D       0602        CMP #CR
8E36 D0E7       0603        BNE H8F1B
8E38 4C2B8E     0604        JMP H8F27
8E3B A000       0605 H8F37  LDY ##00
8E3D 20BDE7     0606        JSR PROMPT
8E40 205FE9     0607 H8F3C  JSR RDRUB
8E43 C90D       0608        CMP ##0D
8E45 D00A       0609        BNE H8F4D
8E47 C000       0610        CPY ##00
8E49 D006       0611        BNE H8F4D
8E4B 20D8F6     0612 H8F47  JSR DNNO
8E4E 4C668E     0613        JMP H8F62
8E51 C90D       0614 H8F4D  CMP ##0D
8E53 F00B       0615        BEQ H8F5C
8E55 99EB00     0616        STA STRING,Y
8E58 C8         0617        INY
8E59 C014       0618        CPY ##14
8E5B D0E3       0619        BNE H8F3C
8E5D 4C72FA     0620        JMP ERROR
8E60 2024EA     0621 $8F5C  JSR CRCK
8E63 8C29A4     0622        STY $A429
8E66 A000       0623 H8F62  LDY ##00
8E68 8C15A4     0624        STY CURP02
8E6B AC15A4     0625 H8F67  LDY CURP02
8E6E A200       0626        LDX ##00
8E70 B1DF       0627 H8F6C  LDA (NOWLN),Y
8E72 D003       0628        BNE H8F73
8E74 4C5CFA     0629        JMP ENDERR
8E77 C90D       0630 H8F73  CMP ##0D
8E79 F0D0       0631        BEQ H8F47
8E7B D9EB00     0632        CMP STRING,Y
8E7E F006       0633        BEQ H8F82
8E80 EE15A4     0634        INC CURP02
8E83 4C6B8E     0635        JMP H8F67
8E86 C8         0636 H8F82  INY
8E87 E8         0637        INX
8E88 EC29A4     0638        CPX $A429
8E8B D0E3       0639        BNE H8F6C
8E8D 60         0640        RTS

CONCATENATE 2 LINES, KEY /
8E8E 20058E     0642 CONCAT JSR H8F01
8E91 A920       0643        LDA ##20
8E93 91DF       0644        STA (NOWLN),Y
8E95 4C27F7     0645        JMP PLNE

DISPLAY HELP FOR COMMANDS, KEY H
8E98 A027       0647 HELP   LDY #MES6-MES1
8E9A 4CF88C     0648        JMP KEPU

SWAP BETWEEN 2 EDITORS, KEY S
8E9D A0C6       0650 SWAP   LDY #MES8-MES1
8E9F 20F88C     0651        JSR KEPU
```

```
8EA2 2040EC   0652          JSR GETKEY
8EA5 C959     0653          CMP #$59
8EA7 F003     0654          BEQ H8FA8
8EA9 4CD4E7   0655          JMP QM
8EAC A205     0656 H8FA8    LDX #$05
8EAE B5E1     0657 H8FAA    LDA BOTLN,X
8EB0 BC74A4   0658          LDY $A474,X
8EB3 9D74A4   0659          STA $A474,X
8EB6 94E1     0660          STY BOTLN,X
8EB8 CA       0661          DEX
8EB9 10F3     0662          BPL H8FAA
8EBB A5E4     0663          LDA TEXT+1
8EBD 05E6     0664          ORA END+1
8EBF D003     0665          BNE H8FC0
8EC1 4C41F6   0666          JMP EDIT
8EC4 4CCFF6   0667 H8FC0    JMP REENTR
-------------------------------------------------------------------
DISPLAY 3 POINTERS OF TEXTBUFFER, KEY #
8EC7 A0DF     0669 STATUS   LDY #MES9-MES1
8EC9 20F88C   0670          JSR KEPU
8ECC A000     0671          LDY #$00
8ECE A920     0672 H8FCA    LDA #$20
8ED0 207AE9   0673          JSR OUTPUT
8ED3 B9E200   0674          LDA BOTLN+1,Y
8ED6 B6E1     0675          LDX BOTLN,Y
8ED8 2042EA   0676          JSR WRAX
8EDB A920     0677          LDA #$20
8EDD 207AE9   0678          JSR OUTPUT
8EE0 C8       0679          INY
8EE1 C8       0680          INY
8EE2 C006     0681          CPY #$06
8EE4 D0E8     0682          BNE H8FCA
8EE6 20F0E9   0683          JSR CRLF
8EE9 4C27F7   0684          JMP PLNE
-------------------------------------------------------------------
LIMIT TEXTBUFFER PARAMTER BOTLN=NOWLN, KEY .
8EEC A000     0686 CUTOFF   LDY #0
8EEE 98       0687          TYA
8EEF 91DF     0688          STA (NOWLN),Y
8EF1 A5DF     0689          LDA NOWLN
8EF3 85E1     0690          STA BOTLN
8EF5 A5E0     0691          LDA NOWLN+1
8EF7 85E2     0692          STA BOTLN+1
8EF9 20D8F6   0693          JSR DNNO          ;DNNO
8EFC 2027F7   0694          JSR PLNE
8EFF 4C2789   0695          JMP EXEDIT
-------------------------------------------------------------------
MAKE NOWLN=TOPLINE, LIFTUP, KEY ;
8F02          0697 ;DELETE ALL LINES TILL NOWLN
8F02 A000     0698 LIFTUP   LDY #0
8F04 A5E3     0699          LDA TEXT
8F06 48       0700          PHA               ;SAVE
8F07 A5E4     0701          LDA TEXT+1
8F09 48       0702          PHA
8F0A B1DF     0703 TRANS    LDA (NOWLN),Y
8F0C 91E3     0704          STA (TEXT),Y
8F0E F00A     0705          BEQ RAUS          ;DELIMITER FOUND
8F10 C8       0706          INY
```

```
8F11 D0F7     0707          BNE TRANS
8F13 E6E0     0708          INC NOWLN+1
8F15 E6E4     0709          INC TEXT+1
8F17 4C0A8F   0710          JMP TRANS
8F1A 18       0711 RAUS     CLC
8F1B 98       0712          TYA                ;OFFSET
8F1C 65E3     0713          ADC TEXT
8F1E 85E1     0714          STA BOTLN
8F20 A5E4     0715          LDA TEXT+1
8F22 6900     0716          ADC #0
8F24 85E2     0717          STA BOTLN+1
8F26 68       0718          PLA                ;RESTORE TEXT POINTER
8F27 85E4     0719          STA TEXT+1
8F29 68       0720          PLA
8F2A 85E3     0721          STA TEXT
8F2C 4CCFF6   0722          JMP REENTR
-------------------------------------------------------------------
ADDITIONAL COMMANDS
8F2F          0724 ;KEY 5 = COLD, COLD ENTRY INTO LANGUAGE
8F2F          0725 ;KEY 6 = WARM, WARM ENTRY INTO LANGUAGE
8F2F          0726 ;KEY N = START OF ASSEMBLER
-------------------------------------------------------------------

8F2F          0728          .END

  ERRORS= 0000                                                  □□
```

AIM Spezial (14)

Heft 28 dieser Zeitschrift enthielt ein revidiertes Monitorprogramm für den AIM aus der Feder der Herren Wollenberg und Redder: REMON und auch den Artikel AIM Spezial (13). Nach weiterer Arbeit am Monitor regen die genannten Autoren jetzt folgende Verbesserungen und Erweiterungen an, für die sich weiter unten auch eine Programmliste findet. Die nachfolgenden numerierten Bezüge betreffen den ursprünglichem Artikel zu REMON.

1.4.3 Verbesserte Repeat-Funktion. Es wurde eine komfortable Version implementiert, die bei dem ersten Tastendruck eine lange Wartezeit von 320 ms für die Zeichenwiederholung hat und die nach längerer Betätigung der Taste auf 60 ms reduziert ist (das Repeat wird beschleunigt).

1.4.7 Erweiterung der Monitorbefehle. Ab der Version REMON 3.3 sind einige Monitorkommandos zusätzlich eingeführt, wie Checksummenbildung, Ansprung von Routinen des Anwenders und der '—'-Befehl, der den M-Befehl zum rückwärtigen Absuchen von Speicherstellen realisiert.

1.4.13 Deutscher Zeichensatz. Um eine Textverarbeitung mit deutschem Zeichensatz zu ermöglichen, wurde der Printbefehl unterdrückt. Stattdessen ist jetzt das 'ö' eingebbar. Die übrigen deutschen Zeichen liegen auf den Tasten F1, F2 und F3. Natürlich läßt sich der Printer weiterhin über CTRL-Print an- und Ausschalten. Die Tastenkappen sind zu vertauschen, um das Keyboard an die deutsche Norm anzunähern.

Tastatur	ASCII	Video	Tastaturbelegung alt	neu	Adresse	neuer Inhalt
F1	5B/7B	Ä/ä	F1	;	F460	3B
F2	5D/7D	Ü/ü	F2	—	F45F	2D
			—	F2	F434	5D
F3	5E/7E	/ß	gleich			
			@	F1	F42A	5B
			:	Print	F437	5C

Print	5C/7C	Ö/ö	Print	@	F42B	60
			Y	Z	F443	5A
			Z	Y	F451	59

3.4 Groß- und Kleinschreibung. Mit CTRL-A wird von Groß- auf gemischte Groß- und Kleinschreibung sowie umgekehrt umgeschaltet. CTRL-Z schaltet das LED-Display an und ab.

```
-------------------------------------------------------------
Verbesserte Repeat-Funktion
0000            0021            *=$ECEF
ECEF 8A         0022 ROONEK     TXA                 ;rette X
ECF0 48         0023            PHA
ECF1 A240       0024            LDX #$40            ;langes Tastenrepeat
ECF3 AD08A4     0025            LDA $A408
ECF6 6A         0026            ROR A
ECF7 9002       0027            BCC *+4
ECF9 A20A       0028            LDX #$A             ;kurzes Tastenrepeat
ECFB 20EAEF     0029            JSR HELP7
ECFE 68         0030            PLA
ECFF AA         0031            TAX                 ;altes X
ED00            0032 ROO1                           ;...
-------------------------------------------------------------
HELP7
ED00            0034            *=$EFEA
EFEA AD82A4     0035 HELP7      LDA DRB2
EFED 0D7FA4     0036            ORA ROLLFL
EFF0 49FF       0037            EOR #$FF
EFF2 D008       0038            BNE *+10
EFF4 A9FE       0039            LDA #$FE            ;andere Taste
EFF6 2D08A4     0040            AND $A408
EFF9 4C30E4     0041            JMP HELP7B
EFFC 4C25E4     0042            JMP HELP7A
EFFF            0043
EFFF            0044            *=$E425
E425 2036ED     0045 HELP7A     JSR $ED36           ;GLEICHE TASTE
E428 CA         0046            DEX
E429 D009       0047            BNE *+11
E42B A901       0048            LDA #$1
E42D 0D08A4     0049            ORA $A408
E430 8D08A4     0050 HELP7B     STA $A408
E433 60         0051            RTS
E434 4CEAEF     0052            JMP HELP7
-------------------------------------------------------------
UMSCHALTUNG DER GROSS-KLEINSCHREIBUNG
E437            0054            *=$ECE1
ECE1 4CBEF2     0055            JMP $F2BE
ECE4 EA         0056            NOP
ECE5 EA         0057            NOP
ECE6 EA         0058            NOP
ECE7 EA         0059            NOP
ECE8 EA         0060            NOP
ECE9 EA         0061            NOP
ECEA EA         0062            NOP
ECEB            0063 ; ...
ECEB            0064
ECEB            0065            *=$F2BE
F2BE C901       0066            CMP #$1             ;CTRL-A ?
F2C0 D004       0067            BNE *+6
F2C2 A940       0068            LDA #$40            ;TOGGLE GROSS-/KLEIN-BIT
```

```
F2C4 D006     0069          BNE *+8
F2C6 C91A     0070          CMP #$1A          ;CTRL-Z ?
F2C8 D00B     0071          BNE *+13
F2CA A980     0072          LDA #$80          ;TOGGLE DISPLAY ON/OFF-BIT
F2CC 4D08A4   0073          EOR $A408
F2CF 8D08A4   0074          STA $A408
F2D2 4C40EC   0075          JMP GETKEY
F2D5 60       0076          RTS
F2D6 EA       0077          NOP

-------------------------------------------------------------
ERWEITERUNG DER MONITORBEFEHLE
F2D7          0079          *=$E19E
E19E 2087E5   0080          JSR HELP5         ;TRITT AN DIE STELLE
E1A1          0081                              VON JSR QM
E1A1          0082          *=$E587
E587 C943     0083 HELP5    CMP #'C ;C - CHECKSUMMENBILDUNG
E589 D03C     0084          BNE HELP52
E58B 20A3E7   0085          JSR FROM          ;STARTADRESSE
E58E B0FB     0086          BCS *-3
E590 203EE8   0087          JSR BLANK
E593 2010F9   0088          JSR ADDRS1
E596 20A7E7   0089          JSR TO            ;ENDADRESSE+1
E599 B0FB     0090          BCS *-3
E59B 204DEB   0091          JSR CRLCK
E59E A000     0092 HELP51   LDY #0
E5A0 A91A     0093          LDA #<S1
E5A2 2058EB   0094          JSR LDAY
E5A5 204EE5   0095          JSR CHEKA
E5A8 205DE5   0096          JSR ADDS1
E5AB AD1AA4   0097          LDA S1
E5AE CD1CA4   0098          CMP ADDR
E5B1 D0EB     0099          BNE HELP51
E5B3 AD1BA4   0100          LDA S1+1
E5B6 CD1DA4   0101          CMP ADDR+1
E5B9 D0E3     0102          BNE HELP51
E5BB 2013EA   0103          JSR CRLOW
E5BE AE1EA4   0104          LDX CKSUM
E5C1 AD1FA4   0105          LDA CKSUM+1
E5C4 4C42EA   0106          JMP WRAX
E5C7 C94F     0107 HELP52   CMP #'O ; O = COLD RESET
E5C9 D006     0108          BNE HELP5J
E5CB 8D02A4   0109          STA NMIV2
E5CE 4CBFE0   0110          JMP RSET
E5D1 4CA4E3   0111 HELP5J   JMP HELP53
E5D4          0112
E5D4          0113          *=$E3A4
A4 C92D     0114 HELP53 CMP #'- ;MEMORY BACKWARD ?
E3A6 D014     0115          BNE HELP54
E3A8 203EE8   0116          JSR $E83E
E3AB AD1CA4   0117          LDA $A41C
E3AE 38       0118          SEC
E3AF E904     0119          SBC #$4
E3B1 8D1CA4   0120          STA $A41C
E3B4 B003     0121          BCS *+5
E3B6 CE1DA4   0122          DEC $A41D
E3B9 4C15E6   0123          JMP $E615
E3BC C937     0124 HELP54   CMP #'7 ;7 - SPRUNG NACH $C000
E3BE D003     0125          BNE *+5
```

```
E3C0 4C00C0   0126          JMP $C000       ;BEISPIEL: ROUTINE DES ANW
E3C3 C938     0127          CMP #'8 ;8 = SPRUNG NACH $C003
E3C5 D003     0128          BNE *+5
E3C7 4C03C0   0129          JMP $C003       ;BEISPIEL: ROUTINE DES ANW
E3CA 6C0AA4   0130 OUT      JMP ($A40A)
E3CD          0131          .END
```

AIM 65 mit RS-232C-Interface für die serielle Datenübermittlung. In Anlehnung an das Rockwell-Document No. R6500 N08 vom 1.7.1979 enthielt Heft 6 dieser Zeitschrift auf S. 48 (Das Buch 1-1-6, Seite 194) einen Vorschlag, wie ein RS232-Interface zu realisieren sei. Auf dem AIM 65 des Herausgebers ergaben sich mit ihm gleichwohl Schwierigkeiten beim Datenempfang mit 9600 Baud. Zunächst wurde daher der Widerstand R24 auf 4.9k erhöht (alte Versionen des AIM hatten hier nur 1k) und der Kondensator C21 entfernt. Als auch das keine Abhilfe brachte, wurde der Ausgang an Pin 6 des Z5 74LS38 (NAND-Leistungsgatter) von PB6 der VIA 6522 abgetrennt. Stattdessen wurde ein Leitungsempfänger für V24-Schnittstellen SN75189 montiert und einer seiner Ausgänge mit PB6 der VIA verbunden. Weiterhin wurde eine freie Zunge am Applikationsstecker J1 mit dem Eingang des RS232-Signales belegt und mit dem Leitungsempfänger 75189 verbunden. Der Empfang serieller Signale mit 9600 Baud klappt nun einwandfrei. □□

FORTH (7)

17. Massenspeicherung

Die Schöpfer der Sprache FORTH haben sich bemüht, dem Anwender in der Software ein möglichst anpassungsfähiges Interface für die Massenspeicherung zur Verfügung zu stellen. Es ist darauf angelegt, mit einem möglichst großen virtuellen Speicher zusammenarbeiten zu können. Benötigte Information wird automatisch in den Computer gezogen, wenn sie nicht resident ist und herausgeschrieben, wenn sie bearbeitet worden ist.

17.1 Screens

Computer mit FORTH haben von Fall zu Fall eine abweichende hardwaremäßige Umgebung. Die Massenspeicherung wurde vor allem für Systeme mit Diskette konzipiert, und hier ist ein Segment mit 1024 Byte, genannt SCREEN, ein Hauptbegriff. Screens werden mit einer laufenden Nummer aufgerufen und in einen Ein-/Ausgabepuffer des Computers geladen. Von hier aus können Programme in das Dictionary kompiliert oder bearbeitet und in der veränderten Form wieder auf Massenspeicher abgesetzt werden. Das Gleiche gilt für Daten.

In der Leserschaft dieser Zeitschrift wird vor allem der AIM 65 mit FORTH betrieben. Daher ein Wort zu seinem FORTH: Es ist von Haus aus nicht auf Screens und Diskettenbetrieb orientiert, hat dafür jedoch die softwaremäßigen Anschlußpunkte, so daß der Leser entscheiden kann, ob er eine entsprechende Treiberroutine für das Arbeiten mit und für das Editieren von Screens und für den Verkehr mit Datenblöcken implementieren möchte. Ohne zusätzliche Arbeiten ist die Zusammenarbeit besonders mit dem Text-Editor für die Eingabe von Programmen (SOURCE IN= M), mit Tapes oder mit einer User-Device geregelt.

Das Arbeiten mit Screens ist nicht unbedingt zu empfehlen, wenn man einen Datenspeicher hat, auf den man nur sequentiell zugreifen kann und wenn Screens in veränderter Form wieder abgespeichert werden sollen. Man muß dann entweder das ganze File in der geänderten Form kopieren oder man muß in der Lage sein, den geänderten Block positionsgenau zu überschreiben, was bei formatierten Bandspeichern durchaus möglich ist.

17.2 Blocks, Sektoren

Der Begriff Block wird häufig gleichsinnig mit Screen benutzt, er ist aber genauer als Untermenge eines Screens zu bezeichnen und ist als ein Diskettensektor mit 128 oder 256 Byte zu sehen, so daß 8 oder 4 Diskettensektoren einen Screen ausmachen. Die Konstante B/SCR (Blocks per Screen) gibt diesen Multiplikator an.

Wir kommen damit auf eine weitere Eigenheit der Diskettenspeicherung und der Blocks: FORTH arbeitet nicht mit benannten Files, sondern mit Screens und Screen-Nummern. Das setzt voraus, daß der Computer die Screen-Nummer in eine Reihe von Sektor-Nummern umsetzen und den gewünschten Sektor gezielt hereinholen oder beschreiben kann. Bei einigen FORTH-Implementierungen für CBM fällt nun auf, daß die Disketten speziell für die Arbeit mit Screens formatiert werden, um 256 statt der üblichen 254 Nutzbytes je Sektor zu erhalten. Dabei wird auch die Directory-Spur überschrieben. Das wäre nicht nötig, wenn man stattdessen ein relatives File schüfe mit einer Datensatzgröße von 128. Und: Eine Diskette ohne Inhaltsverzeichnis gerät in Gefahr, versehentlich falsch behandelt zu werden.

17.3 Screen-Parameter

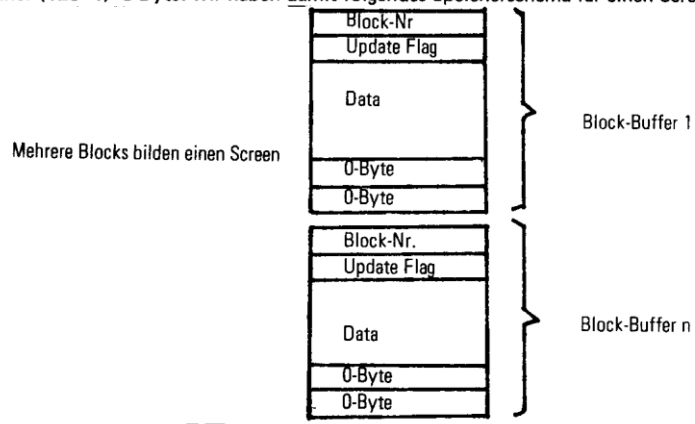
Ein Screen enthält im allgemeinen 1024 Bytes, die 16 virtuell numerierte Zeilen (Nr. 0-15) mit je 64 Zeichen bilden.

C/L	Konstante mit Characters per Line, Zeichen pro Zeile (beim AIM: 60) UC/L bringt die Adresse von C/L auf den Stack. Man kann also die 'Konstante' ändern.
B/SCR	Konstante: Blocks per Screen, im allgemeinen 8. UB/SCR bringt die Adresse der Konstanten auf den Stack.
B/BUF	Konstante: Bytes per Buffer (Sektor), im allgemeinen 128. UB/BUF bringt die Adresse von B/BUF auf den Stack.
FIRST	Konstante, die die Anfangsadresse des zugewiesenen E/A-Speicherbereiches (Buffers) hält.. UFIRST bringt die Adresse von FIRST zwecks Zuweisung auf den Stack.
LIMIT	Konstante, die die Adresse+1 der Obergrenze des zugewiesenen E/A-Bereiches hält. ULIMIT bringt die Adresse der Konstanten auf den Datenstack.

Wie man sieht, kann jede der zuvor genannten 'Konstanten' vom Anwender geändert werden, z.B.

```
HEX 2000 UFIRST !
```

Blocks (Sektoren) sind also die Untermenge eines Screens. Wie ist nun der Ein- und Ausgabebereich zwischen FIRST und LIMIT zu organisieren? Dazu muß man wissen, daß je Block 4 Byte mehr RAM benötigt werden, als es der Sektorgröße entspricht. Ein Block-Buffer enthält im 1. Byte die Block-Nummer, im 2. Byte das 'Update Flag'. (in Bit 7). Er folgen die im allgemeinen 128 Datenbytes und schließlich zwei hex Nullen als Trenner. Ein Screen aus z.B. 8 Blocks (Sektoren) benötigt daher (128+4)*8 Byte. Wir haben damit folgendes Speicherschema für einen Screen:



Der E/A-Bereich für die Massenspeicherung kann aber auch anders organisiert werden, als es dem Schema entspricht. Es sind alle Lösungen mit mehr als 2 Block-Buffern möglich.

PREV bringt die Adresse einer Variablen auf den Stack, die die Adresse des zuletzt angesprochenen Block-Buffers hält.

USE ist eine Systemvariable, die die Adresse des nächsten anzusprechenden Block-Buffers enthält. PREV und USE sind bei Arbeitsbeginn auf den Inhalt von FIRST zu initialisieren (beim AIM zeigen sie nach COLD auf hex 9000).

17.4 Befehlsworte der Massenspeicherung

In FORTH wurde auf eine möglichst automatische Verwaltung der Massenspeicherung Wert gelegt. Die wesentlichen Befehlsworte sind dabei:

n LOAD Lade Screen Nr. n in den Eingabepuffer, wenn noch nicht resident, und compiliere ihren Inhalt in das Dictionary.

;S Beende Compilierung. Rückkehr zu FORTH.

--> compiliere nächste Screen (wird an das Ende eines Screens gesetzt, wenn die Anweisungsfolge noch nicht beendet ist).

n LIST Liste Screen n (ohne Compilierung).

UPDATE markiert das Flag-Byte des laufenden Block-Buffers als bearbeitet (Bit 7 wird gesetzt).

FLUSH Schreibe alle Block-Buffer, die 'updated' sind auf Massenspeicher zurück.

EMPTY-BUFFERS
Notwendige Initialisierungsprozedur, markiert alle Block-Buffer als leer, schreibt jedoch 'updated buffers' nicht auf Massenspeicher zurück.

n BLOCK Befehlswort, um den Datenblock (Sektor) n zur Verfügung zu stellen (genauer: Block n+OFFSET). Wenn er nicht resident ist, wird er hereingezogen. Block liefert die Speicheradresse auf dem Datenstack ab, an der die Daten des Datenblocks beginnen. Zum Verständnis wird die Definition dieses Wortes hier gelistet:

```
: BLOCK    ( BLOCK-# --- ADDRESS)
    OFFSET @ + >R  PREV @ DUP @ R - DUP +
    IF  BEGIN
            +BUF 0= IF DROP R BUFFER DUP R 1 R/W 2 - THEN
            DUP @ R - DUP + 0=
        UNTIL
        DUP PREV !
    THEN  R> DROP  2+  ;
```

n BUFFER liefert die Adresse eine Block-Buffers+2 (hier beginnt ja der Datenteil) ab. Wenn der vorherige Platzhalter-Block unter Adresse 'updated' war, wird er auf Diskette zurückgeschrieben und der Block-Buffer wieder zur Verfügung gestellt. BUFFER zieht jedoch keine Daten herein (s. BLOCK).

R/W ist das Interfacewort des Anwenders, das den physischen Datentransport in die eine oder andere Richtung bewirkt und das von der Umgebung abhängt. Es verlangt folgende Parameter auf dem Datenstack:

Adresse des Datenteils im Block, Block-Nr.,R/W-FLAG

R/W exekutiert die Treiberroutine des Anwenders, deren Startadresse oder Codefeldadresse in der Variablen UR/W verankert wurde. Typisch wird es sich um eine Assemblerroutine handeln, die auch die auf dem Datenstack übergebenen Parameter verwertet. Wenn R/W=0 soll geschrieben werden, wenn =1 soll gelesen werden.

+BUF setzt eine übergebene Buffer-Adresse auf die nächstfolgende um, und zwar auch zirkular, wenn LIMIT erreicht wurde.

Das FORTH-Interface für die Massenspeicherung ist damit umrissen. Es hängt damit vom Anwender ab, wie er seine Treiberroutine formuliert, deren Startadresse unter R/W gehalten wird und wie er die Parameter seines Systems initialisiert. Wichtig ist das EMPTY-BUFFERS beim Beginn der Arbeit und das FLUSH am Ende, um noch nicht abgespeicherte veränderte Blöcke herauszuschreiben. Für das Arbeiten mit Screens ist auf Vorschläge im Rockwell-Anwenderhandbuch für FORTH hinzuweisen, ebenso auf den FORTH-Editor in Heft 29 dieser Zeitschrift.

FORTH macht auch bei kleinem Speicher das Abfahren sehr großer Programme möglich. Auf die dafür erforderliche einfache Overlaytechnik werden wir bei nächster Gelegenheit eingehen.

Literatur

Die vorgenannten Definitionen wurden entnommen aus: Derick, M. und Baker, L.: FORTH Encyclopaedia, Mountain View, Ca. 1982, erschienen bei Mountain View Press. □□

FORTH-Mnemonics?

Verschiedentlich wurde schon erwähnt, daß die Sprache FORTH etwas gewöhnungsbedürftig ist, nicht nur wegen der umgekehrten polnischen Notation, sondern auch wegen ihrer Befehlsworte. Sie haben wenig Ähnlichkeit mit denen anderer Sprachen und sie lehnen sich auch nur gelegentlich an das Englische an. Die Befehlsworte sind daher wenig merkfähig, wenig mnemonisch. Oft bestehen sie nur aus einem einzigen Zeichen, so daß man fast vermuten könnte, die Sprache sei für Tippfaule kreiert.

Gleichwohl kann man für den Anfänger einige Linien ziehen, die ihm das Merken erleichtern, denn es gibt Regelmäßigkeiten in der Wortbildung. Die nachfolgenden Abschnitte zeigen sie am Beispiel häufiger benutzter Worte auf. Dabei wird nicht angestrebt, für das typische Merkmal alle Vorkommen in der Sprache aufzulisten.

Der Punkt symbolisiert die Ausgabe von Zahlen oder Zeichen

.	Ausgabe einer einfach genauen Zahl vom Datenstack als ASCII-String
D.	dito für doppelt genaue Zahl
D.R	wie D. aber nach rechts formatiert
FP.	Ausgabe einer Fließkommazahl (kein Standard)
." "	Ausgabe eines zwischen den Anführungsstrichen stehenden Strings
.S	Anzeige des Stackinhalts, unzerstörend

Das Ausrufungszeichen synbolisiert eine Abspeicherung (store)

!	Abspeicherung eines Wortes (doppelt genaue Zahl) vom Datenstack ins Memory
C!	dito für ein Byte
+!	Addiere eine Zahl zum Inhalt unter Adresse (plus store)

!CSP	(store CSP) Speichere den Stackpointer in die Variable CSP, Teil der Compiler-Sekurität
RP!	Initialisierung des Hardwarestacks aus der Variablen R0
SP!	Initialisierung des Datenstacks aus S0

Der Klammeraffe symbolisiert einen Holvorgang

@	Hole ein Wort (doppelt genaue Zahl) aus Adresse auf den Datenstack
C@	dito für 1 Byte
SP@	Hole Datenstackpointer (wie vorher) auf den Datenstack

Das Komma symbolisiert eine Abspeicherung vom Datenstack in das Dictionary

,	Ablage eines Wortes (2 Byte) in das Dictionary bei HERE
c,	dito, 1 Byte

Das Fragezeichen steht typisch für eine Prüfoperation

?	Zeige (unzerstörend) den Inhalt unter Adresse als Zahl
?TERMINAL	Setze Flag ob Keyboard betätigt oder nicht betätigt wurde
?PAIRS	Überprüfung im Compiler, ob Kontrollstrukturen wie IF ELSE THEN in der richtigen Paarigkeit/Abfolge stehen. Für den Compiler stehen etliche ähnliche Worte zur Verfügung

Die Operatoren < > und = stehen für Vergleiche und setzen ein Flag

>	Vergleich zweier Zahlen n1 und n2, TRUE wenn n1>n2
<	dito, TRUE, wenn n1<n2
U<	Vergleich zweier vorzeichenfreier 16-Bit-Zahlen
0<	unärer Vergleich, ob eine Zahl negativ ist (n<0)
0=	unärer Vergleich, ob eine Zahl 0 ist

Die Operatoren < und > können auch eine Richtung, ein Beginn/Ende anzeigen

S->D	Vorzeichenausweitung von Byte zu Wort
>R	bringt ein Wort vom Daten- zum Hardwarestack
R>	holt es zurück
<# #>	Beginn und Ende einer Zahlenumwandlung
<BUILDS DOES>	Beginn und Ende einer Befehlsfolge, die zur Compilierungszeit ausgeführt werden soll
-->	Interpretiere nächstes Screen

Der Slash steht für Division oder den Begriff 'pro'

/	Division vorzeichenbehafteter Zahlen
U/	Division vorzeichenfreier Integers
*/	'times divide', Skalierung
B/SCR	Zahl der Blöcke je Editor-Screen
B/BUF	Bytes pro Puffer in der Massenspeicherung

Die Operatoren + - und * stehen für arithmetische Operationen

*	Multiplikation zweier vorzeichenbehafteter Zahlen

M*	dito, doppelt genaues Ergebnis
D+	Addiere zwei doppelt genaue Zahlen
2+	Addiere 2 zur Zahl auf dem Stack
1-	Subtrahiere 1 von der Zahl auf dem Stack
+LOOP	Addiere eine Zahl zur Laufvariablen I einer DO .. LOOP
+BUF	Inkrementiere BUFFER-Adresse
-TRAILING	Vermindert den Längenparameter eines String um die Zahl der Blanks am Ende

S bezeichnet im allgemeinen den Datenstack

.S	Unzerstörende Anzeige des Datenstacks
SP@	Bringt den Datenstackpointer (wie er vorher war) auf den Datenstack

R steht für Hardwarestackpointer (Return Stack) oder 'Right Justified'

>R	Bringe Wort vom Daten- auf den Hardwarestack
R	Kopiert wie I das oberste Wort des Hardwarestacks auf den Datenstack
RP@	Bringe den Hardware-Stackpointer auf den Datenstack
.R	Gib rechtsbündig vom Datenstack aus (Right Justified)

U steht für 'vorzeichenfrei' oder USER

U<	Vergleich zweier vorzeichenfreier Zahlen
UB/BUF	Hinterläßt die Adresse einer Variablen, in der die Bytes per Buffer des Anwenders verankert werden

Klammern (und) bezeichnen untergeordnete FORTH-Worte

Sie werden im allgemeinen von den gleichlautenden Worten ohne Klammern in der Compilierung benutzt

Das Semikolon bezeichnet den Abschluß einer Kompilierung

;	Abschluß einer COLON-Definition
;S	Beendigung der Compilierung eines Screens
;CODE	Übergang einer COLON-Definition zu nachfolgenden Assembler-Statements, Hereinrufen des Assembler-Vokabulars

Die eckigen Klammern [und] veranlassen Stoppen und Wiederbeginn der Compilierung

[	Während der Compilierung einer COLON-Definition werden die auf die linke Klammer folgenden Worte während der Compilierungszeit direkt ausgeführt, jedoch nicht in die Definition eingebunden
]	Nach der rechten Klammer wird die Compilierung der nachfolgenden Worte wieder aufgenommen. Beispiel: : TEST [." ICH COMPILIERE"] 1 . ; führt während der Compilierung zur Ausgabe von "ICH COMPILIERE". Zur späteren Ausführungzeit druckt TEST eine 1
[COMPILE]	zwingt zur Einbindung des auf [COMPILE] folgenden IMMEDIATE-Wortes in die laufende Definition, ohne daß es als 'immediate' bereits während der Compilierung ausführt. Also: Unterdrückung der Eigenschaft 'IMMEDIATE' in der laufenden Phase.

Wolfgang Zweygart, 7030 Böblingen

Adreßkartei unter FORTH

Das nachstehende FORTH-Programm benötigt die FORTH-Stringfunktionen von Roland Löhr aus 65xx MICRO MAG, Nr. 25, Juni 1982. Das Programm läuft beim Verfasser zusammen mit dem Rockwell-DOS (deshalb der Beginn ab hex 800) und einem Videointerface der Fa. Neudecker, Berlin. Für den Monitor sind hier die Befehle SET und RES enthalten, sie schalten die AIM-Tastatur auf Groß-/Kleinschreibung um, bzw. wieder zurück. Sie können auch problemlos weggelassen werden.

Das Programm schafft sich in der Compilierungsphase selbst genügend Raum, so daß die maximale Anzahl der möglichen Adressen direkt vom verfügbaren freien RAM abhängig wird. Die Cassettenein- und Ausgabe wird vermutlich nicht mit dem in Heft 28 dieser Zeitschrift veröffentlichten REMON zusammenarbeiten. Ausgabe ist Tape 2, Eingabe ist Tape 1. Im praktischen Betrieb macht sich die Schnelligkeit von FORTH besonders nützlich bemerkbar. Bei gleichen Voraussetzungen ist das Programm um etwa 50% schneller als ein vergleichbares BASIC-Programm. Dies wird auch dadurch erreicht, daß die Länge der Datensätze fixiert wurde und daß nicht bei jedem Variablenaufruf nach deren Länge gesucht werden muß.

Das Programm kann auch als Literaturdatei benutzt werden. Dazu kann man evtl. die Länge der Felder ändern. Es kehrt nach jeder Dienstleistung über die Anzeige des Menues in die Ebene des FORTH zurück, so daß alle Dienstleistungen des FORTH und die des Programms erreicht werden können.

```
( 02.05.1983  von Wolfgang Zweygart )
FORGET TASK
1280 ALLOT ( nur bei Start ab $0800 nötig )
: TASK ;
: STRBUF ( Schafft neu Variablenart mit fester Länge )
<BUILDS
DUP C, 0 DO 32 C, LOOP
0 C, DOES> 1+ 148 ;
148 STRBUF HE$
: STR-ARRAY ( Array mit fester Länge )
<BUILDS
0 DO 150 C, 150 0 DO 32 C, LOOP 0 C,
LOOP DOES>
SWAP 152 * + 1+ 150 ;
( Dimensionierung des Feldes nach größe des Speichers )
ULIMIT @ HERE 3000 + - 152 /
STR-ARRAY DAT$
HEX
14 STRING SU$
0 VARIABLE S
0 VARIABLE N
0 VARIABLE ANZAHL
0 VARIABLE TE
: CURS-HOM 18 EMIT ; ( CURSER HOME + SCHIRM LOESCHEN )
: CR D PUT ;
: TAB ."                    " ;
: ME ( Menue anzeigen )
( nach diesem Wort wird in die Forth-Ebene zurückgekehrt )
( und dann die Dienstleistungen als Forthworte aufgerufen )
SP!
CURS-HOM
CR TAB ." Datei Programm " CR
```

```
CR TAB ."      ****  Menue **** " CR
CR TAB ." Addressen einfügen --> EI "
CR TAB ." Anzeigen           --> SH "
CR TAB ." Ausdrucken         --> DR "
CR TAB ." Addressen ändern   --> AN "
CR TAB ." Sortieren          --> SO "
CR TAB ." Addresse suchen    --> FI "
CR TAB ." Daten suchen       --> SU "
CR TAB ." Warmstart / Menue  --> ME "
CR TAB ." Backup (Cassette)  --> TO "
CR TAB ." Load (Cassette)    --> TI "
CR
;
: SET 0 A45F C! ; ( KLEIN UND GROSSCHREIBUNG )
: RES 80 A45F C! ; ( NUR GROSS )
: DISPLAY ( GIBT EINZELNE ADRESSE AUS )
CR
DECIMAL N @ . ." --> " CR HEX
N @ DAT$ HE$ S!
." NAME           "
HE$ DROP 14 WRITE
A413 C@ D = IF CR ELSE 2C PUT ENDIF
." VORNAME        "
HE$ DROP 14 + F WRITE CR
." STRASSE        "
HE$ DROP 23 + 14 WRITE CR
." PLZ  ORT       "

HE$ DROP 37 + 14 WRITE CR
." TELEFON        "
HE$ DROP 4B + F WRITE
."    GEBURTSTAG  "
HE$ DROP 5B + A WRITE CR
."  TEXT            "
HE$ DROP 66 + 2B WRITE CR
;
: DRUCK ( ADD auf Drucker )
50 A413 C! DISPLAY ;
: SORT2 ( Stringtausch für SORT )
N @ 1- DAT$ N @ DAT$ $SWAP
;
: SORT3 ( Unterprogramm für SORT )
0 TE !
ANZAHL @ 2 DO I N !
N @ DAT$ N @ 1- DAT$ $COMP 1 = IF SORT2 1 TE ! ENDIF LOOP
;
: SO ( Sortierung der Adressen )
." Bitte warten "
1 TE ! BEGIN TE @ 1 = WHILE SORT3 REPEAT
( Garbage collection entfernt leere Strings )
ANZAHL @ 1 DO I DAT$ DROP C@ 20 = IF LEAVE
I ANZAHL ! ENDIF LOOP
;
: NAME ." NAME " HE$ DROP 13 EXPECT ." OK " CR ;
: VN ." VN    " HE$ DROP 14 + 13 EXPECT ." OK " CR ;
: ST ." ST    " HE$ DROP 23 + 13 EXPECT ." OK " CR ;
: ORT ." ORT  " HE$ DROP 37 + 13 EXPECT ." OK " CR ;
: TELE ." TELE " HE$ DROP 4B + F EXPECT ." OK " CR ;
```

```
: GEB ." GEB  " HE$ DROP 5B + A EXPECT ." OK " CR ;
: TEXT ." TEXT " HE$ DROP 66 + 2B EXPECT ." FERTIG " ;
: CONTR ( entfernt $0 aus Eingabestring )
HE$ BOUNDS DO
I C@ 0 = IF 20 I C! THEN
LOOP
0 HE$ + C!
;
: EI ( Eingabe von Adressen )
CURS-HOM
BEGIN CR
." Eingabe der Adresse ( Schreibmaschinentast.) "
CR SET HE$ 2- BLANKS
NAME VN ST ORT TELE GEB TEXT CONTR
RES
ANZAHL @ N ! 1 ANZAHL +!
HE$ N @ DAT$ S!
." NOCH EINE ADRESSE --> J " KEY
4A = NOT UNTIL SO ME
;
: FINDE ( Suche nach erstem Buchstaben )
0 S ! ." GIB EINEN BUCHSTABEN EIN ! "
KEY TE !
ANZAHL @ 1 DO I DUP N ! DAT$ DROP C@ TE C@ = IF DISPLAY
." RICHTIG ?? " KEY 4A = IF N @ S ! LEAVE ELSE
." SUCHE WEITER " ENDIF
ENDIF LOOP
;
: SU ( Suche nach beliebiger Zeichenkette )

0 S !
." Gib das gesuchte ein !! -- > "
SET SU$ DROP 14 EXPECT
ANZAHL @ 1 DO I DUP N ! DAT$
SU$ INSTR IF DROP DISPLAY ." RICHTIG ?? " KEY 4A =
IF LEAVE ELSE ." SUCHE WEITER " ENDIF ENDIF
LOOP RES
ME
;
: DR ( Ausdruck aller Adressen auf Drucker )
0 ANZAHL @ 1- DO I N ! DRUCK -1 +LOOP D A413 C! ME ;
: SH ( Anzeige aller Adressen )
DECIMAL CR
." Anzahl der Sätze " ANZAHL @ . CR 1 N !
0 ANZAHL @ 1- DO I N !
BEGIN ?TERMINAL NOT UNTIL
DISPLAY -1 +LOOP HEX
." OK ??? " KEY DROP
ME
;
: LOSCH ( Entfernt Adresse aus Datei )
HE$ BLANKS CONTR
HE$ N @ DAT$ S!
SO
;
: ANDRE
SET
CURS-HOM CR DISPLAY CR
```

```
N @ DAT$ HE$ S!
." 1 - NAME " CR ." 2 - VORNAME " CR ." 3 - STRASSE " CR
." 4 - ORT " CR ." 5 - TELEPHON " CR ." 6 - GEBOREN " CR
." 7 - TEXT " CR
KEY DUP 2DUP 2DUP DUP
31 = IF NAME ENDIF
32 = IF VN ENDIF
33 = IF ST ENDIF
34 = IF ORT ENDIF
35 = IF TELE ENDIF
36 = IF GEB ENDIF
37 = IF TEXT ENDIF
CONTR HE$ N @ DAT$ S!
DISPLAY RES ." OK ??? " KEY DROP ;
: FI FINDE ME ;
: AN ( Änderung einzelner Adressen )
FINDE
S @ 0 = NOT IF
S @ N ! CURS-HOM DISPLAY CR
." WAS SOLL GEÄNDERT WERDEN " CR
." 1 --> LOESCHEN " CR
." 2 --> TEILE ÄNDERN " CR
KEY DUP
31 = IF LOSCH ENDIF
32 = IF ÄNDRE ENDIF
ELSE ." Nichts gefunden "
ENDIF ME ;
: DATEI ( Kaltstart des Programms )
CURS-HOM
1 N ! 2 ANZAHL !
ME ;
( Unterprogramme für Cassetten-I/O )

( aus Forth-Handbuch von Rockwell )
" DATEN" $CONST NAME$
CODE INCODE XSAVE STX,
E32F JSR,
XSAVE LDX,
NEXT JMP,
END-CODE
CODE CODEOUT
XSAVE STX,
E56F JSR,
XSAVE LDX,
NEXT JMP,
END-CODE
: SETNAME NAME$ DROP A42E 5 CMOVE ;
: SETOUT 54 A413 C! 1 C@ A435 C! ;
: SETIN 54 A412 C! 0 C@ A434 C! ;
: T-OFF A800 DUP C@ CF AND SWAP C! ;
: OPENOUT SETNAME SETOUT CODEOUT ;
: OPENIN SETNAME SETIN INCODE ;
: FDUMP OPENOUT WRITE CLOSE T-OFF ;
: FLOAD OPENIN READ CLOSE T-OFF ;
" !!#$%&'()" $CONST END$
: TO ( Ausgabe auf Cassette )
40 A409 C!
ANZAHL @ 1 DO I DAT$ FDUMP LOOP
```

```
END$ HE$ S!
HE$ FDUMP ME ;
: TI ( Einlesen von Cassettte )
ANZAHL @ N !
BEGIN 1 N +! HE$ FLOAD
HE$ TYPE
HE$ END$ INSTR
0 = WHILE DROP
HE$ N @ DAT$ S!
REPEAT N @ ANZAHL ! ME ;
FINIS
```

□□

Dr. Helmut Mörtl, A-5280 Braunau

FORTH-Disassembler

Für das AIM-FORTH wurde ein Programm entwickelt, das nahezu alle FORTH-Worte rückübersetzen kann. Es ist dabei ein einfaches Disassemblerprogramm eingebaut, das den Disassembler des AIM mitbenutzt. Das Programm kann damit CODE-Definitionen und COLON-Definitionen, die Assemblerteile (;CODE) aufweisen, übersetzen. Wurden Worte durch DEFINING-Words (Datentypen zwischen <BUILS und DOES>) definiert, so wird auch das angegeben. Ebenso werden Inhalte von Variablen, Konstanten und User-Variablen angegeben.

Das Programm ist so geschrieben, daß es auf beliebigen Rechnern mit FIG-FORTH laufen kann. Allerdings ist dann der Disassembler nicht zu verwenden, man muß dann auf einen solchen für die 6502-CPU unter Punkt 3 zurückgreifen.

1. Hilfsprogramme

a) Die CASE-Struktur

Dieses Unterprogramm ist neben seiner Übersichtlichkeit beim Programmieren noch ein schönes Beispiel für die Mächtigkeit von FORTH:

```
0 ( THE "CASE" STRUCTURE                                    PB-801218 )
1
2 : CASE      ?COMP   CSP @ !CSP    6 ; IMMEDIATE          ( START *)
3
4 : OF        6 ?PAIRS   COMPILE OVER   COMPILE = ( START OF ALT *)
5             COMPILE 0BRANCH HERE 0 , COMPILE DROP  7 ; IMMEDIATE
6
7 : ENDOF     7 ?PAIRS   COMPILE BRANCH     HERE 0 ,   ( ENDS ALT *)
8             SWAP   2   [COMPILE] ENDIF    6 ;  IMMEDIATE
9
10 : ENDCASE  6 ?PAIRS   COMPILE DROP       ( ENDS THE STRUCTURE *)
11            BEGIN    SP@   CSP @ =  0=
12            WHILE    2  [COMPILE]   ENDIF    REPEAT
13            CSP  !   ;  IMMEDIATE
14 ;S
15
```

Mit einem einfachen Programm soll die Anwendung der CASE-Struktur gezeigt werden:

```
: CASE-TEST CASE 0 OF ." NICHTS " CR  ENDOF
                 1 OF ." EINS   " CR  ENDOF
                 ." MEHR ODER WENIGER" CR
            ENDCASE ;
```

Das Programm CASE-TEST soll getestet und mit dem noch zu besprechenden Programm UN: dekodiert werden:

```
0 CASE-TEST
NICHTS
OK

1 CASE-TEST
EINS
OK

2 CASE-TEST
MEHR ODER WENIGER
OK

UN: CASE-TEST
CASE-TEST
3756      0
3758      OVER
375A      =
375C      0BRANCH 3772
3760      DROP
3762      (.") NICHTS
376C      CR
376E      BRANCH 37A6
3772      1
3774      OVER
3776      =
3778      0BRANCH 378E
377C      DROP
377E      (.") EINS
3788      CR
378A      BRANCH 37A6
378E      (.") MEHR ODER WENIGER
37A2      CR
37A4      DROP
37A6      ;S
```

b) U<

Das FORTH Standard-Wort versagt beim Vergleich von zwei 16-Bit-Worten, da das negative Vorzeichen berücksichtigt wird. Es wird bei Adreßvergleichen benötigt.

```
0 ( U<                          AEDERUNG FUER 16-BIT ADRESS-VERGLEICH*)
1
2 : U<   ( N1 N2 --- F)                          ( U1 < N2 ODER N1 < U2*)
3     OVER OVER 0< SWAP 0< XOR            ( VERSCHIEDENES VORZEICHEN)
4     IF   SP@ 3 + C@ SP@ 3 + C@ < ROT ROT 2DROP
5     ELSE     - 0<                              ( GLEICHES VORZEICHEN)
6     THEN  ;
7
```

2. Hauptprogramm

Zur Feststellung der Art eines Wortes werden die Codefeld-Adressen einiger Worte benötigt. Diese werden automatisch gesucht und auf Worte (gekennzeichnet mit .CFA) abgespeichert, die mit dem DEFINING-WORD CFA-C definiert werden. CFA-C wandelt den Stack-Wert pfa in cfa um (–2) und speichert ihn ab. Beim Aufruf von z.B. LIT.CFA wird über die Adresse der CFA-Wert von LIT mit @ auf den Stack gelegt.

LIT.NFA ist eine Konstante, die im Wort SUCH eine Directorysuche begrenzen soll (Zeile 4). Die Konstanten COUNT.M und SAVPC.M sind Monitoradressen für den Assembler DISASS (Zeile 18), der zuerst die Adresse des zu disassemblierenden Befehls auf SAVPC abspeichert und der Monitorroutine jeweils eine Disassemblerzeile ausführen läßt (1 COUNT.M !).

Die Unterprogramme ab Zeile 23: ADR. WRITELN und STRNG dienen zum Drucken einer decodierten Zeile einer COLON-Definition. ?LIT ?CPLE ?STOP ?BRA ?STRING dienen der Erkennung von Wortarten. SUCH-NFA und SUCH sind Unterprogramme zur Suche im Directory, z.B. um Beginn- und Endadresse einer Assemblerroutine zu finden, oder um ein DEFINING-Word auszudrucken (SUCH).

DECOD ist das Decodierprogramm für eine COLON-Definition. Der Aufruf des Hauptprogrammes UN: erfolgt mit UN: NAME und unterscheidet zwischen Worten, die als VARIABLE , CONSTANT , USER , DEFINING-WORDs , CODE- und COLON-Definitions definiert worden sind. Ist ein Wort nicht in diese Kategorien einzuordnen, so wird der Name des Wortes, auf die die CFA hinweist, angegeben. Die Codefeldadresse (CFA) verweist immer auf den Beginn einer Assemblerroutine. Solche sind nicht immer als FORTH-Worte im Directory gekennzeichnet (Worte ohne Header). Ein Verweis auf eine solche 'isolierte' Assemblerroutine ergibt daher bei der Decodierung eine falsche Referenz in 'WORD DEFINED BY:'. Es wird der Name ausgedruckt, der im Speicher vor dem Beginn der Assemblerroutine steht.

```
0 ( UNTHREATE  UN:                                                    )
1 HEX
2 : CFA-C      <BUILDS 2- , DOES> @ ;                ( CFA-CONSTANT*)
3
4   ' LIT NFA CONSTANT   LIT.NFA          ' CLIT CFA-C      CLIT.CFA
5       ' LIT CFA-C      LIT.CFA       ' COMPILE CFA-C  COMPILE.CFA
6        ' ;S CFA-C       ;S.CFA       ' (;CODE) CFA-C  (;CODE).CFA
7    ' (LOOP) CFA-C    (LOOP).CFA      ' (+LOOP) CFA-C  (+LOOP).CFA
8   ' 0BRANCH CFA-C  0BRANCH.CFA        ' BRANCH CFA-C   BRANCH.CFA
9      ' (.") CFA-C     (.").CFA          ' :   12 + CONSTANT :.CFA
10  ' USER 4 + CONSTANT USER.CFA  ' CONSTANT 8 + CONSTANT CST.CFA
11 ' (DOES>) CONSTANT (DOES>).CFA  ' VARIABLE 4 + CONSTANT VAR.CFA
12
13      A419 CONSTANT  COUNT.M A425 CONSTANT SAVPC.M     ( MONITOR )
14
15 CODE    DISAS-JSR        ( JSR TO   AIM DISASSEMBLER SUBROUTINE*)
16           E72B JSR,     NEXT JMP,       END-CODE
17
18 : DISASS ( FROM TO --- )       SWAP   SAVPC.M ! ( DISASSEMBIERE*)
19     BEGIN
20       1 COUNT.M ! DISAS-JSR  DUP SAVPC.M @ SWAP U< ?TERMINAL OR
21     UNTIL ;
22
23 : ADR.     ( PFA --- CFA )            ( DRUCKE ADRESSE U. INHALT*)
24     CR DUP DUP 0 4 D.R   @ DUP 0 5 D.R SPACE       ;
25
26 : ?LIT   ( CFA --- CFA F )   DUP    @ LIT.CFA =     ; ( 1 BYTE? *)
27
28 : ?CPLE  ( CFA --- CFA F )  DUP    @ COMPILE.CFA = ; ( COMPILE?*)
29
30 : WRITELN ( PFA --- CFA )                    ( DRUCKE EINE ZEILE*)
31     ADR. DUP CLIT.CFA =
32     IF    ." CLIT " DROP 2+ DUP C@ 0 D. 1-
33     ELSE 2+  NFA ID.
34     THEN   ;
35
```

```
93          :.CFA       OF  DECOD                              ENDOF
94      CR ." WORD DEFINED BY : "     SUCH
95      ENDCASE       BASE ! ;
96   DECIMAL    ;S
```

Beispiele zu UN:

Das Programm UN: wird folgendermaßen aufgerufen:

```
UN: .S
.S
C9FA B81B S0
C9FC B6F7 @
C9FE B514 SP@
CA00 BA26 -
CA02 B801 2
CA04 C3EC /
CA06 B696 DUP
CA08 B7F9 1
CA0A BA71 >
CA0C B0EB 0BRANCH CA22
CA10 B7F9 1
CA12 B173 (DO)
CA14 B18C I
CA16 BA92 PICK
CA18 B2BD CR
CA1A C967 .
CA1C B10C (LOOP) CA14
CA20 B55B ;S
```

Zum Vergleich ein Speicher-Dump:

```
' .S NFA 10 DUMP
NFA     NAME      LFA    CFA VERWEIST AUF ':'
 |     /------\  /---\  /---\
59F3  82 2E D3 E7 C9 56 B7 1B  [illegible]
59FB  B8 F7 B6 14 B5 26 BA 01  [illegible]
```

Aus CFA ($B756) ist ersichtlich, daß es sich hier um eine COLON-Definition handelt, und zwar zeigt CFA auf den Assembler-Teil von ':' .

```
UN: :
:     IMMEDIATE
B744 BB7B ?EXEC
B746 BB3B !CSP
B748 B923 CURRENT
B74A B6F7 @
B74C B916 CONTEXT
B74E B71D !
B750 C020 CREATE
B752 BBE2 ]
B754 BC2F (;CODE)
B756   A5 A0        LDA   $A0
B758   48          PHA
B759   A5 9F        LDA   $9F
B75B   48          PHA
B75C   18          CLC
B75D   A5 A2        LDA   $A2
```

```
36 : ?STOP   ( CFA --- CFA F )     ( ENDE EINER COLON DEFINITION? *)
37     DUP   @  DUP ;S.CFA =  OVER (;CODE).CFA = OR SWAP DROP ;
38
39 : ?BRA    ( CFA --- CFA F )    DUP  @ DUP              ( BRANCH? *)
40     (LOOP).CFA = OVER (+LOOP).CFA = OR OVER
41     BRANCH.CFA = OR OVER 0BRANCH.CFA = OR SWAP DROP ;
42
43 : ?STRING ( CFA --- CFA F ) DUP   @ (.").CFA =       ; ( TEXT? *)
44
45 : STRNG   ( CFA --- CFA+2 )                        ( DRUCKE STRING*)
46     2+ DUP COUNT DUP  >R TYPE R> 1- + ;
47
48 : CODE.C  ( PFA CFA@ --- PFA CFA@ PFA ) OVER  ;  ( CODE-DEF. ?*)
49
50 : SUCH-NFA ( ADR --- NFA-VOR NFA  )   ( SUCH WORD ZUR ADRESSE *)
51     LATEST   SWAP OVER
52     BEGIN  ROT DROP DUP ROT ROT                    ( NFA ADR NFA )
53        PFA LFA @
54        DUP 3 PICK OVER OVER 1+ U< ROT ROT
55        SWAP 1000 + U<            ( SPRUNG VON UND NACH USER-ROMS)
56        AND  ?TERMINAL OR
57     UNTIL     SWAP DROP ;
58
59 : SUCH  ( ADR --- )      ( - NAME, DESSEN NFA < ADR IST          *)
60
61     DUP CFFF ( MIT MATH-P.: D2C0) U< OVER
62     LIT.NFA  SWAP U< AND
63     IF    SUCH-NFA ."    " ID. CR DROP
64     ELSE  ." AUSSER DIR-BEREICH" CR DROP
65     THEN  ;
66
67 : DECOD   ( PFA --- )              ( DECODIERE COLON-DEFINITION*)
68     BEGIN
69        WRITELN     ?STOP 0= ?TERMINAL 0= AND
70     WHILE
71       ?LIT IF 2+ ADR. DROP ELSE ( PRINT ADR. MIT "LIT" UND BYTE)
72       ?BRA IF 2+ DUP   @ OVER + 0 D. ELSE ( BERECHNETER  BRANCH)
73       ?CPLE IF 2+ DUP   @  2+  NFA ID. ELSE      ( COMPILED WORD)
74       ?STRING IF STRNG                                  ( TEXT)
75       THEN THEN THEN THEN
76       2+                                  ( NEXT PARAMETER-ADR.)
77     REPEAT
78     DUP   @ (;CODE).CFA =         ( DISASSEMBLIERE ANSCHLIESSEND)
79     IF    2+ DUP  SUCH-NFA DROP DISASS
80     ELSE  DROP
81     THEN  ;
82
83 : UN:                    ( HAUPTPROGRAMM    AUFRUF:    UN: NAME *)
84     DUP NFA DUP CR ID.
85     C@ 40 AND IF  ."  IMMEDIATE " THEN CR
86     BASE @ SWAP HEX DUP CFA   @
87     CASE VAR.CFA      OF CR ." VARIABLE    "  @ 0 D.          ENDOF
88          CST.CFA      OF CR ." CONSTANT    "  @ 0 D.          ENDOF
89          USER.CFA     OF CR ." USER      " C@  200 + @ 0 D. ENDOF
90          (DOES>).CFA OF CR ." DEFINING WORD" DROP            ENDOF
91          CODE.C       OF CR ." CODE DEFINITION "
92                          DUP  SUCH-NFA DROP DISASS          ENDOF
```

```
B75F   69 02       ADC  #$02
B761   85 9F       STA   $9F
B763   98          TYA
B764   65 A3       ADC   $A3
B766   85 A0       STA   $A0
B768   4C 5A B0    JMP   $B05A
```

Schon das nächste Wort im Directory, MON , ist eine CODE-Definition, hier verweist der Inhalt von CFA auf PFA:

```
UN: MON
MON
   CODE DEFINITION
C9EF   00          BRK
C9F0   4C 5A B0    JMP   $B05A
```

Das Wort CONSTANT ist eine COLON-Definition und besitzt einen Assembler-Teil, der durch das Wort :CODE gekennzeichnet wird.

```
UN: CONSTANT
CONSTANT
B78A C020 CREATE
B78C BBF6 SMUDGE
B78E BA05 ,
B790 BC2F (;CODE)
B792   A0 02       LDY  #$02
B794   B1 A2       LDA  ($A2 ),Y
B796   48          PHA
B797   C8          INY
B798   B1 A2       LDA  ($A2 ),Y
B79A   4C 53 B0    JMP   $B053
```

Das Wort DISASS kann natürlich auch zur Diassemblierung beliebiger anderer Assemblerteile verwendet werden:

```
HEX B000 B009 DISASS
B000   4C 79 C2    JMP   $C279
B003   4C 82 C2    JMP   $C282
B006   0D 01 BD    ORA   $BD01
```

Sicher wird der Leser schnell weitere nützliche Anwendungen des Programms UN: finden, und der Autor hofft, daß es ihm eine unentbehrliche Hilfe bei Programmieren in FORTH werden wird!

3. Allgemeiner Disassembler

```
0 ( DISASSEMBLER                                                    )
1   DECIMAL   5 CONSTANT B/CODE
2            12 CONSTANT CODE/L       CODE/L 14 * CONSTANT CODE/SCR
3   SCR @  4 + CONSTANT SCR.COD-0                  HEX
4
5 : CODE-ADR  ( CODE --- CD-ADR )                ( FINDE CODE IM SCR#)
6     CODE/SCR /MOD SWAP                         ( SCR-OFFS NR-IM-SCR)
7     CODE/L /MOD 1+ ROT SCR.COD-0 +             ( NR-IM-SCR LINE SCR)
8     (LINE) DROP SWAP B/CODE * +  ;             ( --- CD-ADR)
9
10 : .BEFEHL  ( CD-ADR --- CD-ADR )               DUP 3 TYPE ;
11 : ?BYTE    ( CD-ADR --- CD-AD N ) DUP 4 + C@ F AND 1 MAX 3 MIN ;
```

```
12 : ?MODE       ( CD-ADR --- MODE )      3 + C@ F DIGIT DROP ;
13 : .2BYT S->D <# # # # # #> TYPE SPACE  ;
14 : .1BYT S->D <# # # #>      TYPE SPACE  ;
15 : .1BYT+SP .1BYT 2 SPACES ;
16 : .MODE  ( ADR CD-ADR ASS-PARAM --- ADR CD-ADR) ( PRINT PARAM.*)
17       OVER ?MODE
18   CASE 1 OF ."   " DROP                    ENDOF          ( IMPLIED )
19        2 OF ."  ($" .1BYT ." ),Y"          ENDOF           ( [ZP],Y )
20        3 OF ."  ($" .1BYT ." ,X)"          ENDOF           ( [ZP,X] )
21        4 OF DUP 7F > IF FF XOR 1+ MINUS THEN
22             3 PICK + 2+ ."    $" .2BYT    ENDOF          ( BRANCH )
23        5 OF ."  #$" .1BYT                  ENDOF          ( IMMMID )
24        6 OF ."   $" .1BYT                  ENDOF              ( ZP )
25        7 OF ."   $" .1BYT ." ,Y"           ENDOF            ( ZP,Y )
26        8 OF ."   $" .1BYT ." ,X"           ENDOF            ( ZP,X )
27        9 OF ."   $" .2BYT                  ENDOF          ( EXTEND )
28        A OF ."  ($" .2BYT  ." )"           ENDOF        ( [EXTEND] )
29        B OF ."   $" .2BYT ." ,X"           ENDOF          ( EXT,X )
30
31        C OF ."   $" .2BYT   ." ,Y"         ENDOF           ( EXT,Y )
32        0 OF ."  ? "  EMIT ."    "          ENDOF   ( KEIN ASS.-CODE)
33    ENDCASE  ;
34
35 : PRINT  ( ADR --- NEXT-ADR )            ( DRUCKE DISASS-ZEILE*)
36    CR DUP .2BYT SPACE
37    DUP   C@ CODE-ADR
38    ?BYTE      3 PICK   C@ SPACE .1BYT
39    CASE 2 OF OVER 1+   C@ DUP .1BYT+SP   ENDOF
40         3 OF OVER 1+   @  DUP .2BYT      ENDOF
41         1 OF 5 SPACES OVER   C@          ENDOF
42    ENDCASE  4 SPACES SWAP
43    .BEFEHL SPACE SWAP  .MODE
44    ?BYTE SWAP DROP +   ;
45
46
47 : DISASS  ( FROM TO --- )                    ( DISASSEMBLER   *)
48    SWAP
49    BEGIN  PRINT
50       DUP 3 PICK U< ?TERMINAL 0= AND
51    WHILE
52    REPEAT
53    DROP DROP ;
54 ;S

 0 ( INSTRUCTION CODE TABLE  14 ZEILEN/SCR  12 CODES/ZEILE          )
 1 BRK11ORA32-     -     -     ORA62ASL62-     PHP11ORA52ASL11-
 2 -     ORA93ASL93-     BPL42ORA22-     -     -     ORA82ASL82-
 3 CLC11ORAC3-     -     -     ORAB3ASLB3-     JSR93AND32-     -
 4 BIT62AND62ROL62-      PLP11AND52ROL11-      BIT93AND93ROL93-
 5 BMI42AND22-     -     -     AND82ROL82-     SEC11ANDC3-     -
 6 -     ANDB3ROLB3-     RTI11EOR32-     -     -     EOR62LSR62-
 7 PHA11EOR52LSR11-      JMP93EOR93LSR93-      BVC42EOR22-     -
 8 -     EOR82LSR82-     CLI11EORC3-     -     -     EORB3LSRB3-
 9 RTS11ADC32-     -     -     ADC62ROR62-     PLA11ADC52ROR11-
10 JMPA3ADC93ROR93-      BVS42ADC22-     -     -     ADC32-     -
11 SEI11ADCC3-     -     -     ADCB3-     -     -     STA32-     -
12 STY62STA62STX62-      DEY11-     TXA11-     STY93STA93STX93-
13 BCC42STA22-     -     STY82STA82STX72-      TYA11STAC3TXS11-
```

```
14 -     STAB3-      -     LDY52LDA32LDX52-     LDY62LDA62LDX62-
15
16 TAY11LDA52TAX11-      LDY93LDA93LDX93-     BCS42LDA22-      -
17 LDY82LDA82LDX72-      CLV11LDAC3TSX11-     LDYB3LDAB3LDXC3
18 CPY52CMP32-      -    CPY62CMP62DEC62-     INY11CMP52DEX11-
19 CPY93CMP93DEC93-      BNE42CMP22-      -    -     CMP82DEC82-
20 CLD11CMPC3-      -    -     CMPB3DECB3-    CPX52SBC32-      -
21 CPX62SBC62INC62-      INX11SBC52NOP11CPX93SBC93SBC93INC93-
22 BEQ42SBC22-      -    -     SBC82INC82-    SED11SBCC3-      -
23 -     SBCB3INCB3-
24                                                              □□
```

Peter W. Arps, 2000 Hamburg 73

INPUT Window (CBM)

Nachfolgend ist eine Eingaberoutine beschrieben, mit der man ein Bildschirmfenster für INPUT schafft.Es ist für CBM-Rechner mit Bildschirm von 40 Zeichen pro Zeile bestimmt, hier speziell für BASIC 3 (Anm.d. Hrsg.: Im Anhang auch Adressen für BASIC 4).Der Anwender ist in der Lage, einen oberen und einen unteren Rand seines Eingabefensters mit POKE zu bestimmen, siehe Listing, Zellen 218 und 219. Ebenso wird in linker und ein rechter Rand (Bildschirmspalte) durch POKE nach 214 und 215 bestimmt. Damit ist es möglich, in den übrigen Bildschirm Information zur Bedienerführung einzuschreiben (z.B. Vorspalte mit Name, Ort usw.), ohne daß diese die Eingabe stört. Auch die Übernahme ganzer Bildschirminhalte ist möglich.

Das Programm wird mit SYS (Adresse des ENTRY-Punktes)(V1$ ... Vn$) von BASIC her aufgerufen, wobei die Vn$ der Menge der Zeilen im Bildschirmfenster entsprechen. Je Zeile mit 40 Zeichen wird also eine Variable gefüllt, und zwar wenn man die Eingaberoutine mit SHIFT/RETURN verläßt. Die Funktionen 'HOME' und 'CLR' wirken sich während der Eingabe nur auf das Fenster aus. Scrolling findet nicht statt, weil beim Erreichen des unteren Fensterrandes wieder zum oberen Rand gesprungen wird. Während der Eingabe kann man für Korrekturen beliebig im Bildschirmfenster herumfahren.

```
0010              .LS
0020              ;**************************
0030              ;*                        *
0040              ;*     INPUT-WINDOW       *
0050              ;*                        *
0060              ;*  FULL SCREEN INPUT     *
0070              ;*                        *
0080              ;**************************
0090              ;
0100              .BA $7000
0110              ;
0120 FREE         .DE $30
0130 VAR.ADR      .DE $44
0140 CHRGOT       .DE $70
0150 BF.ANZ       .DE $9E
0160 INDX         .DE $A1
0170 LSXP         .DE $A3
0180 LSTP         .DE $A4
0190 FLG.CURS     .DE $A7        ;CURSOR EIN/AUS
0200 GDBLN        .DE $A9
0210 BLK.CURS     .DE $AA        ;CURSOR BLINKT EIN/AUS
0220 CRSW         .DE $AC
```

```
                0230 Y.COUNT     .DE $B8
                0240 ZEIL.CNT    .DE $B9
                0250 ZDATA       .DE $BA
                0260 SCREEN      .DE $C4
                0270 SPALTE      .DE $C6
                0280 LINE.MX     .DE $D5
                0290 LI.RAND     .DE $D6        ;LINKE BEGRENZUNG
                0300 RE.RAND     .DE $D7        ;RECHTE BEGRENZUNG
                0310 ZEILE       .DE $D8
                0320 DATA        .DE $D9
                0330 ZL.VON      .DE $DA        ;OBERE BEGRENZUNG
                0340 ZL.BIS      .DE $DB        ;UNTERE BEGRENZUNG
                0350 LDTB1       .DE $E0
                0360             ;
                0370 HOL.VAR     .DE $CC9F
                0380 VAR.SUCH    .DE $CFC9
                0390 KL.AUF      .DE $CDF5
                0400 KL.ZU       .DE $CDF2
                0410 KOMMA       .DE $CDF8
                0420 STR.TEST    .DE $CC90
                0430 SET.POINT   .DE $D3CE
                0440 SET.CURS    .DE $E25D
                0450 PRT.CHR     .DE $E3D8
                0460 DSPP        .DE $E6EA
                0470 LP2         .DE $E285
                0480             ;
                0490             ;
                0500 ;-------------------------------
                0510 ;--                           --
                0520 ;-- POKE214,1..39 LINKER R.   --
                0530 ;-- POKE215,2..40 RECHTER R.  --
                0540 ;-- POKE216,1..24 OBERER R.   --
                0550 ;-- POKE219,1..24 UNTERER R.  --
                0560 ;--                           --
                0570 ;-- FUER FULL-SCREEN-INPUT    --
                0580 ;-- 'ZL.VON' AUF 1 UND        --
                0590 ;-- 'ZL.BIS ' AUF 24 SETZEN   --
                0600 ;--                           --
                0610 ;-------------------------------
                0620             ;
7000- 20 F5 CD  0630 ENTRY       JSR KL.AUF    ;SYS(ADR)(V1$,...,VN$)
7003- 20 9F CC  0640 VAR.LOOP    JSR HOL.VAR   ;VARIABLE HOLEN
7006- 20 C9 CF  0650             JSR VAR.SUCH          ;GGF. EINRICHTEN
7009- 20 90 CC  0660             JSR STR.TEST          ;STRING-VARIABLE?
700C- A5 B9     0670             LDA *ZEIL.CNT
700E- C5 D2     0680             CMP *ZL.BIS   ;ERSTER AUFRUF?
7010- 90 08     0690             BCC NO.FIRST          ;NEIN
7012- 20 73 70  0700             JSR INIT.INPT TASTATUR-EINGABE
7015- A6 DA     0710             LDX *ZL.VON
7017- CA        0720             DEX
7018- 86 B9     0730             STX *ZEIL.CNT
701A- A9 28     0740 NO.FIRST    LDA #40       ;MAX.LAENGE
701C- 20 CE D3  0750             JSR SET.POINT         ;ADR IN $30
701F- A6 B9     0760             LDX *ZEIL.CNT
7021- A0 00     0770             LDY #0
7023- 20 55 70  0780             JSR LESEN     ;ZEILE 1 ... 24
7026- E6 B9     0790             INC *ZEIL.CNT
7028- A5 B5     0800             LDA *Y.COUNT          ;LAENGE
```

```
702A- A0 00     0810             LDY #0
702C- 91 44     0820             STA (VAR.ADR),Y      ;STORE LAENGE
702E- C8        0830             INY
702F- A5 30     0840             LDA *FREE    ;ADR LOW
7031- 91 44     0850             STA (VAR.ADR),Y
7033- C8        0860             INY
7034- A5 31     0870             LDA *FREE+1  ;ADR HIGH
7036- 91 44     0880             STA (VAR.ADR),Y
7038- 20 76 00  0890             JSR CHRGOT+6 ;
703B- C9 2C     0900             CMP #',
703D- F0 03     0910             BEQ NEXT.VAR
703F- 4C F2 CD  0920             JMP KL.ZU    ;')'
                0930             ;
7042- 20 F8 CD  0940 NEXT.VAR    JSR KOMMA
7045- B0 BC     0950             BCS VAR.LOOP
                0960             ;
7047- A6 DA     0970 CURS.HOME   LDX *ZL.VON  ;1.ZEILE
7049- CA        0980             DEX
704A- A5 D6     0990 POS.CURS    LDA *LI.RAND          ;1. SPALTE
704C- 85 C6     1000             STA *SPALTE
704E- C6 C6     1010             DEC *SPALTE
7050- 86 D8     1020             STX *ZEILE
7052- 4C 5D E2  1030             JMP SET.CURS
                1040             ;
7055- 20 4A 70  1050 LESEN       JSR POS.CURS
7058- 85 BA     1060 LES1        STA *ZDATA
705A- 20 D8 70  1070             JSR INPUT    ;HOL ZEICHEN
705D- 91 30     1080             STA (FREE),Y
705F- C8        1090             INY
7060- C9 0D     1100             CMP #13
7062- D0 F4     1110             BNE LES1
7064- 88        1120             DEY  ;KORREKTUR
7065- C6 A1     1130             DEC *INDX
7067- D0 07     1140             BNE LES2     ;KEINE LEERZEILE
7069- A5 BA     1150             LDA *ZDATA   ;VORHERIGER WERT
706B- C9 20     1160             CMP #'
706D- D0 01     1170             BNE LES2     ;KEINE LEERZEILE
706F- 88        1180             DEY  ;KORREKTUR
7070- 84 B8     1190 LES2        STY *Y.COUNT
7072- 60        1200             RTS
                1210             ;
7073- 20 47 70  1220 INIT.INPT   JSR CURS.HOME         ;TASTATUR-EINGABE
7076- 18        1230             CLC
7077- 90 08     1240             BCC LOOP3    ;IMMER
                1250             ;
7079- A9 9D     1260 CRS.L       LDA #157     ;CRS LINKS
707B- 2C        1270             .BY $2C
707C- A9 1D     1280 CRS.R       LDA $29      ;CRS RECHTS
                1290             ;
707E- 20 D8 E3  1300 LOOP4       JSR PRT.CHR
7081- A4 D8     1310 LOOP3       LDY *ZEILE   ;UNTERE KANTE?
7083- C4 DB     1320             CPY *ZL.BIS  ;JA - AUTOM. HOME
7085- B0 EC     1330             BCS INIT.INPT
7087- A4 C6     1340             LDY *SPALTE
7089- C8        1350             INY  ;KORREKTUR
708A- C4 D6     1360             CPY *LI.RAND          ;LINKE KANTE?
708C- 90 EE     1370             BCC CRS.R    ;JA
708E- 88        1380             DEY
```

```
708F- C4 D7     1390            CPY *RE.RAND          ;RECHTE KANTE
7091- B0 E6     1400            BCS CRS.L     ;JA
7093- A5 9E     1410            LDA *BF.ANZ   ;AUF EINGABE
7095- 85 A7     1420            STA *FLG.CURS         ;WARTEN
7097- F0 E8     1430            BEQ LOOP3
7099- 78        1440            SEI
709A- A5 AA     1450            LDA *BLK.CURS
709C- F0 09     1460            BEQ LP21
709E- A5 A9     1470            LDA *GDBLN
70A0- A0 00     1480            LDY #0
70A2- 84 AA     1490            STY *BLK.CURS
70A4- 20 EA E6  1500            JSR DSPP
70A7- 20 85 E2  1510 LP21       JSR LP2
70AA- C9 0D     1520            CMP #13       ;CR?
70AC- F0 0D     1530            BEQ IST.CR    ;JA
70AE- C9 13     1540            CMP #19
70B0- F0 C1     1550            BEQ INIT.INPT         ; 'HOME'
70B2- C9 93     1560            CMP #147      ;CLEAR?
70B4- F0 BD     1570            BEQ INIT.INPT         ;JA - IGNORE
70B6- C9 8D     1580            CMP #$8D      ;WARTEN AUF
70B8- D0 C4     1590            BNE LOOP4     ;SHIFT+RETURN
70BA- 60        1600            RTS
                1610            ;
70BB- A4 C6     1620 IST.CR     LDY *SPALTE
70BD- C0 28     1630            CPY #40
70BF- 90 0A     1640            BCC NEUE.ZL   ;KEINE LANGE ZEILE
70C1- A6 D8     1650            LDX *ZEILE
70C3- BD E1 00  1660            LDA LDTB1+1,X
70C6- 09 80     1670            ORA #128
70C8- 9D E1 00  1680            STA LDTB1+1,X
70CB- A9 0D     1690 NEUE.ZL    LDA #13
70CD- 20 D8 E3  1700            JSR PRT.CHR
70D0- A6 D6     1710 RAND.L     LDX *ZEILE
70D2- 20 4A 70  1720            JSR POS.CURS
70D5- 18        1730            CLC
70D6- 90 A9     1740            BCC LOOP3     ;IMMER
                1750            ;
                1760            ;
70D8- 98        1770 INPUT      TYA
70D9- 48        1780            PHA
70DA- 8A        1790            TXA
70DB- 48        1800            PHA
70DC- A5 AC     1810            LDA *CRSW
70DE- D0 13     1820            BNE LOP5      ;NICHT 1. AUFRUF
70E0- A6 D6     1830            LDX *LI.RAND
70E2- CA        1840            DEX
70E3- A5 D8     1850            LDA *ZEILE
70E5- 86 A4     1860            STX *LSTP     ;SAVE FUER INPUT
70E7- 85 A3     1870            STA *LSXP     ;SAVE FUER INPUT
70E9- A4 D7     1880            LDY *RE.RAND
70EB- 88        1890            DEY
70EC- 84 D5     1900            STY *LINE.MX          ;ANZAHL ZEICHEN
70EE- E6 AC     1910            INC *CRSW     ;FLAG 'RETURN DA'
70F0- 4C D0 E2  1920            JMP $E2D0     ;CLP5 IM OS
                1930            ;
70F3- 4C FC E2  1940 LOP5       JMP $E2FC     ;LOP5 IM OS
                1950            ;
                1960            ;
                1970            ;.EN
```

```
0027 HOLVAR =48536
0028 VARSUC =49543
0029 KLAUF  =48882
0030 KLZU   =48879
0031 KOMMA  =48885
0032 STRTES =48521
0033 SETPNT =50717
0034 SETCUR =$E07F
0035 PRTCHR =$E202
0036 DSPP   =$E606
0037 LP2    =$E0A7
0038 CLP5   =$E0F2
0039 LOP508 =$E11E
```

Systemroutinen für einige Versionen von BASIC 4

```
5 REM BEISPIEL FUER 3 ZEILEN AB SPALTE 5
10 POKE218,1:POKE219,3:POKE185,4
15 POKE214,5:POKE215,39
20 PRINT"";:REM CLEAR SCREEN
25 PRINT"1.":PRINT"2.":PRINT"3."
30 SYS(64864)(V1$,V2$,V3$)
35 REM KONTROLLE DER VARIABLEN NACH SHIFT/RETURN
40 PRINT:PRINTV1$:PRINTV2$:PRINTV3$
```

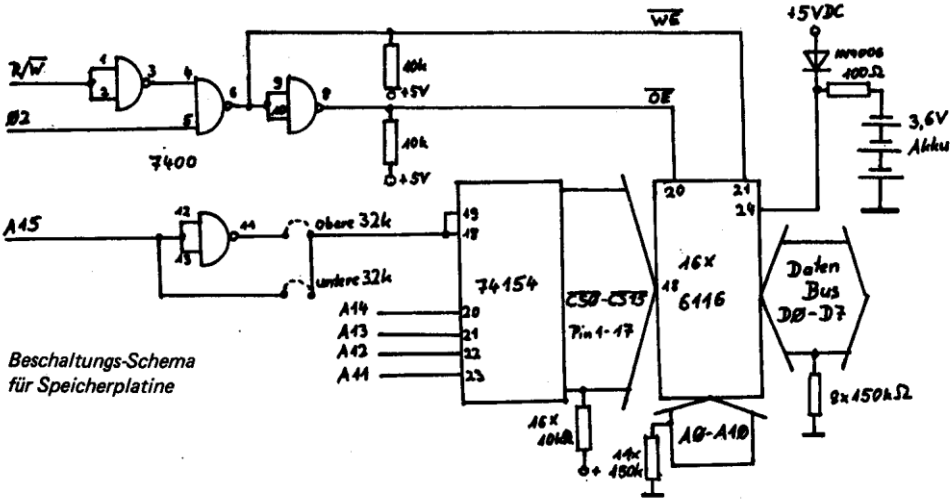
□□

Peter Neuhaus, 5047 Wesseling

32KB CMOS-RAM

Statische RAMs auf Euro-Platine

Durch den raschen Preisverfall der statischen Speicherbausteine ist es jetzt möglich, eine preiswerte und unkomplizierte Speichererweiterung selbst zu bauen. Das Ziel war, den Nachbau durch die Verwendung von gängigen Bauteilen und einer einseitig kaschierten Platine zu erleichtern. Als Speicherbausteine werden CMOS-RAMs vom Typ 6116-LP3 verwendet.

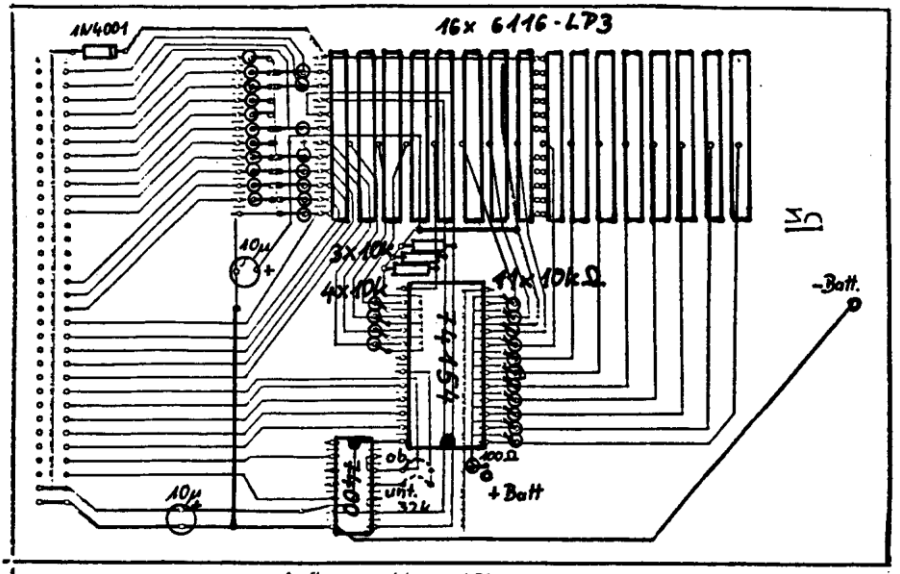


Beschaltungs-Schema für Speicherplatine

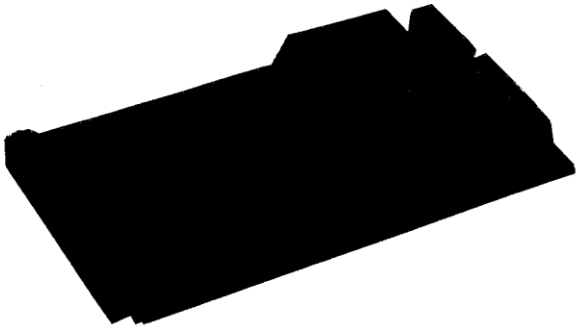
Die einzige Möglichkeit, alle Forderungen zu erfüllen, wurde durch die ungewöhnliche platzsparende Montage der RAMs geschaffen: Die 6116 werden alle aufeinandergelötet (Piggy Pack). Nur der Anschluß $\overline{CS}$ (Pin 18) wird vorher abgewinkelt. Bei der Montage hilft ein Streifen einer Lochrasterplatine. Man steckt die abgewinkelten $\overline{CS}$-Anschlüsse ein, richtet die ICs aus und verlötet die äußeren Pins miteinander. Zum Schluß alle Pins sauber aneinander löten. Vom untersten IC werden die Pins 13-24 abgewinkelt. Damit ist das 'Speicherpaket' fertig.

Bevor es aufgelötet wird, erst alle Brücken, Widerstände, Elkos, Diode und ICs bestücken. Dann erst das 'Speicherpaket' auf die Platine löten. Die Verbindung der Pins 1-12 zur Platine wird durch ein passendes Stück Bandkabel hergestellt.

Sollen Speicherbereiche ausgeblendet werden, kann man für je 2 KByte einfach 1 IC fehlen lassen. Innerhalb des Speicherpakets muß die Verbindung dann durch kurze Brücken hergestellt werden. Vor der Inbetriebnahme sollte man alle Bausteile auf richtigen Einbau kontrollieren und ohne Akku anschließen. Die Stromaufnahme darf bei 32 KByte im Betrieb maximal 600 mA betragen.



Aufbauvorschlag und Photo



Rolf-Hubert Pobloth, 2207 Kiebitzreihe

64 K DRAM am AIM 65 (2)

In Heft 30 dieser Zeitschrift wurde auf Seite 61 bereits das Beschaltungsschema für eine dynamische RAM-Karte vorgestellt, die auch für andere 6502-Systeme aufgebaut werden kann. Hier folgt nun eine genauere Funktionsbeschreibung.

Die wesentlichen Bauteile sind DRAMs vom Typ 4164 und der DRAM-Controller TMS 4500 A von Texas Instruments. Die Speicher haben 256 Refresh-Zyklen pro 4 ms. Beim TMS 4500 A handelt es sich um einen kompletten Controller, der die vollständige Ansteuerung für RAS/CAS und den Refresh übernimmt. Nur die R/W-Leitung und die Datenleitungen werden mit der CPU und den Speichern verbunden. Die Adreßleitungen A0-A7 werden beim TMS an CA0-CA7 gelegt und A8 bis A15 an RA0-RA7. Hier kann natürlich die CS-Anwahl und der Speicherbereich durch entsprechende Codierung individuell angepaßt werden. Mit 0 oder 1 am REN1-Anschluß des TMS kann eine 2. Bank mit 64 Kx8 angesteuert werden. Auf der Speicherseite besteht dafür der Anschluß RAS0 und RAS1. Der Refresh läuft bei beiden Banks gleichzeitig. Der von der CPU kommende Takt 01 wird mit dem ALE und ACR-Eingang vom TMS verbunden.

Für die Refresh-Steuerung ist es wichtig, daß TWST = 0 ist und ein FS-Eingang auf 0 und der andere auf 1 liegt. Mit dieser Kombination arbeitet der TMS mit 3 Refresh-Zyklen, ohne daß es eines externen Refresh-Requesters bedarf. So würde alle 4 ms der Refresh durchgeführt. In diesem Falle ist jedoch der 2 MHz-Takt angelegt, der innerhalb der ersten 500 ns des CPU-Taktes am REF-Request-Eingang mit fallender Flanke einen externen Refresh anmeldet. Da zu diesem Zeitpunkt am ALE/ACR-Eingang noch 1 anliegt, wird nach dem internen Programm des TMS 4500 A ein Refresh durchgeführt, und zwar innerhalb 3 Halbzyklen des 4-MHz-Taktes (3*125 ns=375 ns). An dieser Stelle ist die Bausteinbeschreibung von TI nicht ganz eindeutig. Bei der TI-Beschreibung würde man auf 3*250= 750 ns kommen.

Während der TMS noch den Refresh durchführt, gehen ALE und ACR von 1 auf 0. Nach dem Programm des TMS wird erst der Refresh beendet und dann mit der nächsten steigenden/fallenden Flanke des 4 MHz-Taktes der Memory Access (Speicherzugriff) ausgeführt.

Zu diesem Zeitpunkt stehen bereits die Daten stabil auf dem Datenbus und die R/W-Leistung ist z.B. beim WRITE auf 0. Es kommt zum 'Early-Write-Cycle'. Der TMS steuert nun die RAS-Leitung und zieht nach Ablauf einer genau definierten Zeit die CAS-Leitung auf 0. Rein rechnerisch ist CAS 125 ns vor Ende des CPU-Zyklus auf 0, und es werden nun die Daten in den Speicher geschrieben oder daraus gelesen. In dem Moment, wo die ALE und ACR-Leitung von 0 auf 1 geht (Ende des 01-Taktes und CPU-Taktes), wird auch CAS wieder vom TMS auf 1 gezogen und der Datenbus 'tristated'.

Es ist besonders wichtig, daß alle Takte vom Mutteroszillator durch Teilung mit Flip-Flops (SN 7474) hergeleitet werden. Durch den 2 MHz-Takt wird der TMS 4500 A zwangssynchronisiert. Daß auch beim 'Power On' der 2 MHz-Takt vor dem 01-Takt der CPU am TMS vorhanden ist, wird immer erst der Refresh durchgeführt und dann der Memory Acess. Nur so funktionieren beide Bausteine miteinander. Der Anschluß RDY am TMS ist leider nicht am 6502 zu gebrauchen, er führt zu einem völlig unkontrollierten Verhalten der CPU.

Zeitrechnung: Ein Speicherzugriff der CPU benötigt genau 1000 ns. 1000 ns = 1 CPU-Zyklus mit je 500 ns Halbzyklen 01 und 02.

Ab 000 ns bis 250 ns	Es passiert nichts.
Ab 250 ns bis 625 ns	Es erfolgt der Refresh (3*125 ns=375 ns). In der Zwischenzeit ist 01 von 1 auf 0 gegangen und hat ein MEM-ACESS bei 500 ns am ALE/ACR-Anschluß gemeldet.
Ab 625 ns bis 750 ns	Warten auf nächste 4 MHz-Flanke.
Ab 750 ns bis 875 ns	RAS ausgeben und Wartezeiten.
Ab 875 ns bis 1000 ns	CAS auf 0 ziehen und beim 0- auf 1-Übergang des 01 Taktes wieder CAS auf 1 ziehen und damit den Datenbus wieder in den 'tristated' Zustand bringen. In dieser Zeit von 125 ns werden die Daten in den Speicher geschrieben oder ausgelesen (latchen!).
Neuer CPU-Zyklus	Ablauf wie vor.

Allgemeiner Hinweis: Durch Verwendung von 128 Kx1-Typen kann mit dem TMS ein Speicher bis zu 256 K x 8/16 aufgebaut werden.

□□

Der 68000 VMEbus

Allgemeines

Der Prozessor MC 68000 von Motorola und der Begriff VMEbus werden fast immer zusammen genannt. Was nun ist der VMEbus? Es ist eine freiwillige industrielle Norm, die von Anfang an gewährleisten soll, daß Systemkarten für den 68000 miteinander kompatibel sind. Der VMEbus ist speziell auf die Leistungsfähigkeit und die Signale dieser CPU zugeschnitten und wird von Motorola und den Second Sources (Zweitlieferanten) Philips/Valvo/Signetics, Mostek und Thomson-Efcis weltweit unterstützt und eingehalten. Weiter ist zu erwähnen, daß diese Firmen seit Beginn dieses Jahres wohl mehr als 800 System-Designer aus anderen europäischen Betrieben auf die Spezifikationen dieses Busses geschult haben. Er findet damit Akzeptanz, und es ist zu erwarten, daß damit das Angebot von VMEbus-Produkten rasch zunimmt.

Die Spezifikationen des Busses sind streng und beziehen sich nicht nur auf die elektrischen, sondern ebenso auf die mechanischen Eigenschaften wie Stecker, Signalbelegung, Kartengröße, Lage und Größe der Bohrlöcher bis hin zu den Abmessungen bei den Kartenträgern. Man kann damit sagen, daß eine Firma, die VMEbus-Produkte anbietet, sich zur Einhaltung einer Norm verpflichtet hat und sich daher davon ggfs. abweichende Eigenschaften entgegenhalten lassen muß.

Die bisher vollständigste Beschreibung des VMEbus scheint das etwa 150seitige Buch 'VMEbus Specification Manual Rev. B' zu sein, das von der VMEbus Manufacturers Group (oben genannte Firmen) im August 1982 veröffentlicht wurde und das hier z.B. als Veröffentlichung von Valvo/Signetics vorliegt und die Nummer 9398 315 200 11 trägt.

Warum VMEbus?

Beim 68000 handelt es sich nur um einen Prozessor, sondern um eine ganze CPU-Familie mit 8, 16 und 32 Bit Datenbus und Adreßbussen von 20 bis 32 Bit. Und auch die Betriebsfrequenz überstreicht den Bereich von 4 bis 20 MHz. Daneben gibt es die Möglichkeit, Ko-Prozessoren auf dem Bus zu betreiben sowie Mehrprozessorsysteme aufzubauen sowie DMA (Direct Memory Acess) auszuführen. Und auch die Zahl der Systembausteine wächst. Diese Vielfalt der Konfiguriermöglichkeiten legte es nahe, von Anfang an Kompatibilität durch Festlegungen zu gewährleisten, die die genannten Firmen in der VMEbus Manufacturers Group miteinander entwickelten.

Wichtige mechanische Eigenschaften

Die Kartengröße ist auf Europa- und Doppeleuropaformat festgelegt, und es kommen 64- und 96-polige VG-Buchsen und Federleisten nach DIN 41612 zum Einsatz, dazu Kartenträger im 19''-System. Für alle Komponenten einschließlich Mutterkarten (Backplanes) und Frontplatten gibt es eindeutige Maßvorschriften.

Wichtige elektrische Eigenschaften

Es wurden die Bezeichnungen P1 (oben) und P2 (unten), für die zwei VG-Leisten festgelegt, die den Systembus und den Expansions-/Interfacebus aus einer Doppeleuropakarte herausführen. Ebenso ist die Signalbelegung lt. nachstehenden Aufstellungen vorgeschrieben, und zwar für P1 für die Pinreihen a, b und c sowie für P2 in der mittleren Reihe b. Die Reihen a und c des P2 können vom Designer frei für Interfacezwecke belegt werden.

Die Signalbelegungen mit GND erfolgten auch unter dem Gesichtspunkt der Abschirmung. Der Leser beachte, daß auch die Spannungsversorgung über die VG-Leiste erfolgt und nicht etwa an ein Spannungsterminal angeklemmt wird. Jeder Pin im System muß 1 A Strom leiten können.

Auch für die VME-Mutterkarte (Backplane) gelten Festlegungen hinsichtlich des Busabschlusses mit einem Widerstandsnetzwerk, hinsichtlich der Impedanz und hinsichtlich einer Frequenztauglichkeit von über 20 MHz. Die Buskarte wird damit zu einem aufwendigen Ingenieurprodukt (meistens wohl in Multilayertechnik ausgeführt), das seinen Preis kostet.

Für alle Signale sind elektrische und Ablaufspezifikationen festgelegt und es wird auch auf die für sie geeigneten Treiberbausteine hingewiesen. Für Mehrprozessorsysteme gibt es daneben Signalprotokolle für die Übergabe der Buskontrolle und für die Durchleitung und Abarbeitung von Inter-

J1/P1 Pin Assignments

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BG0IN*	D11
5	D04	BG0OUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BR0*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AM0	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK (1)	A17
22	IACKOUT*	SERDAT (1)	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5V STDBY	+12V
32	+5V	+5V	+5V

NOTE:

(1) SERCLK and SERDAT represent provision for a special serial communication bus protocol still being finalized.

VMEbus: Signalbelegung am Stecker P1

rupts, wenn mehrere Prozessorkarten die gleiche Interruptebene (IRQ1 bis IRQ7) beim Master benutzen, untereinander jedoch verschiedene Priorität genießen. Zu diesem Zwecke wird die Leitung IACKIN/IACKOUT durch die Buskarte und durch die Prozessorkarten hindurchgeführt (daisy chain), und eine Karte mit Vorrang kann eine nachrangige vom IACKIN vorläufig abschneiden.

Zusammenfassung

Für Systeme, die im Laufe der Zeit auf mehr als eine Grundkarte erweitert werden sollen, ist der VMEbus Voraussetzung dafür, daß Kompatiblität zwischen marktgängigen Produkten gewahrt bleibt. Da in diesem Bus viele Erfahrungen und Entwicklungen berücksichtigt werden, stellt er zugleich auch eine Leitlinie für solche Systeme dar, die nicht vermarktet werden sondern als Anwendungscomputer benutzt werden sollen. - Designern ist zu empfehlen, an einem der sicher weiterhin stattfindenden VMEbus-Seminare teilzunehmen, zumal sie bei dieser Gelegenheit auch umfangreiche Unterlagen erhalten.

J2/P2 Pin Assignments

PIN NUMBER	ROW A SIGNAL MNEMONIC	ROW B SIGNAL MNEMONIC	ROW C SIGNAL MNEMONIC
1	User I/O	+5 Volts	User I/O
2	User I/O	GND	User I/O
3	User I/O	RESERVED	User I/O
4	User I/O	A24	User I/O
5	User I/O	A25	User I/O
6	User I/O	A26	User I/O
7	User I/O	A27	User I/O
8	User I/O	A28	User I/O
9	User I/O	A29	User I/O
10	User I/O	A30	User I/O
11	User I/O	A31	User I/O
12	User I/O	GND	User I/O
13	User I/O	+5 Volts	User I/O
14	User I/O	D16	User I/O
15	User I/O	D17	User I/O
16	User I/O	D18	User I/O
17	User I/O	D19	User I/O
18	User I/O	D20	User I/O
19	User I/O	D21	User I/O
20	User I/O	D22	User I/O
21	User I/O	D23	User I/O
22	User I/O	GND	User I/O
23	User I/O	D24	User I/O
24	User I/O	D25	User I/O
25	User I/O	D26	User I/O
26	User I/O	D27	User I/O
27	User I/O	D28	User I/O
28	User I/O	D29	User I/O
29	User I/O	D30	User I/O
30	User I/O	D31	User I/O
31	User I/O	GND	User I/O
32	User I/O	+5 Volts	User I/O

VMEbus: Signalbelegung am Stecker P2

Christian Persson, 2872 Hude 2

SUPERTAPE

Kassettenaufzeichnung mit 600 Byte/Sek.

Ein Cassetteninterface gehört zur Standardausstattung jedes Low Cost-Mikrocomputers. Viele Anwender sind allerdings mit der 'serienmäßig' geboten Leistung nicht besonders glücklich. 'Unzuverlässig', 'zu langsam', 'umständlich in der Bedienung' lautet meistens die Kritik. Grund genug, das für den elrad-COBOLD-Computer entwickelte 'SUPERTAPE'-Verfahren und die dazugehörige Soft- und Hardware ausführlich vorzustellen. Es ist schnell, zuverlässig, komfortabel und leicht für andere 6502-Systeme anzupassen. In vielen Fällen wird man ohne zusätzliche Hardware auskommen.

Voraussetzung ist das Vorhandensein eines Timerbausteines. Bei den meisten 6502-Systemen dürfte diese Bedingung erfüllt sein. Der elrad-COBOLD ist mit einem 6532-RIOT ausgestattet, der u.a. einen 8-Bit-Timer enthält. Die Interface-Hardware ist kaum der Rede wert: Bild 1 zeigt den Vorschlag für eine geeignete Schaltung, die ausgangsseitig ein von Störimpulsen gereinigtes Rechtecksignal liefert und eingangsseitig das vom Kassettenrekorder empfangene Signal in TTL-Pegel umsetzt. Wichtig ist, daß die Versorgungsspannung für das Interface gut gesiebt wird (R1, C1).

Worin bestehen die Vorzüge des SUPERTAPE-Verfahrens?

Punkt 1: Zuverlässigkeit. SUPERTAPE stellt keine besonders hohe Anforderung an den Cassettenrekorder. Die höchste benutzte Frequenz beträgt 4800 Hz. Gleichlaufeigenschaften spielen keine entscheidende Rolle. Es besteht zwar, wie bei jedem Verfahren mit hoher Aufzeichnungsdichte, theoretisch eine relativ große Empfindlichkeit gegenüber Drop-Outs. Diese tritt aber überhaupt nicht in Erscheinung, wenn man Kassetten guter Qualität verwendet und die allgemein bekannten Empfehlungen bezüglich staubfreier Lagerung und gelegentlicher Reinigung des Tonkopfes beachtet. Das SUPERTAPE-Verfahren umfaßt eine Prüfsummenberechnung, durch die Übertragungsfehler mit hoher Wahrscheinlichkeit erkannt werden.

Punkt 2: Schnelligkeit. Der Trend bei Tischcomputern zieltauf eine flexible Speicherausstattung mit viel RAM und wenig ROM. Umfangreiche Systemprogramme werden je nach Bedarf vom Massenspeicher (Magnetband oder Diskette) geladen. Die Ladedauer für ein 8KB-Programm zeigt folgende Tabelle im Vergleich:

KIM-1	ca. 16 min. 30 sec
Kansas City Standard	ca. 11 min
Hypertape (J. Butterfield)	ca. 2 min. 45 sec.
SUPERTAPE	ca. 14 sec.

Punkt 3: Komfort. Es können Datenblöcke beliebiger Größe auf die Cassette übertragen werden. Dies läßt dem Anwender die Freiheit, eine Formatierung vorzusehen. Die Datenblöcke werden durch eine hexadezimale Nummer in 1 Byte identifiziert.Beim Lesen vom Bandlädt der Computer den Datenblock mit der gewünschten ID-Nummer an seinen alten Speicherplatz. Wird eine Sendung mit einer anderen ID-Nummer empfangen, so zeigt der COBOLD-Computer diese auf dem LED-Display an. Das ermöglicht das schnelle Auffinden von Bandstellen ohne Gebrauch des Zählwerkes.

Verfahren

Der Kassettenrekorder als Übertragungskanal weist eine Bandpaß-Charakteristik auf: Signale oberhalb und unterhalb bestimmter Grenzfrequenzen werden zunehmend verfälscht, beziehungsweise gar nicht übertragen. Die Bitwerte '0' und '1' können deshalb nur in Form von Wechselspannungssignalen übertragen werden, die Frequenzen innerhalb der Bandbreite des Übertragungskanales aufweisen. Beim SUPERTAPE-Verfahren wird eine '0' durch eine Schwingungsperiode der höheren Frequenz (4800 Hz), eine '1' durch eine halbe Schwingungsperiode der niedrigeren Frequenz (2400 Hz) repräsentiert. Auf diese Weise entsteht ein praktisch gleichspannungsfreies Signal, das mit einfachen Mitteln (siehe Bild 1) zurückgewonnen werden kann.

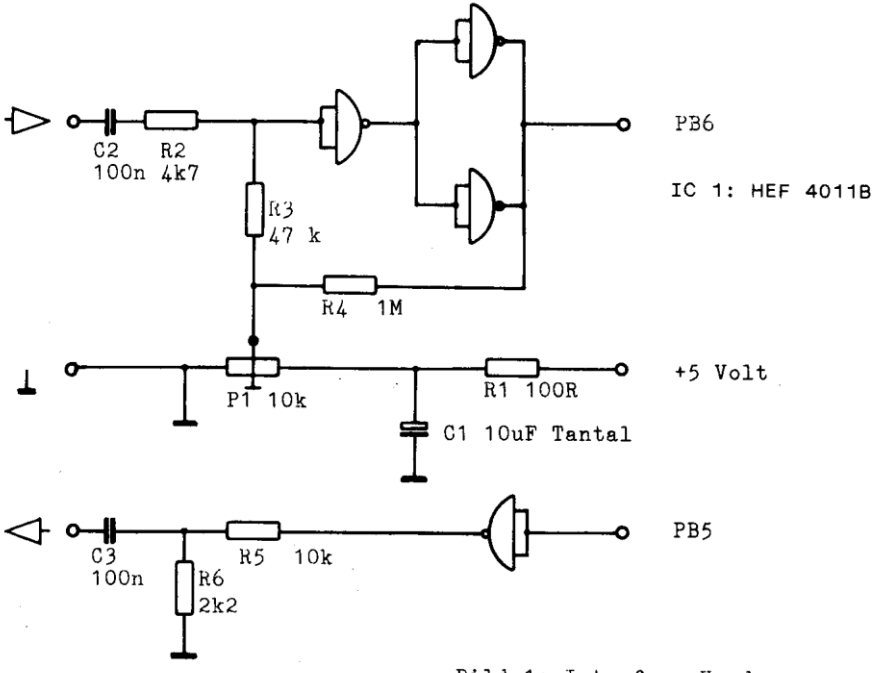


Bild 1: Interface-Hardware

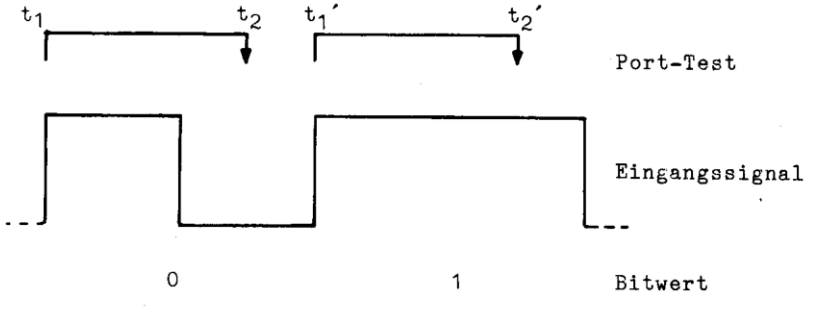


Bild 2: Bit-Repräsentation

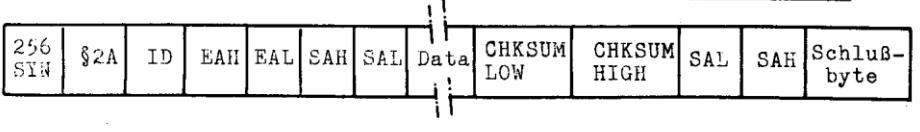


Bild 3: Elemente einer Datensendung

Beim Lesen von der Kassette (Bild 2) ermittelt der Computer jeweils den Beginn der Übertragung eines Bits (t1) und testet die Portleitung PB6, an der das eingehende TTL-Signal anliegt, zum Zeitpunkt t2. Dazwischen liegt eine Dreiviertel-Periodendauer der 4800-Hz-Schwingung. Der zum Zeitpunkt t1 eingenommene logische Zustand wird jeweils mit dem Pegel zum Zeitpunkt t2 verglichen. Sind beide gleich, liest der Computer eine '1', im anderen Fall eine '0'.

Bei näherem Hinsehen wird deutlich, daß dieses Verfahren Bandgeschwindigkeits-Schwankungen von mehr als 20 Prozent toleriert und gegenüber Störimpulsen, die das Signal vom Band überlagern und zu Mehrfach-Triggerung führen, unempfindlich ist: Es kommt lediglich darauf an, daß zum Zeitpunkt t2 ein definierter Pegel vorliegt.

Software

Die Subroutine OUTCH übersetzt jeweils 8 Bits in die zugeordneten Tonfrequenzen und gibt das Signal an PB5 des RIOT aus. Der RIOT-Timer wird im Polling-Mode betrieben, um die jeweilige Halbperioden-Dauer zu bemessen. Beim Aufruf befindet sich das auszugebende Datenbyte im Akku. Bei der Rückkehr aus der Subroutine ist die Übertragung des letzten Bits noch nicht abgeschlossen: Die CPU hat zu diesem Zeitpunkt durch Invertieren von PB5 lediglich die letzte Halbperiode begonnen. Während der Timer läuft, erfüllt die CPU zusätzliche Aufgaben, sie muß rechtzeitig zum Time-Out in das Unterprogramm zurückkehren.

Jede Datensendung enthält neben dem zu speichernden Datenblock eine Anzahl von Begleitzeichen. Sie dienen dazu, den Block zu identifizieren, dessen Speicheradressen zu übermitteln und die Kontrolle auf fehlerfreie Übermittlung zu ermöglichen. Beim COBOLD-Computer setzt sich eine Datensendung aus folgenden Elementen zusammen:

1. Synchronisationssequenz: 256 SYN-Zeichen im ASCII-Code ($16) ermöglichen beim Lesen das 'Einrasten' auf den Takt der Datensendung. So kann die CPU ermitteln, mit welchem der empfangenen Bits ein Byte beginnt.
2. Das Startzeichen ($2A) beendet die SYN-Sequenz.
3. Die ID-Nummer dient zur Unterscheidung der Datenblöcke.
4. Anfangs- und Endadresse ermöglichen es, den Datenblock an seinen alten Platz im Speicher zu übertragen, ohne daß der Anwender die Blockgrenzen angeben müßte.
5. Der Datenblock kann einen beliebigen Umfang haben. Er wird durch die Anfangs- und Endadresse (+1) bezeichnet.
6. Die Prüfsumme dient zur Fehlerkontrolle. Während der Sendung summiert der Computer alle Datenbytes und hält die unteren 16 Bits des Ergebnisses auf dem Band fest. Beim Empfang addiert er wiederum alle Daten und vergleicht das Ergebnis am Ende mit der Prüfsummme vom Band. Tritt eine Differenz auf, so erfolgt eine Fehlermeldung. (im COBOLD wird ein neuer Ladeversuch unternommen, es erscheint ein entsprechendes Bild auf der LED-Anzeige).
7. 2 Byte sind für Sonderzwecke reserviert: Beispielsweise kann hier die Adresse für einen automatischen Programmstart abgelegt werden. Beim COBOLD wird die Anfangsadresse erneut übertragen und beim zweiten Lesen in Zero-Page-Zellen gerettet.
8. Das Schlußbyte dient dazu, die Übertragung des letzten Bits durch das Unterprogramm OUTCH korrekt abzuschließen.

Das Unterprogramm CHKSUM führt die 16-Bit-Addition zur Bildung der Prüfsumme aus. Als Zwischenspeicher dienen die ZP-Zellen TABL/H. Es gelten folgende Parametervereinbarungen: Beim Aufruf von SAVE weist POINTL/H auf die Anfangsadresse des Datenblocks. Die Endadresse (+1) ist in EPL/H enthalten. TOL enthält die ID-Nummer. Die Anfangsadresse wird zu Beginn nach BEGL/H gerettet: POINT dient bei der Ausgabe als Zeiger auf den Speicherplatz, dessen Inhalt gerade übertragen wird. Die anschließende Deklarierung der Portleitungen ist natürlich systemabhängig. Beim COBOLD haben die Daten folgende Bedeutung:

PAD	25	Segmentmuster für 'S'
PADD	7F	PA0 – PA6 sind Ausgänge

PBD	04	Erste LED-Stelle einschalten
PDBB	2F	PB0 – PB3 und PB5 sind Ausgänge

Das X-Rgister wird benutzt, um die Puffer für die Prüfsumme zurückzusetzen und dient dann als Zähler für die SYN-Zeichen, die in der Schleife SYNOUT gesendet werden. In der Schleife unter dem Label PAROUT sendet die CPU die Parameter ID-Nummer, Endadresse und Anfangsadresse. Vor jeder Übertragung eines Bytes vergleicht die CPU unter dem Label ENDCHK die aktuelle Arbeitsadresse POINT mit der angegebenen Endadresse EP. Stimmen beide überein, so verzweigt das Programm zu SUMOUT. Es werden dann die Prüfsumme, die zu Beginn gerettete Anfangsadresse und das Schlußbyte gesendet.

Der Empfang eines einzelnen Datenbits wird in der Subroutine RDBIT ausgeführt. Die CPU startet den RIOT-Timer mit dem erwähnten Offset von einer Dreiviertel-Periodendauer am Ende des Unterprogramms, sie kehrt dann in das aufrufende Programm zurück und kann während der Timerlaufzeit andere Aufgaben erledigen. Beim erstmaligen Aufruf von RDBIT arbeitet das Programm nicht ordnungsgemäß. Die korrekte Funktion ist nur dann gewährleistet, wenn die CPU eine ununterbrochene Folge von Datenbits empfängt. – Die Speicherzelle BYTES dient als Zwischenspeicher für den logischen Zustand der Portleitung PB6. In der Instruktionsfolge hinter dem Label PCHANG lädt die CPU den Inhalt des Registers PBD in den Akku, maskiert die nicht relevanten Bits und bleibt in einer Warteschleife, bis der Pegel sich ändert. Dies ist der in Bild 2 mit t1 bezeichnete Zeitpunkt. Der neue logische Zustand wird wieder in BYTES registriert. Danach startet die CPU den Timer und kehrt aus dem Unterprogramm zurück. Beim nächsten Aufruf von RDBIT wartet die CPU in der Polling-Schleife zu Beginn das Time-Out ab. Danach vergleicht sie den Eingangspegel mit dem in BYTES gespeicherten letzten Stand. Stimmen beide überein, hat das empfangene Bit den Wert '1'. Aufgrund des Vergleichs ist das C-Flag dann gesetzt. Im anderen Fall ist das Bit '0', und das C-Flag wird gelöscht. Unter BITIN registriert die CPU den neuen Stand in BYTES und rotiert den Wert des Carry-Flags als MSB in den Puffer CHAR. Danach wartet sie wiederum die nächste Signalflanke ab. RDCHA dient dazu, ein Byte zu empfangen, wobei Y als Zähler dient. Der Aufruf setzt voraus, daß das Bytemuster der Datensendung erkannt ist.

Im Unterprogramm LOAD empfängt der Computer eine Datensendung vom Band und überträgt die Daten an ihren alten Speicherplatz. Die ID-Nummer muß beim Aufruf in TOL enthalten sein. Beim COBOLD wird zunächst die eingegebene ID-Nummer angezeigt, sobald irgendwelche Signale an PB6 eintreffen. Erkennt er die SYN-Zeichen zu Beginn einer Datensendung, so erscheint das Symbol H (Heading) auf der Anzeige. Während des Lesens wird L angezeigt. Falls die empfangene Datensendung eine andere Nummer aufweist, so erscheint diese auf dem Display. Diese Funktionen sind für einen entsprechend ausgestatteten Computer leicht übertragbar. Zu Beginn werden bei LOAD die Portregister wie folgt deklariert:

PAD	7F	Alle Segmente ausgeschaltet
PADD	7F	PA0–PA6 sind Ausgänge
PBD	09	letzte LED-Stelle aktiviert
PBDD	0F	PB0–PB3 sind Ausgänge.

Damit ist die Display-Steuerung vorbereitet..Um ein Byte in Form zweier Hex-Ziffern darstellen zu können, muß der Computer die beiden Nibble getrennt in Segmentmuster umsetzen. Dazu dient die systemabhängige Tabelle SEGMF. Unter dem Label SHOWNR führt der Computer die Umwandlung aus und rettet die beiden Bitmuster in die Puffer ADL/H. Das Display-Unterprogramm, das bei jedem Aufruf die Codierung ausführt, kann aus Zeitgründen nicht benutzt werden.

Nach RSTCHA werden der Puffer CHAR und TABL/H initialisiert, die die Prüfsumme aufnehmen sollen. Mit SWITCH beginnt der Empfang beliebiger Daten. Der COBOLD bringt die in ADL/H gespeicherten Bitmuster zur Anzeige. Bei jedem Schleifendurchlauf wird PB0 invertiert und damit zwischen den LED-Feldern 5 und 6 umgeschaltet. Nach jedem Aufruf der Routine RDBIT wird geprüft, ob sich im Puffer das SYN-Zeichen hex 16 befindet. Nach dem Erkennen von weiteren 10 SYN-Zeichen wartet die CPU auf das Startzeichen hex 2A. Anschließend wird die ID-Nummer empfangen und angezeigt, wenn sie nicht mit der Vorgabe übereinstimmt. Wenn doch, dann

Ausgabe von 'L'. Danach werden die Parameter empfangen (RDPAR). In der Hauptschleife DATLOP wird vor dem Empfang jeden Datenbytes geprüft, ob das Ende schon erreicht ist. Wenn ja, folgt der Prüfsummenvergleich und bei Nichtübereinstimmung ggfs. ein neuer Ladeversuch.

Bei der Anpassung an andere Computer und Interfacebausteine werden die Portadressen zu ändern sein, z.T. aber auch die Konstanten für den Timer, weil diese auch von der Programmlaufzeit abhängen. Das Interface gem. Abb. 1 wird mit P1 auf größte Eingangsempfindlichkeit abgeglichen. Der Trimmer ist so zu ziehen, daß es auf der Kippe steht, welchen Zustand der Ausgang des Interfaces einnimmt.

```
0200: 85 FE       OUTCH   STA-CHAR          Zeichen in Puffer
0202: A0 08               LDY&08            Bitzähler
0204: A9 C8       CHALOP  LDA&C8            2400-Hz-Zeit
0206: 46 FE               LSR-CHAR
0208: B0 14               BCS DELB          0 oder 1?
020A: A9 60               LDA&60            0: 4800-Hz-Zeit
020C: 2C D5 EF    DELA    BIT RDFLAG
020F: 10 FB               BPL DELA          Time Out abwarten
0211: 8D F4 EF            STA CNTA          Timer starten
0214: AD 82 EF            LDA PBD
0217: 49 20               EOR&20            PB5 invertieren
0219: 8D 82 EF            STA PBD
021C: A9 60               LDA&60
021E: 2C D5 EF    DELB    BIT RDFLAG
0221: 10 FB               BPL DELB
0223: 8D F4 EF            STA CNTA
0226: AD 82 EF            LDA PBD
0229: 49 20               EOR&20
022B: 8D 82 EF            STA PBD
022E: 88                  DEY
022F: D0 D3               BNE CHALOP        Acht Bits ausgegeben?
0231: 60                  RTS

0232: 18          CHKSUM  CLC
0233: 65 EC               ADC-TABL
0235: 85 EC               STA-TABL
0237: A9 00               LDA&00            Akku + TAB: TAB
0239: 65 ED               ADC-TABH
023B: 85 ED               STA-TABH
023D: 60                  RTS

023E: A5 E6       SAVE    LDA-POINTL
0240: 85 E2               STA-BEGL          Startadresse retten
0242: A5 E7               LDA-POINTH
0244: 85 E3               STA-BEGH
0246: A9 25               LDA&25
0248: 8D 80 EF            STA PAD           Ports deklarieren
024B: A9 7F               LDA&7F
024D: 8D 81 EF            STA PADD
0250: A9 04               LDA&04
0252: 8D 82 EF            STA PBD
0255: A9 2F               LDA&2F
0257: 8D 83 EF            STA PBDD
025A: A2 00               LDX&00
025C: 86 EC               STX-TABL          Prüfsumme
025E: 86 ED               STX-TABH          initialisieren
0260: A9 16       SYNOUT  LDA&16
0262: 20 00 02            JSR OUTCH         256 SYN-Zeichen
0265: CA                  DEX
0266: D0 F8               BNE SYNOUT
0268: A9 2A               LDA&2A
026A: 20 00 02            JSR OUTCH         Startzeichen
026D: A2 04               LDX&04
026F: B5 E6       PAROUT  LDA-POINTL,X
0271: 20 00 02            JSR OUTCH         Parameter
0274: CA                  DEX
0275: 10 F8               BPL PAROUT
0277: A5 E6       ENDCHK  LDA-POINTL
0279: C5 E8               CMP-EPL           Endadresse erreicht?
027B: D0 06               BNE DATOUT        Nein: Daten senden
027D: A5 E7               LDA-POINTH
027F: C5 E9               CMP-EPH
0281: F0 13               BEQ SUMOUT
0283: B1 E6       DATOUT  LDA(POINTL),Y
0285: 20 32 02            JSR CHKSUM        Prüfsumme bilden
0288: B1 E6               LDA(POINTL),Y
```

```
028A: 20 00 02          JSR OUTCH       Byte senden
028D: E6 E6             INC-POINTL
028F: D0 E6             BNE ENDCHK
0291: E6 E7             INC-POINTH      Zeiger erhöhen
0293: 4C 77 02          JMP ENDCHK
0296: A5 EC     SUMOUT  LDA-TABL        Prüfsumme senden
0298: 20 00 02          JSR OUTCH
029B: A5 ED             LDA-TABH
029D: 20 00 02          JSR OUTCH
02A0: A5 E2             LDA-BEGL        Anfangsadresse senden
02A2: 20 00 02          JSR OUTCH
02A5: A5 E3             LDA-BEGH
02A7: 20 00 02          JSR OUTCH
02AA: 4C 00 02          JMP OUTCH       Schlußbyte senden

02AD: 2C D5 EF  RDBIT   BIT RDFLAG
02B0: 10 FB             BPL RDBIT       Time Out abwarten
02B2: AD 82 EF          LDA PBD
02B5: 29 40             AND&40          PB6 prüfen
02B7: C5 F9             CMP-BYTES       Zustand geändert?
02B9: F0 01             BEQ BITIN       Nein: C = 1
02BB: 18                CLC             Ja: C = 0
02BC: 85 F9     BITIN   STA-BYTES       PB6 retten
02BE: 66 FE             ROR-CHAR        Bitwert in Puffer
02C0: AD 82 EF  PCHANG  LDA PBD
02C3: 29 40             AND&40          Warte auf
02C5: C5 F9             CMP-BYTES       nächste Flanke
02C7: F0 F7             BEQ PCHANG
02C9: 85 F9             STA-BYTES
02CB: A9 80             LDA&80          3/4 x 2400-Hz-Zeit
02CD: 8D F4 EF          STA CNTA        Timer starten
02D0: 60                RTS

02D1: A0 08     RDCHA   LDY&08
02D3: 20 AD 02  RDCHLP  JSR RDBIT       Acht Bits empfangen
02D6: 88                DEY
02D7: D0 FA             BNE RDCHLP
02D9: A5 FE             LDA-CHAR        Zeichen: Akku
02DB: 60                RTS

02DC: A9 7F     LOAD    LDA&7F
02DE: 8D 80 EF          STA PAD         Ports deklarieren
02E1: A9 7F             LDA&7F
02E3: 8D 81 EF          STA PADD
02E6: A9 09             LDA&09
02E8: 8D 82 EF          STA PBD
02EB: A9 0F             LDA&0F
02ED: 8D 83 EF          STA PBDD
02F0: A5 EA             LDA-TOL
02F2: 48        SHOWNR  PHA
02F3: 4A                LSR-A           Oberes Nibble
02F4: 4A                LSR-A           wird Index
02F5: 4A                LSR-A
02F6: 4A                LSR-A
02F7: A8                TAY
02F8: B9 4F FF          LDA SEGMF,Y     Segmentmuster
02FB: 85 FC             STA-ADL         retten
02FD: 68                PLA
02FE: 29 0F             AND&0F          Unteres Nibble
0300: A8                TAY             wird Index
0301: B9 4F FF          LDA SEGMF,Y     Segmentmuster
0304: 85 FD             STA-ADH         retten
0306: A9 FF     RSTCHA  LDA&FF          Puffer initialisieren
0308: 85 FE             STA-CHAR
030A: A9 00             LDA&00          Prüfsumme
030C: 85 EC             STA-TABL        initialisieren
030E: 85 ED             STA-TABH
0310: AD 82 EF  SWITCH  LDA PBD
0313: 49 01             EOR&01          Umschalten zwischen
0315: 8D 82 EF          STA PBD         LED-Digits 5 und 6
0318: 29 01             AND&01
031A: A8                TAY             PB0 wird Index
031B: B9 FC 00          LDA ADL,Y       Segmentmuster
031E: 8D 80 EF          STA PAD         ausgeben
0321: 20 AD 02          JSR RDBIT       Ein Bit empfangen
0324: A5 FE             LDA-CHAR        SYN-Zeichen
0326: C9 16             CMP&16          im Puffer?
0328: D0 E6             BNE SWITCH
```

```
032A: A2 0A             LDX&0A          Mindestens 10 SYN
032C: 20 D1 02  SYNCA   JSR RDCHA       empfangen
032F: C9 16             CMP&16
0331: D0 D3             BNE RSTCHA      Nicht SYN: Von vorn
0333: CA                DEX
0334: 10 F6             BPL SYNCA
0336: A9 48             LDA&48          Zeige "H"
0338: 8D 80 EF          STA PAD
033B: 20 D1 02  SYNCB   JSR RDCHA
033E: C9 16             CMP&16          Weitere SYN-Zeichen?
0340: F0 F9             BEQ SYNCB       O.k., weiter
0342: C9 2A             CMP&2A          Startzeichen?
0344: D0 C0             BNE RSTCHA      Nein: Von vorn
0346: 20 D1 02          JSR RDCHA
0349: C5 EA             CMP-TOL         Gesuchter Block?
034B: D0 A5             BNE SHOWNR      Nein: Zeige Nummer
034D: A9 71             LDA&71
034F: 8D 80 EF          STA PAD         Ja: Zeige "L"
0352: A2 03             LDX&03
0354: 20 D1 02  RDPAR   JSR RDCHA       Parameter empfangen
0357: 95 E6             STA-POINTL,X
0359: CA                DEX
035A: 10 F8             BPL RDPAR
035C: A5 E6     DATLOP  LDA-POINTL
035E: C5 E8             CMP-EPL         Endadresse erreicht?
0360: D0 06             BNE RDDAT       Nein: Daten empfangen
0362: A5 E7             LDA-POINTH
0364: C5 E9             CMP-EPH
0366: F0 11             BEQ RDSUM
0368: 20 D1 02  RDDAT   JSR RDCHA       Daten in den
036B: 91 E6             STA(POINTL),Y   Speicher übertragen
036D: 20 32 02          JSR CHKSUM      Prüfsumme bilden
0370: E6 E6             INC-POINTL
0372: D0 E8             BNE DATLOP      Zeiger erhöhen
0374: E6 E7             INC-POINTH
0376: 4C 5C 03          JMP DATLOP
0379: 20 D1 02  RDSUM   JSR RDCHA       Prüfsumme vom Band
037C: C5 EC             CMP-TABL        mit neuer Summe
037E: D0 12             BNE LOADVC      vergleichen
0380: 20 D1 02          JSR RDCHA
0383: C5 ED             CMP-TABH
0385: D0 0B             BNE LOADVC      Ungleich: Von vorn
0387: 20 D1 02          JSR RDCHA
038A: 85 E2             STA-BEGL        Anfangsadresse retten
038C: 20 D1 02          JSR RDCHA
038F: 85 E3             STA-BEGH
0391: 60                RTS
0392: 4C DC 02  LOADVC  JMP LOAD        Von vorn                    □□
```

Bücher

Walz, Georg: Grundlagen und Anwendung des IEC-Bus. Verlag Markt & Technik, Haar 1982, ISBN 3-922120-22-9, 169 S., DM 44,--. Das Buch hat folgende Hauptkapitel: Das Schnittstellensystem, Die Schnittstellen-Funktionen, Nachrichten, Codes und Datenformate, Systemkonfiguration, Realisierung der Schnittstellentechnik, Tischcomputer, Geräte mit IEC-Bus-Anschluß. Mit seiner Darstellung der Hardware, der Signalbelegung, der Schnittstellenbausteine und -schaltungen sowie mit den Ablaufdiagrammen, logisch und impulsmäßig, gibt das Werk erschöpfende Auskünfte für alle Aspekte des Interfacebusses. Es kann damit Entwicklern und Anwendern des IEEE 488-Bus sehr empfohlen werden.

Schumny, H. (Hrsg.): Iterationen, Näherungsverfahren, Sortiermethoden. BASIC-Programme für CBM 3032, HP-9830, Olivetti 6060. Braunschweig, Vieweg 1982. 80 S. ISBN 3-528-04207-9, DM 21,80. Das Buch enthält komplette Programme, die wegen der beigefügten Ablaufdiagramme auch auf andere Rechner übertragbar sind. Folgende Lösungen mögen für die Leser interessant sein: Allgemeines Iterationsverfahren y=f(x), Quadratische Gleichung, Polynomerzeuger, Berechnung der Nullstellen eines Polynoms, Polynomdivision, Magisches Quadrat, dann Programme speziell für CBM: Sortieren durch Auswahl, Austausch, Einfügen, Heap- und Quicksort, Sortieren durch Mischen, Binäres Suchen. Programme zur Pi-Bestimmung.

Dripke, Andreas: 6502 Assembler-Kurs für Beginner, Wiesbaden 1983, ISBN 3-88986-000-1, etwa 150 Seiten, als Ringbuch im Schreibmaschinensatz. A. Dripke ist der Autor des EXBASIC Level II für Commodore und damit ein intimer Kenner der Assemblersprache. Er führt den Leser schrittweise und geordnet nach Befehlsgruppen an die Programmierung unter Assembler heran. Am Schluß enthält das Buch Referenzseiten für CBM, Apple II, Atari 400 und 800, VC20 und VC64 und für die CPU 6510. Es wird seinem begrenzten Anspruch, ein Buch für Anfänger zu sein, sehr gut gerecht und hat für viele den Vorteil, in Deutsch geschrieben zu sein. Es wird vertrieben durch den Verlag Interface Age.

Kaier, Ekkehard: BASIC-Programmierbuch. Braunschweig: Vieweg 1983, 185 S. ISBN 3-528 0422 -2, DM 32,--. Ein weiteres BASIC-Buch, aber ein anderes mit 46 Programmbeispielen: Es wird die Struktur der Programmierung herausgestellt, und zwar für lineare Abläufe, verzweigende Abläufe (Auswahlstrukturen) und für Wiederholungsstrukturen (Abläufe mit Schleifen). Jedes der Programme ist nach dem gleichen Schema beschrieben: a) Problemstellung, b) Problemanalyse, c) Programmablaufplan und Struktogramm, d) Codierung in BASIC, e) Programmausführung mit Computer- und Schreibtischtest und f) Fragen. BASIC verleitet bekanntlich zur Programmierung im Freistil. Das Buch ist keine Einführung in den Sprachschatz, wohl aber ein Anleitung, ihn sinnvoll zu gebrauchen. Es ist jedem zu empfehlen, der sich um klare Problemlösungen bemüht.

Floegel, E.: FORTH on the ATARI, Learning by Using. Hofacker-Verlag, Holzkirchen 1982, 118 S DM 29,80. Der deutsche Autor schrieb ein Buch in English für den internationalen Markt. Sein Ziel ist, dem Neuling in dieser Sprache ihren Gebrauch auf dem ATARI zu erklären. Die Beispiele sind kurz und benutzen Tonerzeugung und Grafik um Anwendungen der Sprache zu zeigen. Enthalten ist ferner das Programm einer Mailing List und für die serielle Ansteuerung eines Druckers. Mit seinen zahlreichen Beispielen regt es sicher zum Gebrauch der Sprache und des Computers an.

Langfelder, C.: BASIC ohne Probleme, Einführung in die BASIC-Programmierung auf CBM-Rechnern (CBM 8032). Verlag Markt & Technik, Haar 1983, 226 S. ISBN 3-922120-25-3, DM 29,80. Es handelt sich um 'ein Anfängerhandbuch zum Erlernen von BASIC' und zur Hantierung des Rechnermodells. Es hat etwa die Aufmachung eines Anwenderhandbuches für den Rechner und man sollte vermuten, daß es mehr die Hantierung zeigt und die Wirkung von Befehlen, als das es aufzeigt, wie man Probleme in der Sprache BASIC löst.

Aus der Branche

Das Deutsche Video Institut e.V. in Berlin veranstaltet ab Juni 1983 an verschiedenen Orten eintägige Einführungsseminare für 'Heimcomputer für die private Anwendung'. Damit sind Geräte in der immer populärer werdenden Preisklasse von etwa DM 400-1500 gemeint, auch die sog. Hand Held Computer. Als Teilnehmer sind die Betriebe des Rundfunk- und Fersehfachhandels eingeladen. Einem Arbeitsgespräch, das Ende Mai in Berlin auf Einladung des DVI hin zwischen Vertretern der Industrie, des Fachhandels und weiteren Fachleuten geführt wurde, war zu entnehmen, daß sich der Rundfunk- und Fernsehfachhandel als den geeigneten Vertriebsweg für solche Heimcomputer für die private Anwendung sieht.

Ein Inhaltsverzeichnis für die Hefte 1 bis 29 des 65xx MICRO MAG als Tonbandfile wird von Herrn Rainer Borchers, Habichtweg 19, 5600 Wuppertal 1 angeboten. Das File wird nach Wunsch als 32K-Version zusammenhängend oder als 2*16K-Version getrennt auf Tonbandcassette geliefert, und zwar für den AIM 65. Es erfaßt vor allem solche Beiträge, die für den AIM 65 relevant sind und natürlich auch solche von allgemeinem Interesse. Man kann es ausdrucken oder es mit dem Find-Befehl des Editors als kleine Datenbank benutzen. Es wird gegen Vorüberweisung von DM 15 (für Cassette, Verpackung, Porto) auf das Konto 2597 39-504 (R. Borchers) beim Postscheckamt Köln geliefert.

Die GWK Gesellschaft für technische Elektronik in Herzogenrath hat ihre zahlreichen Systemkarten für 8-Bit-Computer (6502 und 6809) durch Adapter für den VME-Bus der 68000-CPU anschließbar und nutzbar gemacht, so daß auch von dieser Seite her von Anfang an viel Peripherie und Interfacing zur Verfügung steht. Der Adapter wurde bereits auf der diesjährigen Hannover-Messe ausgestellt.

Zu dem in Heft 30 ab Seite 31 abgedruckten Artikel MOVE und RELOCATE ist ergänzend darauf hinzuweisen, daß diese Zeitschrift in Heft 17 eine ähnliche Dienstleistung enthielt, und zwar aus der Feder von Dr. W.-D. Schnell unter dem Titel: LMR - Load, Move, Relocate.

Assembler unter FORTH

CROSS–09

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

CROSS–05

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

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

Mathe–ROM für 6502

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

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

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

Kleinanzeigen

AIM 65 REMON V3.4: Geänderter Monitor in 2*4k-EPROMs. Tastenrepeat, Groß-/Kleinschreibung, neue Monitorbefehle, deutscher Zeichensatz, Erweiterung der Monitor- und Editorbefehle über Vektoren, serielles Druckerinterface u.v.a. für DM 98,--. Bestellungen bei Dipl.-Ing. Rüdiger Wollenberg, Stockumer Str. 234, 4600 Dortmund 50, Tel.: 0231 - 75 34 77.

AIM 65 4K RAM 700,-; Netzteil 5V/6A, 24 V/1A in Metallgehäuse 150,-; Gehäuse von MSB 80,-; Videointerface 120,-; SK-Busverstärkung 120,-; 12K stat. RAM auf E-Karte 290,-; 8K BASIC + 2K Erweiterung 180,-; Assembler 100,-. Th. Weinstein, Tel.: 0721 - 68 53 66.

AIM 65 mit DRAM+ (16K, 2 VIA, Eprommer) Netzteil, Gehäuse, alle Sprachen! DM 1850 VB. EUROCOM 2 Software. R. Hammerschmidt, Heinr.-Heine-Str. 4, 4000 Erkrath, T. 02104-42 837.

Kleinanzeigen kosten DM 10,- für die ersten 2 Zeilen. Betrag bitte möglichst gleich bei Auftragserteilung überweisen/einsenden.

Editorial

In diesem Heft konnten leider nicht alle vorliegenden Arbeiten vollständig abgedruckt werden. Der Herausgeber bittet die Leser und Autoren um Verständnis und beschränkt sich auch selbst aus Platzgründen. Wegen der bevorstehenden Sommerpause kommt das nächste Heft erst in der 2. Hälfte des Monats August zum Versand.

Der für die CMOS-CPU R65C02 von Rockwell in Arbeit befindliche 2-Pass-Assembler konnte aus Zeitgründen noch nicht ganz abgeschlossen werden. Wenn es irgendwo deswegen brennen sollte, darf man hier durchaus Dampf machen.

Das 65xx MICRO MAG geht mit diesem Heft in den 6. Jahrgang. Der Herausgeber möchte sich aus diesem Anlaß bei den Lesern für die langjährige beständige Verbindung bedanken und auch den vielen freien Autoren Dank sagen, die in einer sehr vielfältigen Art Lösungswege aufzeigten und damit den sinnvollen Gebrauch der kleinen Rechner beförderten. Trotz vieler anderer großer Zeitschriften, die in diesen Jahren hinzutraten, ist es gelungen, wie es immer wieder bestätigt wird, in der Qualität des Inhalts eine führende Stelle zu behaupten. Die Leser sind damit aufgerufen, auch weiterhin aus ihren Arbeitsgebieten zu berichten. Bei der Vielzahl der hinzutretenden neuen Rechnermodelle wird es auch darauf ankommen, Produktbesprechungen für diese zu erhalten und zu veröffentlichen, um die Marktübersicht zu behalten.

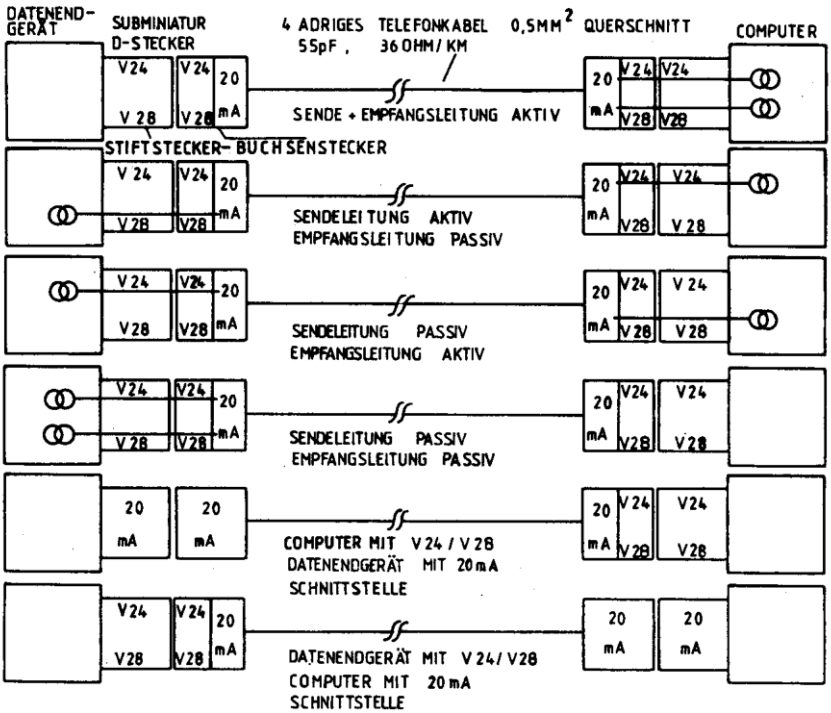
Es sind einige Bitten auszusprechen: Vor dem Ausdruck einer Programm-Liste sollten die Autoren ein kräftiges schwarzes Fabband auf den Drucker auflegen, denn das häufig eingesandte Blaugrau verursacht erhebliche Schwierigkeiten für die Wiedergabe. Begleittext braucht nur lesbar zu sein, weil er sowieso über diese Setzmaschine eingegeben wird.

Jede Ausnahme vom normalen Geschäftsgang verursacht vermeidbaren Verwaltungsaufwand, wenn z.B. ein Überweisungsabschnitt weder den Namen des Absenders noch eine Rechnungsnummer trägt. In diesen Fällen sind Nachforschungen anzustellen. Oder wenn eine Überweisung erst nach dem Erscheinen des Anschlußheftes eingeht. Dann ist in manuellem Verfahren ein Nachversand vorzunehmen, während der rechnergesteuerte Hauptversand wesentlich arbeitssparender ist. Vermeiden Sie bitte Pannen dieser Art!

Die Stecker

von Stecker haben es in sich!

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



Der Stecker

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

5 km

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

die Stecker

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

STECKER
INGENIEURBÜRO

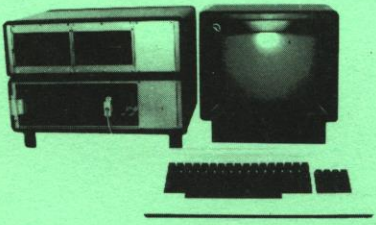
Postfach 60 07 66 5000 Köln 60 Neue Telefon-Nummer!! Tel.: (0221) 7 12 40 18

GWK

GESELLSCHAFT FÜR TECHNISCHE ELEKTRONIK mbH.
HARDWARE SOFTWARE SYSTEMENTWICKLUNG

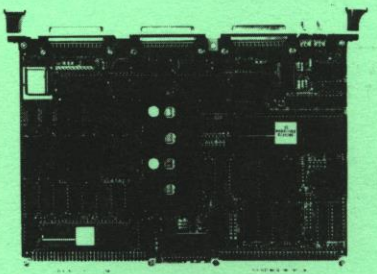
MICROCOMPUTER

FÜR DEN EINSATZ IN TECHNIK U. WISSENSCHAFT



GWK EURO BOARD COMPUTER SYSTEM

Modulares Europakarten System für die 8 Bit CPU s 6809 und 6502



SYS 68K VME

Modulares VME-BUS System mit 16 Bit CPU 68000

FORTUNE 32:16

Hochleistungs System CPU 68000 und UNIX Betriebssystem Multiuser Multitasking

D 5120 Herzogenrath Asternstr. 2
Tel.: 02406/6035 Telex: 832109 gwk d

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

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

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

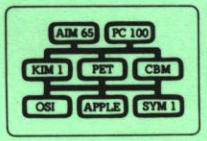
Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- (Inlandsendpreis). Ausland/foreign via surface mail DM 59,--, USA air 26 Dollar. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu zwei Wochen vor deren Ablauf.

Nachliefermöglichkeiten: Hefte 1-6 und 7-13 sind als Buch nachlieferbar (s. Anzeige unten). Hefte 14-30 können alle als Einzelhefte zu DM 7,80/St. + DM 2,50 je Sendung geliefert werden. Einzelpreis bei 10 und mehr Heften je Sendung DM 6,--.

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

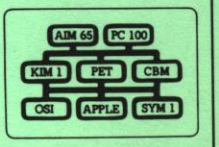
Leser-Service des Herausgebers

Das Buch 1-6 des 65.. MICRO MAG



230 Seiten, DM 26,--

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Thermokopf (Printerplatte für AIM 65 und PC 100
Artikel läuft aus. Lieferung nur solange Vorrat. DM 28,--

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

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

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

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

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