65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7,80 Nr. 12 April 1980

Die Entwicklung von Software

ist mit Zeit und Kosten behaftet, die den Anschaffungspreis eines Systemes schnell übersteigen können. Bei der Systemauswahl sollte man daher den zur Verfügung stehenden 'software support' genauer betrachten. Die Dokumentation wird von vielen Herstellern sicher noch zu lieblos behandelt, man darf aber auch nicht erwarten, daß sie alle Interessen der Betreiber abdeckt. Der Anwender sollte sich daher an den greifbaren Lösungen und Entwicklungsinstrumenten orientieren. Für sein 65er System findet er, wie diese Ausgabe mit den Beiträgen einer engagierten Leserschaft wieder dokumentieren möchte, ein reiches Angebot an Lösungsvorschlägen und Belehrung. Möge der Gedankenaustausch sich weiterhin so fruchtbar entwickeln!

Inhaltsverzeichnis

I. DOHMANN ELEKTRONISCHE BAUGRUPPEN
Im Wiehagen 15 - 4830 Gütersloh 12 Tel.: 05241 - 67 480

CPU-Karte NICO 65

CPU: 6502, 1 oder 2 MHz, 2 kB RAM, max. 8 kB EPROM beliebigen Typs (24-polig), 2 VIAs = 40 Pin I/O über Pfostenleiste nach vorne ausgeführt, alle Busse gepuffert, VG 64 Steckerleiste, Power-On-Reset, alle Interruptmöglichkeiten, Europakartenformat.

Preis:	DM 450,-
in Grundausstattung = 1 kB RAM, 1 VIA	DM 390,-

CPU-Karte NICO 69

CPU: 6809, Beschreibung wie oben. Die CPU hat eine interne 16-Bit-Struktur und arbeitet nach außen hin auf 8 Bit Datenbusbreite. Die Busse sind zum 6502 hin kompatibel.

Preis:	DM 510,-
in Grundausstattung = 1 kB RAM, 1 VIA	DM 450,-

Combo-Karte

8 kB RAM statisch, 2 VIAs, 40 Pin I/O über Pfostenleisten nach vorne ausgeführt, RAM und VIAs getrennt selektierbar, busgepuffert.

Preis:	DM 560,-
nur mit RAMs bestückt	DM 470,-
nur mit VIAs bestückt	DM 140,-

Video-Karte

Bildschirmformat beliebig (max. 80 Zeichen x 24 Zeilen), bis zu 16 Rasterzeilen pro Zeichenzeile, daher echte Unterlängen, Potenzschreibweise und Indices möglich. Zeichengenerator für 128 Zeichen frei programmierbar, ausbaubar auf 256 Zeichen on board, Betriebsprogramm im EPROM on board, 2 k Bildwiederholspeicher, Grafik bzw. Semigrafik möglich. Durch Erweiterung halbe Helligkeit, Blinken, 4 k Bildwiederholspeicher, Tongenerator, Strichgrafik für Formulare zwischen den Zeichen bzw. Zeilen.

Preis:	DM 660,-

Buskarte

8 Steckplätze für alle Normen mit BG 64 Leisten	DM 130,-

Matrixdrucker

Friktionsantrieb für normales Rollenpapier, 80 Zeichen pro Zeile, Zeilenabstand 4,23 mm, 100 Zeichen pro Sek., Zeichenvorrat beliebig; normal: 64 Standard ASCII-Zeichen. Betriebssoftware incl.

Preis:	DM 745,-
Stahlblechgehäuse dazu:	DM 94,-

Interfacekarte dafür

Treiber, Schutzschaltung für Druckkopf, Motorsteuerung und Stromversorgung, ohne Trafo (2x24 V, 1 A AC).

Preis:	DM 73,-

Alle Preise zuzüglich 13 % MWSt.

Wolfgang Radeloff, Elmshorner Str. 13, 2080 Pinneberg

BASIC-Keywords to Shifted Keys

Diese utility erzeugt mit einem einzigen Tastendruck komplette Befehle (Schlüsselworte) in BASIC. Sie beschleunigt damit den Eingabemodus beim Programmieren und vermindert grammatikalische Fehler. Das Programm wird in zwei Versionen vorgelegt, als BASIC-Lader für ein Maschinenprogramm, in dessen Ausführung es nahtlos übergeht, und als reines Maschinenprogramm in Assembler-Schreibweise. Es ist für den PET 2001 mit 8kB geschrieben. Es kann durch Änderung der Systemadressen auf CBM umgestellt werden. Es wurde auch bewußt darauf verzichtet, einen Vergleich mit den im Interpreter-ROM niedergelegten Schlüsselworten durchzuführen. Das Programm kann so leicht auch auf andere Systeme umgeschrieben werden. Es ist natürlich auch möglich, die BASIC-Schlüsselworte durch andere Worte zu ersetzen, z.B. durch mnemonische Codes im 65xx-Assembler. Die Indextabelle ist dann zu modifizieren.

Programmbeschreibung: Außer den Funktionstasten gibt jeder mit SHIFT kombinierte Tastendruck ein BASIC-Schlüsselwort ab aktueller Cursor-Adresse auf den Bildschirm aus. Dabei ist folgende Grundfunktion implementiert: Die Routine ON/OFF ändert den IRQ-Vektor in $ 0219 und 021A und schaltet das Anwenderprogramm der PET-Interruptroutine nach. Ein erneutes Aufrufen der Routine stellt den ursprünglichen Zustand wieder her.

Während des Einschaltzustandes arbeitet die Cassettenladeroutine nicht. Mit SYS(6805) wird die EIN/AUS-Funktion von BASIC aufgerufen. Der IRQ-Zeiger ist auf den Programmteil 'TKEY' gerichtet. Mit einem JSR-Aufruf wird der Stack mit 4 x 00 geladen und dann die PET-IRQ-Routine abgearbeitet.

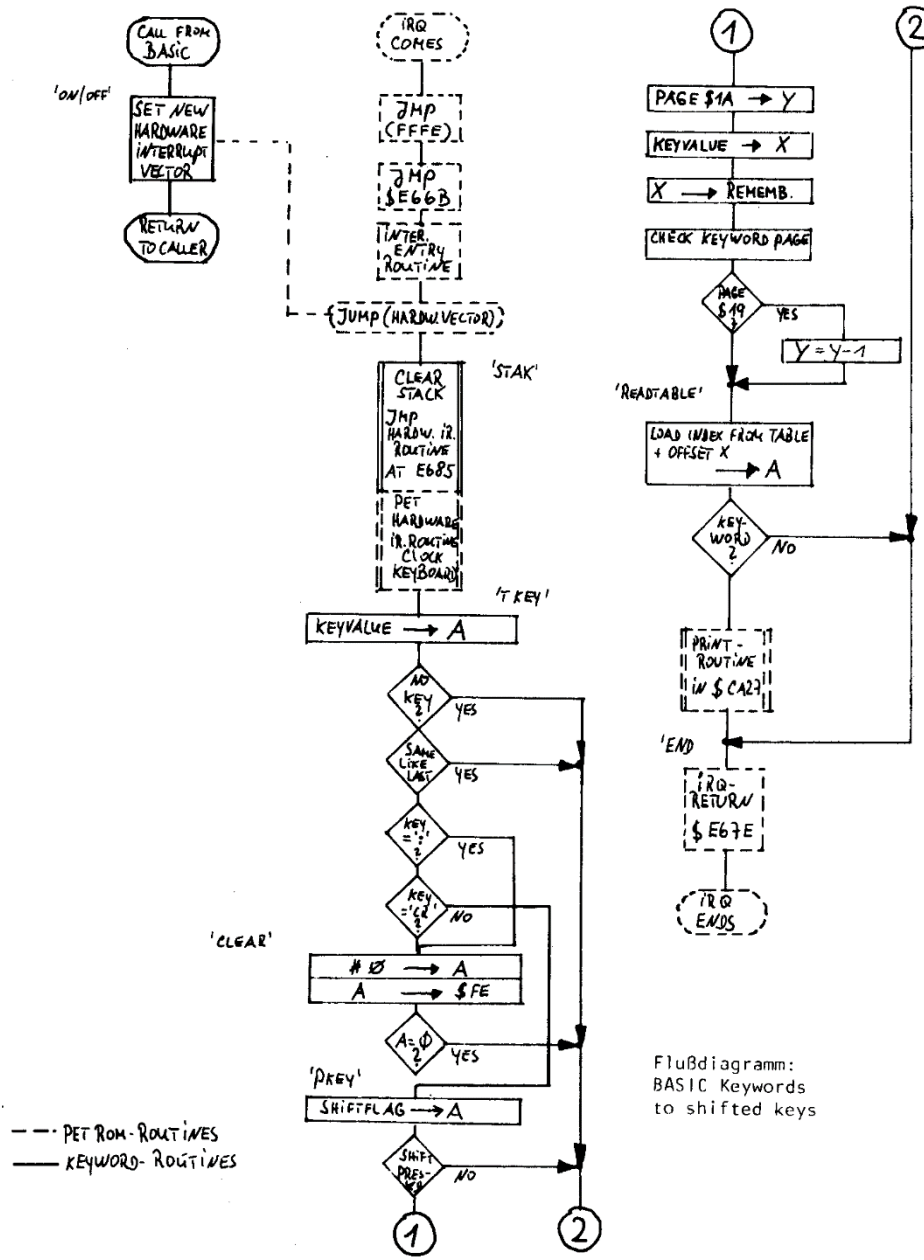
Die Tasten werden durch Lesen des Registers $0203 erkannt. Jeder Tastendruck wird dort mit Codes von % 1-80 gespeichert. Ist keine Taste gedrückt, so ist 255 hinterlegt. In Zeile 635 wird dies getestet, das Programm dann abgebrochen und der Interrupt beendet. - Um zu verhindern, daß ein Wort mehrmals erscheint, wird der zugehörige Tastencode in einem Register zwischengespeichert. Zeile 645 testet auf Wiederholung und bricht ab, wenn eine Wiederholung vorliegt. Die Tasten ':' und 'CR' löschen den Zwischenspeicher und erlauben eine nochmalige Ausgabe des Wortes.

Der Programmteil 'PKEY' testet, ob die Shift-Taste betätigt ist. Wenn nicht, wird das Programm beendet. Die Pageadresse $ 1A der BASIC-Worttabelle wird in das Y-Register geladen. Der Tastencode stellt gleichzeitig die Stelle in der Indextabelle dar, in der das Low-Byte der BASIC-Wortadresse notiert ist. Der Tastencode wird mit der Anzahl der in Page $ 19 notierten Worte verglichen. Bei Überschreitung wird das Y-Register auf $ 1A belassen, sonst auf $ 19 dekrementiert.

In 'READTABL' wird mit dem in das X-Register geladenen Tastencode aus der Index-Tabelle das Low-Byte der Wortadresse geladen. Die vollständige Wortadresse ist jetzt mit dem High-Byte im Y-Register und dem Low-Byte im Akku gefunden, und die PET-Subroutine zur Speicherausgabe auf den Bildschirm wird aufgerufen. Die IR-Routine wird dann mit Sprung zum PET IRQ-Return beendet.

Literatur:

KILOBAUD 3/1979, PET Userport Cookbook, Gregory Yob,
CHIP 7/1979, PET 2001: Simulierung der Repeat-Taste,
65xx MICRO MAG 10/79, Restore Line Numer, U. Kornnagel
65xx MICRO MAG 10/79, Ein Leitfaden für die Programmierung, R. Löhr

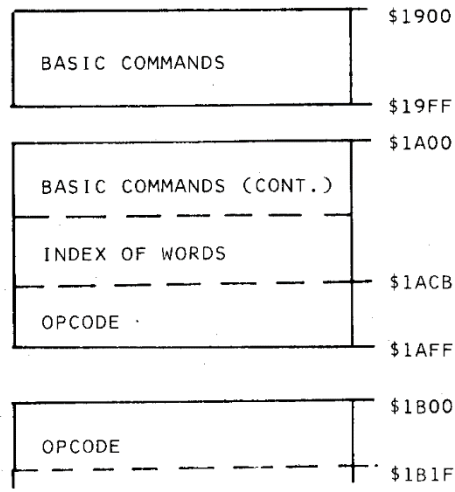


Flußdiagramm:
BASIC Keywords
to shifted keys

```
10 REM BASIC KEYWORDS TO SHIFTED KEYS
11 REM ------------------------------
12 REM COPYRIGHT BY WOLFGANG RADELOFF
13 REM 208 PINNEBERG, ELMSHORNERSTR 13
14 REM TEL: C/O SANYO VIDEO VERTRIEB
15 REM 040/24 62 66    3.1.1980
100 REM ********************************
101 REM *                              *
102 REM *  TABLE OF BASIC-KEYWORDS     *
103 REM *                              *
104 REM ********************************
105 POKE134,00:POKE135,25:DIMC$(15)
109 AD=6400:POKEAD,0:AD=AD+1
110 READ KW$:IF KW$="***"THEN164
115 SP=LEN(KW$)
116 FOR I=1TOSP:AS$=MID$(KW$,I,1)
117 AS=ASC(AS$)
118 POKE AD,AS:AD=AD+1
119 NEXTI
120 POKEAD,0:AD=AD+1
125 IFAD=6652THENPOKE6656,0:AD=6657
130 GOTO110
135 REM
140 REM ********************************
145 REM *                              *
150 REM *       LOAD PROGRAMM          *
155 REM *                              *
160 REM ********************************
162 REM
164 AD=6784
166 FORI=0TO15:READC$(I):NEXT
168 READBY$:IFBY$="+++"THEN1500
169 HB$=LEFT$(BY$,1):LB$=RIGHT$(BY$,1)
170 FORK=0TO15
171 IFHB$=C$(K)THENDE=K*16:K=15
172 NEXTK
173 FORL=0TO15
174 IFLB$=C$(L)THENDE=DE+L:L=15
175 NEXTL
176 POKEAD,DE:AD=AD+1
178 IFAD=6896THENAD=6912
179 GOTO168
190 REM
191 REM ********************************
192 REM *                              *
193 REM *    TABLE OF KEYWORDS         *
194 REM *                              *
195 REM ********************************
196 REM
200 DATA "■COS(","■SGN(","■SQR(","■INT(","■SIN(","■LOG("
202 DATA "■EXP(","■ABS(","■RND(","■ATN(","■OPEN","■PRINT"
204 DATA "■GET","■READ","■CLR","■REM","■TAN(","■CLOSE"
206 DATA "■INPUT","■DATA","■LIST","■NEW","■RUN","■FN","■CMD"
207 DATA "■FRE(0)","■GET#","■RESTORE","■END","■LOAD"
208 DATA "■DEFFN","■WAIT","■INPUT#","■ON","■STOP"
209 DATA "■PRINT#","■TAB(","■NOT","■AND","■POKE","■IF"
210 DATA "■GOSUB","■STEP","■TO","■OR","■DIM","■THEN"
212 DATA "■RETURN","■GOTO","■NEXT","■FOR","■SYS(","■LEN("
```

```
214 DATA "■VAL(","■CHR$(","■LEFT$(","■USR(","■PEEK("
216 DATA "■STR$(","■ASC(","■RIGHT$(","■MID$(","***"
218 DATA 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
1000 REM
1001 REM ****************************
1002 REM *                          *
1003 REM *    MACHINE LANGUAGE PGM  *
1004 REM *                          *
1005 REM ****************************
1006 REM
1220 DATA 78,A9,00,85,86,A9,19,85,87,A9,AE,8D,19,02,A9,1A,8D,1A,02,58,6
1222 DATA 78,A9,85,8D,19,02,A9,E6,8D,1A,02,58,60
1224 DATA A9,00,48,48,48,48,4C,85,E6
1226 DATA 4C,7E,E6
1228 DATA 20,A2,1A,EA,AD,03,02,C9,FF,F0,F2,C9,1B,F0,2D,C9,24,F0,29,C5,F
1230 DATA AD,04,02,F0,E1,AD,03,02,85,FE,AA,E9,37,30,11,A0,1A
1232 DATA BD,00,1B,F0,D0,20,27,CA,4C,AB,1A,00,00,00,00
1234 DATA A0,19,4C,D6,1A                                    ↳F0,E
1236 DATA A9,00,85,FE,F0,BB
1238 DATA 00
1240 DATA 01,07,00,00,0D,79,13,00
1242 DATA 19,1F,00,25,00,2B,31,00
1244 DATA 37,3D,00,43,4A,4F,55,5A
1246 DATA 5F,65,00,6C,73,92,7F,84
1248 DATA 89,8D,00,00,9A,A0,A9,AE
1250 DATA B4,BB,00,C1,C9,CD,D3,DB
1252 DATA E1,E6,00,EB,F1,F5,01,07
1254 DATA 0B,0F,00,14,1A,22,28,2E
1256 DATA 00,00,00,33,39,3F,45,4C
1258 DATA 00,00,54,5A,61,68,6E,77
1260 DATA +++
1500 SYS(6784):PRINT"VOR BAND-OPERATIONEN AUSSCHALTEN MIT:"
1510 PRINTTAB(10);"*** SYS(6805) ***"
1520 NEW
```

PROGRAM MAPPING



```
10 KEYWORDS TO SHIFTED KEYS
15 ========================
20 FOR COMMODORE PET 2001
25 !
30 !
35 COPYRIGHT WOLFGANG RADELOFF
40 PINNEBERG  20.1.1980
45 !
50 !
60 IRVEC    = $0219
70 IRSUB    = $E685
75 IRRTI    = $E67E
80 KEYIN    = $0203
85 SHFLAG   = $0204
90 FEREG    = $00FE
95 PETPR    = $CA27
96 !
100 *=1900
105 !
106 LABEL 'KEYWORD'
107 !
110 .BYTE $00:.TEXT "■COS(":.BYTE $00
115 .TEXT "■SGN(":.BYTE $00
120 .TEXT "■SQR(":.BYTE $00
125 .TEXT "■INT(":.BYTE $00
130 .TEXT "■SIN(":.BYTE $00
135 .TEXT "■LOG(":.BYTE $00
140 .TEXT "■EXP(":.BYTE $00
145 .TEXT "■ABS(":.BYTE $00
150 .TEXT "■RND(":.BYTE $00
155 .TEXT "■ATN(":.BYTE $00
160 .TEXT "■OPEN":.BYTE $00
165 .TEXT "■PRINT":.BYTE $00
170 .TEXT "■GET":.BYTE $00
175 .TEXT "■READ":.BYTE $00
180 .TEXT "■CLR":.BYTE $00
185 .TEXT "■REM":.BYTE $00
190 .TEXT "■TAN(":.BYTE $00
195 .TEXT "■CLOSE":.BYTE $00
200 .TEXT "■INPUT":.BYTE $00
205 .TEXT "■DATA":.BYTE $00
210 .TEXT "■LIST":.BYTE $00
215 .TEXT "■NEW":.BYTE $00
220 .TEXT "■RUN":.BYTE $00
225 .TEXT "■FN":.BYTE $00
230 .TEXT "■CMD":.BYTE $00
235 .TEXT "■FRE(0)":.BYTE $00
240 .TEXT "■GET#":.BYTE $00
245 .TEXT "■RESTORE":.BYTE $00
250 .TEXT "■END":.BYTE $00
255 .TEXT "■LOAD":.BYTE $00
260 .TEXT "■DEFFN":.BYTE $00
265 .TEXT "■WAIT":.BYTE $00
270 .TEXT "■INPUT#":.BYTE $00
275 .TEXT "■ON":.BYTE $00
280 .TEXT "■STOP":.BYTE $00
285 .TEXT "■PRINT#":.BYTE $00
290 .TEXT "■TAB(":.BYTE $00
295 .TEXT "■NOT":.BYTE $00
300 .TEXT "■AND":.BYTE $00
305 .TEXT "■POKE":.BYTE $00
310 .TEXT "■IF":.BYTE $00
315 .TEXT "■GOSUB":.BYTE $00
320 .TEXT "■TO":.BYTE $00
325 .BYTE $00:TEXT"■STEP":.BYTE $00
330 .TEXT "■OR":.BYTE $00
335 .TEXT "■DIM":.BYTE $00
340 .TEXT "■THEN":.BYTE $00
345 .TEXT "■RETURN":.BYTE $00
350 .TEXT "■GOTO":.BYTE $00
355 .TEXT "■NEXT":.BYTE $00
360 .TEXT "■FOR":.BYTE $00
365 .TEXT "■SYS(":.BYTE $00
370 .TEXT "■LEN(":.BYTE $00
375 .TEXT "■VAL(":.BYTE $00
380 .TEXT "■CHR$(":.BYTE $00
385 .TEXT "■LEFT$(":.BYTE $00
390 .TEXT "■USR(":.BYTE $00
395 .TEXT "■PEEK(":.BYTE $00
400 .TEXT "■STR$(":.BYTE $00
405 .TEXT "■ASC(":.BYTE $00
410 .TEXT "■RIGHT$(":.BYTE $00
415 .TEXT "■MID$(":.BYTE $00
420 !
425 !
```

REDAKTIONS- UND ANZEIGENSCHLUSS
FÜR HEFT 13
AM 6. JUNI 1980, FREITAG.

```
430 LABEL 'TABL'
435 !
440 !
445 .BYTE $00,$01,$07,$00,$00,$0D,$79,$13,$00
450 .BYTE $19,$1F,$00,$25,$00,$2B,$31,$00
455 .BYTE $37,$3D,$00,$43,$4A,$4F,$55,$5A
460 .BYTE $5F,$65,$00,$6C,$73,$92,$7F,$84
465 .BYTE $89,$8D,$00,$00,$9A,$A0,$A9,$AE
470 .BYTE $B4,$BB,$00,$C1,$C9,$CD,$D3,$DB
475 .BYTE $E1,$E6,$00,$EB,$F1,$F5,$FC,$01
480 .BYTE $07,$0B,$00,$10,$16,$1E,$24,$2A
485 .BYTE $00,$00,$00,$2F,$35,$3B,$41,$48
490 .BYTE $00,$00,$50,$56,$5D,$64,$6A,$73
```

```
505 LABEL 'ONOFF'
510 !
515 SEI
520 LDA IRVEC
525 EOR #$62
530 STA IRVEC
535 LDA IRVEC+1
540 EOR #$FC
545 STA IRVEC+1
550 CLI
555 RTS
560 !
565 LABEL 'STAX'
570 !
575 LDA #$00
580 PHA
585 PHA
590 PHA
595 PHA
600 JMP IRSUB
605 !
610 LABEL 'TKEY'
615 !
620 JSR STAX
625 NOP
630 LDA KEYIN
635 CMP #$FF
640 BEQ END
645 CMP FEREG
650 BEQ END
655 CMP ':'
660 BEQ CLEAR
```

```
665 CMP '[CR]'
670 BNE PKEY
675 !
680 LABEL 'CLEAR'
685 !
690 LDA #$00
695 STA FEREG
700 BEQ END
705 !
710 LABEL 'PKEY'
715 !
720 LDA SHFLAG
725 BEQ END
730 LDY #$1A
735 LDX KEYIN
740 STX FEREG
745 CPX #$38
750 BCS READTABL
755 INY
760 !
765 LABEL 'READTABL'
770 !
775 LDATABL,X
780 BEQ END
785 JSR PETPR
790 !
795 LABEL 'END'
800 !
805 JMP IRRTI:.END
810 *
815 *
```

```
820 *SYMBOLTABLE
825 *===========
830 *
840 *CLEAR=$1AFE    END=$1B1D       FEREG=$00FE
845 *IRRTI=$E67E    IRSUB=$E685     IRVEC=$0219
850 *KEYIN=$0203    KEYWORD=$1900   ONOFF=$1ACB
855 *PKEY=$1B04     READTABL=$1B15  STAX=$1ADE
860 *TABL=$1A7A     TKEY=$1AE7
865 *
870 *ERRORS:0
875 *==========
880 *UNDEFINED LABELS:0
885 *====================
890 *END
```

Ing. (grad.) Horst Steder, Talstr. 10 b, 6000 Frankfurt 56

Cross Reference Table für den AIM 65-Assembler

Seit einigen Monaten verwendet der Verfasser zur Dokumentation der entwickelten Programme eine alphanumerisch sortierte Symboltafel. Obwohl diese schon eine wesentliche Hilfe zum Auffinden von Symboladressen darstellt, entstand bald der Wunsch, eine echte Cross Reference Table (CRT) zu verwenden. Das vorliegende Programm stellt einen ersten Versuch einer Erweiterung der Symbol Table in eine CRT dar (hat denn noch keiner den Assembler geknackt?).

Zur Wirkungsweise

Nach vollendetem Assembler-Durchlauf wird zuerst die Symbol-Table alphanumerisch sortiert und dann ein Symbol nach dem anderen über den Disassembler mit dem erzeugten Object-Programm verglichen. Wenn die entsprechende Symboladresse (Definition) im Programm aufgefunden wird, erfolgt der Ausdruck der entsprechenden Programmadresse als Referenz. Da für jedes Symbol das Programm einmal komplett durchlaufen wird, kann es bei längeren Programmen durchaus bis zu 20 Min. bis zur Fertigstellung der Liste dauern (man kann mit einer Disassemblergeschwindigkeit von ca. 25 Befehlen/sec. rechnen).

Auf einige Besonderheiten sei noch hingewiesen: Da die Referenzen über den Disassembler effektiv aus dem Objektprogramm generiert werden, können Symbole mit Rechenoperatoren (z.B. DIBUFF+13) nicht erkannt werden. Hier kann man sich aber helfen, indem man nach dem Assemblerdurchlauf der Symbol-Tafel manuell noch entsprechende Symbole mit ihrer Adresse anfügt. Es ergibt sich auch eine hervorragende Möglichkeit, unbekannte Maschinenprogramme zu untersuchen, indem man eine Symboltafel erstellt (die entsprechenden Vektoren in 32, 33, 3A, 3B, OB und OC nicht vergessen).

Weiterhin konnte der Verfasser in der Zeropage nirgends die Startadresse des erzeugten Objektprogrammes finden. Sie muß also beim Start der Routine eingegeben werden. Außerdem wurde vom Zähler für die Zahl der Symbole ($OB und $OC) nur das Byte Low ($OC) verwandt, weil eine Anzahl von mehr als 255 wohl sehr selten vorkommen dürfte. Das kann aber sehr leicht in der Sortier- und Abfrageroutine geändert werden.

Ein weiteres Problem stellte die Darstellung von Direktoperanden dar, weil hier jedes Adreßbyte, das zufällig mit dem Wert des Operanden übereinstimmt, als Referenz ausgedruckt wird. Das ist zwar grundsätzlich richtig, führt aber u.U. zu einer Häufung von Pseudo-Referenzen. Besonders stark zeigt sich dieser Effekt bei dem häufig vorkommenden Operanden '00' und Zeropage-Adressen (High-Byte =00). Der Verfasser wendet daher folgendes Verfahren an: Bei Zeropageadressen und Direktoperanden wird das Adreßbyte High nicht abgefragt. Außerdem wird jede Referenz eines Direktoperanden mit '#' gekennzeichnet, um sie in der Liste sofort erkennen zu können. Dies ist nur ein Weg, der Phantasie sind keine Grenzen gesetzt, und der Verfasser wäre für ein Feedback und für Anregungen aus dem Anwenderkreis dankbar.

Das Ausgabeformat wurde für Drucker mit mindestens 72 Zeichen Zeilenbreite konzipiert, wobei in einer Zeile neben Symbol und Definition bis zu 8 Referenzen ausgedruckt werden. Bei mehr als 8 Referenzen wird die folgende Zeile zur übersichtlichen Darstellung entsprechend eingerückt. Weitere Einzelheiten können den Programm-Kommentaren entnommen werden.

Zur Bedienung

Nach dem Start (in diesem Fall mit F3) erfolgt der Prompt "FROM", der mit der Startadresse des generierten Programms beantwortet wird. Danach erfolgt der Ausdruck der CRT über die entsprechende USER-Routine bis zum bitteren Ende. Der Ablauf kann jederzeit durch Drücken der 'SPACE'-Taste unterbrochen und durch Drücken einer anderen Taste fortgesetzt werden. Beim Betätigen der 'ESCAPE'-Taste erfolgt Aussprung in den Monitor.

Der Verfasser hofft, mit diesem Programm Anregungen zum weiteren Experimentieren geben zu können und ist für Hinweise und Kritik dankbar.

```
0000          ;       ************************************
0000          ;       *                                  *
0000          ;       *      PROGRAM    < C R S R F >    *
0000          ;       *                                  *
0000          ;       * V 1.6              H.STEIER 1/80 *
0000          ;       ************************************
0000
0000          ;THIS PROGRAM GENERATES A CROSS REFERENCE TABLE OF ALL
0000          ;SYMBOLS IN ALPHANUMERIC ORDER AFTER COMPLETED ASSEMBLY.
0000
0000 ENDADR           =$0032
0000 STRFLG           =$0051
0000 ADL              =$0052
0000 ADH              =$0053
0000 SYMVLL           =$0054
0000 SYMVLH           =$0055
0000 SWPFLG           =$0056
0000 RFCNT            =$0057
0000 SAVPC            =$A425
0000 DIBUFF           =$A438          ;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0000 USROUT           =$5EFF <-----   ;X     USER OUTPUT ROUTINE     X
0000 AIM65            =$E182          ;XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
0000 CGPC1            =$E5D7
0000 FROM             =$E7A3
0000 BLANK2           =$E83B
0000 BLANK            =$E83E
0000 KBSCAN           =$E90C
0000 CRCK2            =$EA3B
0000 PACK             =$EA84
0000 OUTPUT           =$EF02
0000 CRLF             =$E9F0
0000 WRAX             =$EA42
0000 NUMA             =$EA46
0000 DISASM           =$F46C
0000
0000                  *=$0112         ;FUNCTION KEY <F3>
0112          4C0002  JMP START
0115
0115                  *=$0200
0200
0200 START    20A3E7  JSR FROM        ;GET OBJECT START ADDRESS
0203          20C003  JSR SETOUT      ;SET DILINK TO OUTPUT ROUTINE
0206          20F0E9  JSR CRLF
0209          20F0E9  JSR CRLF
020C          A21B    LDX #27
020E HDPRT    BDE403  LDA HEADER,X    ;PRINT HEADER
0211          2002EF  JSR OUTPUT
0214          CA      DEX
0215          10F7    BPL HDPRT
```

```
0217          20F0E9 JSR CRLF
021A          20CB03 JSR RSTORE      ;DILINK BACK TO MONITOR DISPLAY
021D
021D          ; ***  START OF CROSS REFERENCE ROUTINE  ***
021D
021D          20B002 JSR SYMSRT      ;SORT OF SYMBOL TABLE
0220          A60C   LDX #0C         ;SAVE # OF SYMBOLS
0222          8650   STX #50
0224 NXTSYM   209702 JSR GETSYM      ;GET SYMBOL AND PRINT IT
0227          203BE8 JSR BLANK2
022A          A900   LDA #0
022C          8557   STA RFCNT       ;RESET REF COUNTER
022E          20D7E5 JSR CGPC1       ;GET PROGRAM START ADDR
0231 NXTIN    A92E   LDA #'.'
0233          8D45A4 STA DIBUFF+13   ;TO FIND REGISTER OP'S...
0236          8D47A4 STA DIBUFF+15   ;OR NON-INDEXED 0-PAGE OP'S
0239          206CF4 JSR DISASM      ;DISASSEMBLE INSTRUCTION
023C          200CE9 JSR KBSCAN      ;HALT ON <SPACE>, EXIT ON <ESCAPE>
023F          201F03 JSR COMPAR      ;CHECK FOR REF + PRINT IF MATCH
0242          203BEA JSR CRCK2       ;RESET DIBUFF POINTER
0245          204C02 JSR NXTADR      ;GET NEXT ADDRESS
0248          90E7   BCC NXTIN
024A          B0D8   BCS NXTSYM      ;LAST ADDR, GET NEXT SYMBOL
024C
024C          ;     ***  SUBROUTINES  ***
024C
024C N.ISADR  AD26A4 LDA SAVPC+1     ;GET ADDR AND CHECK FOR END
024F          C533   CMP ENDADR+1
0251          D00A   BNE ADDS
0253          AD25A4 LDA SAVPC
0256          C532   CMP ENDADR
0258          D003   BNE ADDS
025A          38     SEC             ;SET CARRY IF LAST ADDR
025B          B00F   BCS RET1
025D ADDS     AD25A4 LDA SAVPC
0260          38     SEC
0261          65EA   ADC $EA         ;ADD INSTRUCTION LENTGH
0263          8D25A4 STA SAVPC
0266          9004   BCC RET1
0268          EE26A4 INC SAVPC+1
026B          18     CLC
026C RET1     60     RTS
026D
026D SYMBL    20C003 JSR SETOUT      ;DILINK TO OUTPUT DEV
0270          20F0E9 JSR CRLF
0273          A000   LDY #0
0275 SYMB     B140   LDA ($40),Y     ;DISPLAY SYMBOL (6 CHARS)
0277          2002EF JSR OUTPUT
027A          C8     INY
027B          C006   CPY #6
027D          D0F6   BNE SYMB
027F          203BE8 JSR BLANK2
0282          B140   LDA ($40),Y
0284          8555   STA SYMVLH      ;SAVE SYMBOL ADDR HIGH
0286          2046EA JSR NUMA        ;DISPLAY SYMBOL ADDRESS
0289          C8     INY
028A          B140   LDA ($40),Y
028C          8554   STA SYMVLL      ;SAVE SYMBOL ADDR LOW
028E          2046EA JSR NUMA
0291          203BE8 JSR BLANK2
0294          4CCB03 JMP RSTORE
0297
```

```
0297 GETSYM  20F0E9 JSR CRLF        ;NEW LINE
029A         C650   DEC $50         ;LAST SYMBOL?
029C         100C   BPL *+14         ;NO, GO ON
029E         20C003 JSR SETOUT
02A1         20F0E9 JSR CRLF        ;YES, SEND CR/LF AND...
02A4         20CB03 JSR RSTORE      ;RETURN TO MONITOR
02A7         4C82E1 JMP AIM65
02AA         206D02 JSR SYMBL       ;PRINT SYMBOL + ADDRESS
02AD         4C0803 JMP ADRADD      ;GET NEXT SYMBOL ADDR
02B0
02B0         ;ALPHANUMERIC SORT OF SYMBOL TABLE
02B0 SYMSRT  20D802 JSR SAVTB       ;SAVE START ADDR OF SYMBOLTAB
02B3         A60C   LDX $0C         ;TRANSFER # OF SYMBOLS TO X-REG
02B5         CA     DEX
02B6         A000   LDY #0
02B8         8456   STY SWPFLG      ;CLEAR SWAP FLAG
02BA NXT1    A0FF   LDY ##FF
02BC NXT2    C8     INY
02BD         B140   LDA ($40),Y     ;LOAD CHAR OF SYMBOL
02BF         D142   CMP ($42),Y     ;COMP WITH CHAR OF NEXT SYMBOL
02C1         F007   BEQ NXT3        ;EQUAL, GET NEXT CHAR
02C3         9009   BCC ADD         ;1ST SYMBOL SMALLER, NO SWAP
02C5         20EC02 JSR SWPSYM      ;2ND SYMBOL SMALLER, SWAP SYMBOLS
02C8         B004   BCS ADD         ;JUMP ALWAYS
02CA NXT3    C006   CPY #6          ;LAST CHAR OF SYMBOL?
02CC         D0EE   BNE NXT2        ;NO, GET NEXT CHAR
02CE ADD     200803 JSR ADRADD      ;YES, GET NEXT SYMBOL
02D1         CA     DEX             ;LAST SYMBOL?
02D2         D0E6   BNE NXT1        ;NO, GET NEXT ONE
02D4         A556   LDA SWPFLG      ;SWAP FLAG SET?
02D6         D0D8   BNE SYMSRT      ;YES, START ALL OVER AGAIN
02D8
02D8 SAVTB   18     CLC
02D9         A53A   LDA $3A         ;SAVE START ADDRESS OF SYMBOLTABLE
02DB         8540   STA $40
02DD         6908   ADC #8
02DF         8542   STA $42         ;ALSO ADDRESS OF NEXT SYMBOL
02E1         A53B   LDA $3B
02E3         8541   STA $41
02E5         8543   STA $43
02E7         9002   BCC *+4
02E9         E643   INC $43
02EB         60     RTS
02EC
02EC         ;SWAP SUBROUTINE
02EC SWPSYM  98     TYA             ;SAVE Y-REG
02ED         48     PHA
02EE         A000   LDY #0
02F0 SWP1    B140   LDA ($40),Y     ;SWAP SYMBOLS CHAR BY CHAR
02F2         8550   STA $50         ;USING LOCATION $50
02F4         B142   LDA ($42),Y     ;AS INTERMEDIATE BUFFER
02F6         9140   STA ($40),Y
02F8         A550   LDA $50
02FA         9142   STA ($42),Y
02FC         C8     INY
02FD         C008   CPY #8
02FF         D0EF   BNE SWP1
0301         A901   LDA #1
0303         8556   STA SWPFLG      ;SET SWAP FLAG
0305         68     PLA
0306         A8     TAY             ;RESTORE Y-REG
0307         60     RTS
0308
```

```
0308 ADRADD  A540   LDA $40         ;INCREMENT TABLE ADDRESS
030A         18     CLC             ;TO NEXT SYMBOL
030B         6908   ADC #8
030D         8540   STA $40
030F         9002   BCC *+4
0311         E641   INC $41
0313         18     CLC
0314         A540   LDA $40         ;ALSO FOR FOLLOWING SYMBOL
0316         6908   ADC #8
0318         8542   STA $42
031A         9002   BCC *+4
031C         E643   INC $43
031E RET2    60     RTS
031F
031F COMPAR  A200   LDX #0          ;CHECK FOR OPERAND TYPE
0321         8651   STX STRFLG      ;CLEAR #-FLAG
0323         A20D   LDX #13
0325         BD38A4 LDA DIBUFF,X
0328         C92E   CMP #'.'         ;REGISTER OPERAND?
032A         F0F2   BEQ RET2

032C         C923   CMP #'#'         ;IMMEDIATE OPERAND?
032E         F032   BEQ IMMED
0330         C928   CMP #'('         ;INDIRECT OP ?
0332         F00D   BEQ NXTST
0334         BD3AA4 LDA DIBUFF+2,X
0337         C92E   CMP #'.'         ;SINGLE BYTE OP ?
0339         F021   BEQ ZROPGE
033B         C92C   CMP #','         ;ALSO SINGLE BYTE, INDEXED?
033D         F01D   BEQ ZROPGE
033F         D00C   BNE ABSLTE       ;NO, 2-BYTE OP
0341 NXTST   E8     INX             ;INC POINTER
0342         BD3AA4 LDA DIBUFF+2,X
0345         C92C   CMP #','         ;SINGLE BYTE INDIRECT?
0347         F013   BEQ ZROPGE
0349         C929   CMP #')'         ;SAME?
034B         F00F   BEQ ZROPGE
034D ABSLTE  20B403 JSR PACK1       ;GET ADDR HIGH
0350         8553   STA ADH
0352         E8     INX
0353 GTAD    E8     INX
0354 GTADL   20B403 JSR PACK1       ;GET ADDR LOW
0357         8552   STA ADL
0359         4C6803 JMP COMPR
035C ZROPGE  A900   LDA #0
035E         8553   STA ADH
0360         F0F2   BEQ GTADL
0362 IMMED   A980   LDA #$80
0364         8551   STA STRFLG      ;SET #-FLAG
0366         D0EB   BNE GTAD
0368
0368 COMPR   2451   BIT STRFLG      ;IMMEDIATE OP?
036A         100E   BPL CMPR        ;NO, GO ON
036C         A555   LDA SYMVLH      ;IF ZERO-PAGE (SYMVLH=00),
036E         F00A   BEQ CMPR        ;SKIP TO AVOID TOO MANY...
0370         C552   CMP ADL         ;PSEUDO-REFERENCES
0372         F012   BEQ PRTREF
0374         A554   LDA SYMVLL
0376         C552   CMP ADL
0378         F00C   BEQ PRTREF
037A CMPR    A554   LDA SYMVLL
037C         C552   CMP ADL
037E         D09E   BNE RET2
```

```
0380          A555   LDA SYMVLH
0382          C553   CMP ADH
0384          D098   BNE RET2
0386
0386 PRTREF   20C003 JSR SETOUT      ;DILINK TO OUTPUT DEV
0389          203BE8 JSR BLANK2
038C          A557   LDA RFCNT
038E          C908   CMP #8           ;MAX # OF REFS IN LINE ? (MAX=8)
0390          9003   BCC PRTRF       ;NOT YET
0392          20D603 JSR NEWLIN      ;YES, START NEW LINE
0395 PRTRF    2451   BIT STRFLG      ;#-FLAG SET?
0397          1005   BPL **+7        ;NO
0399          A923   LDA #'#'        ;FLAG SET, IS IMMEDIATE OP
039B          2002EF JSR OUTPUT      ;PRINT '#'
039E          D003   BNE **+5
03A0          203EE8 JSR BLANK
03A3          20AB03 JSR FRMPRT      ;PRINT REFERENCE ADDR
03A6          E657   INC RFCNT       ;INCREMENT REF COUNTER
03A8          4CCB03 JMP RSTORE      ;RESET DILINK
03AB
03AB FRMPRT   AD26A4 LDA SAVPC+1     ;PRINT REFERENCE
03AE          AE25A4 LDX SAVPC
03B1          4C42EA JMP WRAX

03B4 PACK1    BD38A4 LDA DIBUFF,X    ;GET DATA FROM DIBUFF...
03B7          2084EA JSR PACK        ;IN PACKED FORMAT TO ACCU
03BA          BD39A4 LDA DIBUFF+1,X
03BD          4C84EA JMP PACK
03C0
03C0 SETOUT   A9FF   LDA #<USROUT    ;SET DILINK VECTOR TO...
03C2          8D06A4 STA $A406       ;USER OUTPUT DEVICE
03C5          A95E   LDA #>USROUT
03C7          8D07A4 STA $A407
03CA          60     RTS
03CB
03CB RSTORE   A905   LDA #5          ;SET DILINK VECTOR...
03CD          8D06A4 STA $A406       ;BACK TO AIM-65 DISPLAY
03D0          A9EF   LDA #$EF
03D2          8D07A4 STA $A407
03D5          60     RTS
03D6
03D6 NEWLIN   20F0E9 JSR CRLF        ;START A NEW LINE
03D9          A208   LDX #8
03DB          203BE8 JSR BLANK2      ;INSERT 16 SPACES...
03DE          CA     DEX             ;FOR CLEAN FORMATTING
03DF          D0FA   BNE *-4
03E1          8657   STX RFCNT       ;RESET REF COUNTER
03E3          60     RTS
03E4
03E4 HEADER   3A53   .BYT ':SECNEREFER     .FED  LOBMYS'
0400
0400
0400                 .END
0400           ERRORS= 0000
```

```
                    **  SYMBOL TABLE  **

ASSLTE  034D     ADD     02CE     ADDS    025D     ADH     0053
ADL     0052     ADPADD  0308     AIM65   E182     BLANK   E83E
BLANK2  E83B     CGPC1   E5D7     CMPR    037A     COMPAR  031F
COMPR   0368     CRCK2   EA3B     CRLF    E9F0     DIBUFF  A438
```

DISASM	F46C	ENDADR	0032	FRMPRT	03AB	FROM	E7A3
GETSYM	0297	GTAD	0353	GTADL	0354	HDPRT	020E
HEADER	03E4	IMMED	0362	KBSCAN	E90C	NEWLIN	03D6
NUMA	EA46	NXT1	02BA	NXT2	02BC	NXT3	02CA
NXTADR	024C	NXTIN	0231	NXTST	0341	NXTSYM	0224
OUTPUT	EF02	PACK	EA84	PACK1	03B4	PRTREF	0386
PRTRF	0395	RET1	026C	RET2	031E	RFCNT	0057
RSTORE	03CB	SAVPC	A425	SAVTB	02D8	SETOUT	03C0
START	0200	STRFLG	0051	SWP1	02F0	SWPFLG	0056
SWPSYM	02EC	SYMB	0275	SYMBL	026D	SYMSRT	02B0
SYMVLH	0055	SYMVLL	0054	USROUT	5EFF	WRAX	EA42
ZROPGE	035C						

SYMBOL	DEF.	REFERENCES:							
ABSLTE	034D	033F							
ADD	02CE	02C3	02C8						
ADDS	025D	0251	0258						
ADH	0053	0350	035E	0382					
ADL	0052	0357	0370	0376	037C	03ED			
ADRADD	0308	02AD	02CE	#02DD	#02FD	#030B	#0316	#038E	#03D9
AIM65	E182	02A7							
BLANK	E83E	03A0							
BLANK2	E83B	0227	027F	0291	0389	03DB			
CGPC1	E5D7	022E							
CMPR	037A	036A	036E						
COMPAR	031F	023F							
COMPR	0368	0359							
CRCK2	EA3B	0242							
CRLF	E9F0	0206	0209	0217	0270	0297	02A1	03D6	
DIBUFF	A438	0325	03B4						
DISASM	F46C	0239							
ENDADR	0032	0256							
FRMPRT	03AB	03A3							
FROM	E7A3	0200							
GETSYM	0297	0224							
GTAD	0353	0366							
GTADL	0354	0360							
HDPRT	020E	0215							
HEADER	03E4	020E							
IMMED	0362	032E							
KBSCAN	E90C	023C							
NEWLIN	03D6	0392							
NUMA	EA46	0286	028E						
NXT1	02BA	02D2							
NXT2	02BC	02CC							
NXT3	02CA	02C1							
NXTADR	024C	0245							
NXTIN	0231	0248							
NXTST	0341	0332							
NXTSYM	0224	024A							
OUTPUT	EF02	0211	0277	039B	#03D0				
PACK	EA84	03B7	03BD						
PACK1	03B4	034D	0354						
PRTREF	0386	0372	0378						
PRTRF	0395	0390							
RET1	026C	025B	0266						
RET2	031E	031A	032A	037E	0384				
RFCNT	0057	022C	038C	03A6	03E1				
RSTORE	03CB	021A	0294	02A4	03A8				
SAVPC	A425	0253	025D	0263	03AE				
SAVTB	02D8	02B0							

```
SETOUT  03C0    0203    026D    029E    0386
START   0200   #022A   #0273   #02B6   #02EE  #031F  #035C
STRFLG  0051    0321    0364    0368    0395
SWP1    02F0    02FF
SWPFLG  0056    02B8    02D4    0303
SWPSYM  02EC    02C5
SYMB    0275    027D
SYMBL   026D    02AA
SYMSRT  02B0    021D    02D6
SYMVLH  0055    0284    036C    0380
SYMVLL  0054    028C    0374    037A
USROUT  5EFF   #02BA   #03C0   #03C5
WRAX    EA42    03B1
ZROPGE  035C    0339    033D    0347    034B
```

****

Uwe Pitz, Am Schwarzen Berge 38, 3300 Braunschweig

Die Challengers von OSI

Der Artikel in 65xx MICRO MAG 11, S. 44 von Herrn Löhr veranlaßt mich, das etwas schiefe Bild, das er vom CHALLENGER zeichnet, etwas zurechtzurücken.

Vorweg möchte ich die Unterschiede zwischen dem C2-4P/8P und dem C1P skizzieren. Unterschiede bestehen fast nur in der Hardware, z.B. liegen beim 4P/8P Monitor und Support in drei PROMs 1702A, während beim C1P dafür ein PROM 2716 verwendet wird. Der C2-4P ist aus 2 Karten 8" x 10" für den 48poligen OSI-Bus aufgebaut und um 1 bis 2 Karten erweiterbar (jedenfalls insgesamt 4 Steckplätze; entsprechend der C2-8P mit 8 Steckplätzen und getrennter Tastatur). Von diesen Karten wird eine reichhaltige Palette angeboten, die Lieferbarkeit scheint endlich auch gesichert zu sein. Die Grundausstattung: CPU-Platine, 8k BASIC, 8k RAM, RS 232/20 mA Serialschnittstelle mit jumperbarer und feineinstellaber Baudrate (für Cassettenbetrieb ist korrekte Einstellung wichtig!), Parallelschnittstelle 6821 ist vorgesehen.

Die Video-Platine: Zeichengenerator für 256 Zeichen, 4k Bildwiederholspeicher für Farbbetrieb (bessere Qualität als APPLE), Bildschirmformat 64x32/32x32, softwaremäßig umschaltbar, Cassettenausgang.

Demgegenüber der 1P/Superboard 2: Kompletter Einplatinencomputer mit 8k BASIC/8k RAM, Tastatur. Innerhalb des Gehäuses nur einmal durch Huckepackplatine um 24k RAM, Floppycontroller und Echtzeituhr zu erweitern (Netzteil muß jedoch aufgerüstet werden), darüberhinaus mit Aufwand. Zur Vergrößerung der Zeichenzahl von 24 pro Zeile gibt es angeblich bereits Umbauanleitungen oder -sätze.

Die Unterstützung eines OSI-Betreibers gab es lange Zeit nur auf dem Papier (in Form von Prospekten und Preislisten). Erst in letzter Zeit scheint es mit Lieferung von Hard- und Software sowie Beratung und Service zu klappen.

Bemerkungen zum BASIC

Zwar ist die Ein-/Ausgabe zur Cassette sehr primitiv und verlangt viel zusätzlichen Aufwand, jedoch gerade darin verbergen sich auch Vorteile: Man kann mehrere Programme von verschiedenen Cassetten einlesen, die dann gleichzeitig im Speicher resident sind (z.B. RENUMBER-Programm, Suchpro-

gramm, Programm zum Einbau von Maschinenprogrammen in ein BASIC-Programm usw.). In ein ausgestiegenes Programm kann man nach Hardware-Reset über die Taste 'W' wieder einsteigen.

Betrieb in Maschinensprache

Der Monitor im ROM ist wirklich sehr dürftig, doch ein negatives Urteil darüber ist nicht gerechtfertigt: Laut den Unterlagen von OSI ist ein 'Extended Monitor' mit 2k Bytes Teil des Gesamtmonitors. Der Grund für diese merkwürdige Aufteilung dürfte darin liegen, daß das CHALLENGER-Konzept schon mehrere Jahre alt ist (damals offenbar sehr hoher Preis für PROMs - es gab nur den 1702/1) und OHIO SCIENTIFIC sich ziemlich streng an das Prinzip der Hardware-/Softwarekompatibiltität als einziger Lieferant halten mußte. Auch die Ungereimtheiten mit den Graphikzeichen in den BASIC-Fehlermeldungen sind übrigens so zu erklären. Der 'Extended Monitor' enthält Kommandos wie FILL, RELOCATE, MOVE, Disassemblieren, arithmetische Operationen in Hexadezimal, Anzeige/Änderung von memory contents, Akku und Registern, Breakpoint-Handling, Cassetten-Operationen usw.. Es ist daher unverständlich, warum der 'Extended Monitor' nicht von vornherein im Lieferumfang einbegriffen ist. Niedrigpreis um jeden Preis?

Als weiteres Dienstleistungsprogramm steht ein recht komfortabler Editor/Assembler zur Verfügung (ca. 6,5k). Eine OSI-Besonderheit ist der Auto-Loader, der auf OSI-Unterlagen beruht, aber nicht von OSI vertrieben wird. Mit diesem Cassettenprogramm kann man jedes Maschinenprogramm plus einem Bootstrap-Vorspann und einem Nachspann auf Cassette speichern. Beim Einspielen erweitert der Vorspann zunächst den Monitor im ROM, so daß die Maschine jetzt das Checksum-Format versteht, in dem alle OSI-Tapes geschrieben sind. Mit Hilfe des Nachspanns wird auf die Einsprungstelle des interessierenden Programmes gesprungen.

Was allerdings z.Zt. fast vollkommen fehlt und was man sehr vermißt, ist eine Dokumentation nach Art der AIM- und APPLE-Handbücher. Die disassemblierten Programme mühsam abzuschreiben, wie ich es begonnen habe, ist nicht gerade das Gelbe vom Ei. Inzwischen gibt es jedoch - nicht von OSI(!) - disassemblierte ROM-Listings zu kaufen, auch einige Programm-Einsprungsadressen sind veröffentlicht worden.

Alles in allem: Solide Maschinen, die leider (noch?) viel zu wenig unterstützt werden.

Literatur und Lieferanten, soweit mir bekannt:

1. Ohio Scientific's SMALL SYSTEMS JOURNAL. Als selbständige Zeitschrift 1978 eingestellt, enthält wertvolle Hinweise und Programme. Jetzt als Teil von KILOBAUD MICROCOMPUTING im Stil einer Werbeserie wiederbelebt. Vor kurzen jedoch eine Sondernummer, nur Programme enthaltend, erschienen. Möglicherweise sind noch einzelne Exemplare von 1977/78 bei OSI vorrätig.
2. CHALLENGER TIMES - Independent OSI Users Newsletter. Als Fortsetzung des alten SMALL SYSTEMS JOURNAL gedacht; angeblich seit Juli 1979. P.O.B 518, Newton Corner, MA 02158, USA.
3. AARDVARK TECHNICAL SERVICES, 1690 Bolton, Walled Lake, MI 48088, USA liefert verschiedene Spielprogramme auf Cassette, den oben erwähnten Auto-Loader, andere nützliche Programmcassetten in BASIC, wie RENUMBER, POKER ROUTINE MAKER, SEARCH, BRANCH FINDER usw., ferner ein DISASSEMBLED ROM LISTING und 20 Seiten Untersuchungen über das ROM BASIC. In der Preisliste finden sich auch nützliche Prograämmchen und Hinweise.
4. PEGASUS SOFTWARE 1981-A St. Louis Drive, Honolulu, Hawaii 96816, USA: XPLO for OHIO SCIENTIFIC COMPUTERS (unter Vorbehalt).
5. D&N MICRO PRODUCTS, Inc. 3932 Oakhurst Drive, Fort Wayne In 46815, USA, OSI-Kompatible Karten. - 6. PEEK(65), The Unofficial OSI Users Journal, 62 Southgate Ave., Anna-

 -polis, MD 21401, USA.

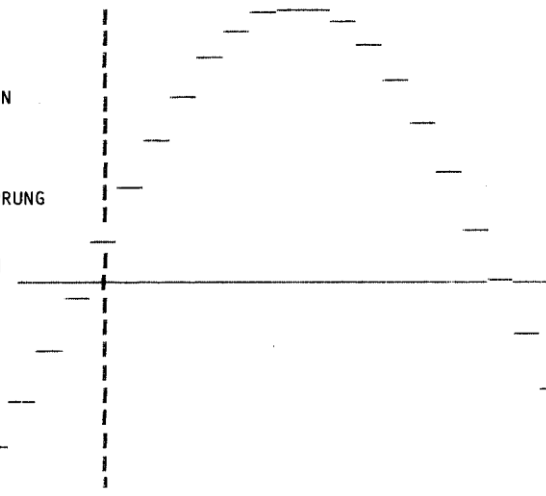
Dipl.-Ing. Uwe Kornnagel, Raunheim

- CBM - UNIPLOTT V 5.1

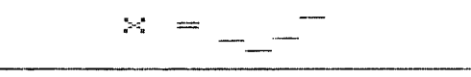
Programm zur mathematischen Kurvenanalyse in beliebigen Intervallen mit Wertetabelle, Darstellung auf dem Bildschirm, Bildung eines Ausschnittes, wahlweises Plotten auf dem CBM 3022-Drucker. Der in Heft 7 abgedruckte VIDEO DRIVER ist für den CBM adaptiert. Das Programm eignet für Unterrichtszwecke ebenso wie für die wissenschaftliche Arbeit.

Kommando-Menue:

```
     ├ VON -| BIS AUSSCHNITT
     E EINGABE DES INTERVALLS
     T WERTETABELLE DER FUNKTION
     P PLOTTEN DER FUNKTION
     L LÖSCHEN DER FUNKTION
     S SPIEGELUNG UM ACHSE/URSPRUNG
     G GESAMTFUNKTION
     ← BILDSCHIRMINHALT DRUCKEN
     RVS NEGATIVE VIDEO
```



Ein verkürzter Probeausdruck:



X =	Y =
-4	.756802495
-3.58974359	.433299807
-3.17948718	.037855457 1
-2.76923077	-.36381647
-2.35897436	-.705138387
-1.94871795	-.929433542
-1.53846154	-.999477276
-1.12820513	-.903644934
-.71794872	-.657841122
-.30769231	-.302860135
.102564101	.102384376
.512820511	.490636887
.923076921	.797461912
1.33333333	.971937901
1.74358974	.985108325
2.15384615	.834787385
2.56410256	.545922768
2.97435897	.166455264
3.38461538	-.240637628
3.79487179	-.607793617

```
100 REM -CBM 3001- UNIPLOTT V5.0
105 REM DEF FNY(X)=....
110 IFPEEK(634)<>169ANDPEEK(826)<>169THENGOSUB885
115 DIMX(40),Y(40):GOTO145
120 PR%=826:GOTO135:REM SET CRT
125 PR%=871:GOTO135:REM CLR&SET CRT
130 PR%=979:REM ACHSENKREUZ
135 POKE216,LN%:POKE198,SP%:SYSPR%
140 RETURN:REM AUSFUEHRUNG
145 REM *** UNIPLOTT START ***
150 PRINT"□";:LN%=0:SP%=5:GOSUB125:PRINT"-CBM 3001- UNIPLOTT V5.0"
155 IFPEEK(1060)=150GOTO200
160 LN%=3:GOSUB125:PRINT"BITTE Y = ■F(X)■ EINGEBEN."
165 LN%=5:SP%=0:OPEN1,0,0:GOSUB125:PRINT"Y = ";
170 INPUT#1,A$:CLOSE1
175 PRINT"□105 DEFFNY(X)=";A$
180 POKE623,ASC("■"):POKE624,13:POKE625,ASC("R")
185 POKE626,ASC("↗"):POKE627,13:POKE158,5
190 END
195 PRINT"□105 REM DEF FNY(X)=....":GOTO180
200 REM +++ DIRECTORY UNIPLOTT +++
205 PRINT"□";:LN%=0:SP%=5:GOSUB125:PRINT"-CBM 3001- UNIPLOTT V5.0"
210 LN%=2:SP%=2:GOSUB125:PRINT"■ MENUE -CBM 3001- UNIPLOTT■"
215 LN%=6:GOSUB125:PRINT"■E■ EINGABE DES INTERVALLS"
220 LN%=4:GOSUB125:PRINT"■H■ VON ■H■ BIS AUSCHNITT "
225 LN%=8:GOSUB125:PRINT"■T■ WERTETABELLE DER FUNKTION
230 LN%=10:GOSUB125:PRINT"■P■ PLOTTEN DER FUNKTION
235 LN%=12:GOSUB125:PRINT"■L■ LOESCHEN DER FUNKTION":LN%=20:GOSUB125
240 PRINT"■RVS■ NEGATIVE VIDEO"
245 LN%=16:GOSUB125:PRINT"■G■ GESAMTFUNKTION"
250 LN%=14:GOSUB125:PRINT"■S■ SPIEGLUNG UM ACHSE"
255 LN%=18:GOSUB125:PRINT"■←■ BILDSCHIRMINHALT DRUCKEN"
260 LN%=22:SP%=2:GOSUB125
265 PRINT"EINGABE ■E■; ■T■; ■P■; ■L■; ■G■; ■S■; ■←■ ODER ■RVS■ ";
270 SYS990:GETQ$
280 IFQ$="$"THENLOAD"MATH.PACK01",8
285 JP%=1
290 IFQ$="E"ORQ$="T"ORQ$="P"ORQ$="L"ORQ$="■"ORQ$="G"ORQ$="S"ORQ$="←"GOTO300
295 GOTO205
300 IFQ$="S"THENGOSUB760
301 IFQ$="←"GOTO705
305 IFQ$="L"GOTO195
310 IFQ$="G"THENVO=V0:BI=B1:PRINT"OK":GOSUB730
315 IFQ$="T"GOTO395
320 IFQ$="P"GOTO475
325 IFQ$="■"GOTO700
330 PRINT"□";:LN%=0:SP%=5:GOSUB125:PRINT"-CBM 3001- UNIPLOTT V5.0"
335 LN%=3:SP%=5:GOSUB125:PRINT"■INTERVALL EINGABE"
340 LN%=5:GOSUB125:PRINT"UNTERSUCHEN  VON ";
345 OPEN1,0,0:INPUT#1,A$:VO=VAL(A$)
350 LN%=7:SP%=18:GOSUB125:PRINT"BIS ";
355 INPUT#1,A$:BI=VAL(A$):CLOSE1
360 V0=VO:B1=BI:AX%=0
365 POKE216,8:SYS875:XP=-1E30:XM=1E30
370 DX=(BI-VO)/39:X0=VO
375 FORI=0TO39:X(I)=X0:Y(I)=FNY(X0)
380 IFY(I)>XPTHENLN%=10:SP%=5:GOSUB125:PRINT"MAX. ="X0;Y(I):XP=Y(I)
```

```
385 IFY(I)<XMTHENLN%=12:SP%=5:GOSUB125:PRINT"MIN. ="X0;Y(I):XM=Y(I)
390 X0=X0+DX:NEXT:GOTO205
395 PRINT"■":LN%=2:SP%=20:GOSUB130
400 JP%=2
405 LN%=0:SP%=5:GOSUB120:PRINT"X =":SP%=25:GOSUB120:PRINT"Y =":LN%=3
410 FORI=0TO39STEP2:SP%=0:GOSUB120:PRINTX(I)
415 SP%=21:GOSUB120:PRINTY(I):LN%=LN%+1:NEXT
420 FORI=21TO39:POKE198,I:SYS914:NEXT
425 WAIT158,1:GETQ$:IFQ$="■"THENGOSUB435
430 GOTO290
435 LN%=3:GOSUB470
440 WAIT158,1:GETQ$:GOSUB470:IFQ$="■"THENLN%=LN%+1:GOTO455
445 IFQ$="■"THENLN%=LN%-1:GOTO455
450 RETURN
455 IFLN%>22THENLN%=3
460 IFLN%<3THENLN%=22
465 GOSUB470:GOTO440
470 POKE216,LN%:SYS899:RETURN
475 AF%=0:EN%=0
480 JP%=3
485 PRINT"■":DY=XP-XM:DX=BI-VO:PR$="──────"
490 IFSGN(VO)=SGN(BI)THEN640
495 IFSGN(XP)=SGN(XM)THEN675
500 Y0=INT(24*XP/DY):AX%=Y0
505 X0=39-INT(39*BI/DX)
510 LN%=Y0:SP%=X0:GOSUB130
515 PRINT"■"
520 FORI=0TO39:SP%=I
525 YW=192*(XP-Y(I))/DY:YW%=INT(YW)
530 LN%=INT(YW%/8):POKE216,LN%
535 CH%=(YW%/8-LN%)*8+1
540 POKE198,I:SYS826:PRINTMID$(PR$,CH%,1);:NEXT
545 WAIT158,1:GETQ$:IFQ$="■"THENGOSUB555
550 GOTO290
555 SP%=0:GOSUB605:JP%=4
560 WAIT158,1:GETQ$:GOSUB605
565 IFQ$="┣"ORQ$="┫"THENGOSUB710
570 IFQ$="▐"THENSP%=SP%+1:GOTO590
575 IFQ$="▌"THENSP%=SP%-1:GOTO590
580 IFQ$="="GOTO610
585 RETURN
590 IFSP%<0THENSP%=39
595 IFSP%>39THENSP%=0
600 GOSUB605:GOTO560
605 POKE198,SP%:SYS914:RETURN
610 YY=Y(SP%):XX=X(SP%)
615 PRINT"■":LN%=0:SP%=5:GOSUB125:PRINT"-CBM 3001- UNIPLOTT V5.0"
620 LN%=5:SP%=1:GOSUB125:PRINT"■ KOORDINATEN DES PUNKTES:"
625 LN%=8:GOSUB125:PRINT"X = ■";XX
630 LN%=10:GOSUB125:PRINT"Y = ■";YY
635 WAIT158,1:GETQ$:RETURN
640 IFSGN(XP)=SGN(XM)THEN515
645 Y0=INT(24*XP/DY):LN%=Y0:AX%=Y0
650 POKE216,Y0:SYS875
655 IFBI<0THENSP%=0:CH$="<"
660 IFBI>0THENSP%=39:CH$=">"
665 GOSUB120:PRINTCH$:SP%=(SP%OR39)ANDNOTSP%:GOSUB120:PRINT"X"
670 GOTO515
```

```
675 X0=39-INT(39*BI/DX):SP%=X0
680 IFXP<0THENLN%=24:CH$="V"
685 IFXP>0THENLN%=0:CH$="↑"
690 POKE198,SP%:SYS950:GOSUB120:LN%=(LN%OR24)ANDNOTLN%:GOSUB120:PRINT"X"
695 GOTO515
700 SYS934:ONJP%GOTO270,425,545,260
705 SYS634:ONJP%GOTO270,425,545,260
710 IFQ$="4"GOTO720
715 AF%=SP%:Q$="■":RETURN
720 EN%=SP%:FORKD=AF%TOEN%:POKE198,KD:SYS914:NEXT
725 VO=X(AF%):BI=X(EN%)
730 XP=-1E30:XM=1E30
735 DX=(BI-VO)/39:X0=VO
740 FORI=0TO39:X(I)=X0:Y(I)=FNY(X0)
745 IFY(I)>XPTHENXP=Y(I)
750 IFY(I)<XMTHENXM=Y(I)
755 X0=X0+DX:NEXT:Q$="P":RETURN
760 PRINT"□ SPIEGLUNG DER FUNKTION UM ..."
765 LN%=3:SP%=5:GOSUB125:PRINT"■X■ - ACHSE"
770 LN%=5:GOSUB125:PRINT"■Y■ - ACHSE"
775 LN%=7:GOSUB125:PRINT"■*■ - URSPRUNG"
780 LN%=11:GOSUB125:PRINT"EINGABE ■X■, ■Y■ ODER ■*■ ";
785 SYS990:GETQ$
790 IFQ$<>"X"ANDQ$<>"Y"ANDQ$<>"*"GOTO785
795 PRINT" >■";Q$;"■<"
800 IFQ$<>"X"GOTO830
805 XP=-1E30:XM=1E30
810 FORI=0TO39:Y(I)=-Y(I)
815 IFY(I)>XPTHENXP=Y(I)
820 IFY(I)<XMTHENXM=Y(I)
825 NEXT:GOTO880
830 IFQ$<>"Y"THEN845
835 II=39:FORI=0TO19:YY=Y(I):Y(I)=Y(II)
840 Y(II)=YY:II=II-1:NEXT:GOTO880
845 XP=-1E30:XM=1E30
850 FORI=0TO39:Y(I)=-Y(I)
855 IFY(I)>XPTHENXP=Y(I)
860 IFY(I)<XMTHENXM=Y(I)
865 NEXT
870 II=39:FORI=0TO19:YY=Y(I):Y(I)=Y(II)
875 Y(II)=YY:II=II-1:NEXT:GOTO880
880 Q$="P":RETURN
885 REM CBM - HARD - COPY
890 DATA169,4,133,212,169,111,133,211,32,186,240,165,211,32,40,241,169,13,32
895 DATA111,241,169,1,32,111,241,169,0,133,94,169,128,133,95,160,0,177,94,41
900 DATA127,201,32,48,9,201,64,48,8,24,105,128,208,3,24,105,64,32,111,241,20(
905 DATA192,40,48,228,169,13,32,111,241,169,1,32,111,241,24,165,94,105,40,13
910 DATA94,165,95,105,0,133,95,201,131,48,199,165,94,201,232,208,193,169,13
915 DATA32,111,241,76,131,241,0,-1
920 SD=634:PRINT"□ INIT -CBM 3001- HARD - COPY "
925 READCD%:IFCD%<0THEN935
930 POKESD,CD%::SD=SD+1: GOTO925
935 REM CBM - VIDEO - DRIVER 2
940 DATA169,0,162,128,164,216,240,9,24,105,40,144,1,232,136,208,247,133,196
945 DATA134,197,96,165,216,133,90,165,198,133,91,169,24,133,216,96,165,90,13
950 DATA216,165,91,133,198,16,211,169,32,208,7,169,64,208,3,173,255,3,41,127
```

```
955 DATA133,94,32,58,3,160,39,165,94,145,196,136,16,251,96,32,58,3,160,39,1
960 DATA196,73,128,145,196,136,16,247,96,32,80,3,32,58,3,164,198,177,196,73
965 DATA128,145,196,198,216,16,241,48,183,32,80,3,32,131,3,198,216,16,249,4
970 DATA171,169,32,208,7,169,93,208,3,173,255,3,41,127,133,94,32,80,3,32,58
975 DATA3,165,94,164,198,145,196,198,216,16,243,48,138,32,182,3,32,107,3,16
980 DATA219,76,210,255,162,16,165,143,164,158,208,17,197,143,240,248,202,20
985 DATA243,164,198,177,196,73,31,145,196,208,231,96,32,50,247,231,0,75,75,
990 SD=826:PRINT"□INIT CBM -3001- VIDEO - DRIVER 2"
995 READCD%:IFCD%<0THENRETURN
1000 POKESD,CD%:SD=SD+1:GOTO995
```

Es folgen die disassemblierten Maschinenprogramme für VIDEO2 (CBM) und für HARDCOPY, das den Bildschirminhalt auf eine vorwählbare IEC-Device ausgibt, hier Nr. 4, Drucker. Für sich ist es mit SYS 634 aufrufbar.

```
033A LDA #00
033C LDX #80
033E LDY D8
0340 BEQ 034B
0342 CLC
0343 ADC #28
0345 BCC 0348
0347 INX
0348 DEY
0349 BNE 0342
034B STA C4
034D STX C5
034F RTS
0350 LDA D8
0352 STA 5A
0354 LDA C6
0356 STA 5B
0358 LDA #18
035A STA D8
035C RTS
035D LDA 5A
035F STA D8
0361 LDA 5B
0363 STA C6
0365 BPL 033A
0367 LDA #20
0369 BNE 0372
036B LDA #40
036D BNE 0372
036F LDA 03FF
0372 AND #7F
0374 STA 5E
0376 JSR 033A
0379 LDY #27
037B LDA 5E
037D STA (C4),Y
037F DEY
0380 BPL 037D
0382 RTS
0383 JSR 033A
0386 LDY #27
0388 LDA (C4),Y
038A EOR #80
038C STA (C4),Y
038E DEY
038F BPL 0388
0391 RTS
0392 JSR 0350
0395 JSR 033A
0398 LDY C6
039A LDA (C4),Y
039C EOR #80
039E STA (C4),Y
03A0 DEC D8
03A2 BPL 0395
03A4 BMI 035D
03A6 JSR 0350
03A9 JSR 0383
03AC DEC D8
03AE BPL 03A9
03B0 BMI 035D
03B2 LDA #20
03B4 BNE 03BD
03B6 LDA #5D
03B8 BNE 03BD
03BA LDA 03FF
03BD AND #7F
03BF STA 5E
03C1 JSR 0350
03C4 JSR 033A
03C7 LDA 5E
03C9 LDY C6
03CB STA (C4),Y
03CD DEC D8
03CF BPL 03C4
03D1 BMI 035D
03D3 JSR 03B6
03D6 JSR 036B
03D9 LDA #DB
03DB JMP FFD2
03DE LDX #10
03E0 LDA 8F
03E2 LDY 9E
03E4 BNE 03F7
03E6 CMP 8F
03E8 BEQ 03E2
03EA DEX
03EB BNE 03E0
03ED LDY C6
03EF LDA (C4),Y
03F1 EOR #1F
03F3 STA (C4),Y
03F5 BNE 03DE
03F7 RTS
03F8 JSR F732
03FB ■■ ERR
03FC BRK
```

```
027A LDA #04
027C STA D4
027E LDA #6F
0280 STA D3
0282 JSR F0BA
0285 LDA D3
0287 JSR F128
028A LDA #0D
028C JSR F16F
028F LDA #00
0291 STA 5E
0293 LDA #80
0295 STA 5F
0297 LDY #00
0299 LDA (5E),Y
029B AND #7F
029D CMP #20
029F BMI 02AA
02A1 CMP #40
02A3 BMI 02AD
02A5 CLC
02A6 ADC #80
02A8 BNE 02AD
02AA CLC
02AB ADC #40
02AD JSR F16F
02B0 INY
02B1 CPY #28
02B3 BMI 0299
02B5 LDA #0D
02B7 JSR F16F
02BA CLC
02BB LDA 5E
02BD ADC #28
02BF STA 5E
02C1 LDA 5F
02C3 ADC #00
02C5 STA 5F
02C7 CMP #83
02C9 BMI 0297
02CB LDA 5E
02CD CMP #E8
02CF BNE 0297
02D1 LDA #0D
02D3 JSR F16F
02D6 JMP F183
02D9 BRK
02DA BRK
```

```
CBM - DISSASSEMBLER
BY UWE KORNNAGEL
PRINTOUT BY:
  ** CBM 3022 **
********************
```

Dipl.-Math., Dipl.-Wirtsch.-Math. Georg Huber, Ungarten 11, 5300 Bonn 3

INTERRUPT-GETRIEBENE CASSETTENEIN- UND -AUSGABE (1)

E: This article describes an interrupt driven I/O routine for cassette recorder which does not block the processor during I/O.

1. Microcomputer führen üblicherweise eine Schreib- oder Leseroutine für Cassettenspeicherung in einem Zuge bis zum Ende aus und geben anschließend die Kontrolle ans Hauptprogramm oder an den Monitor zurück. Diese E/A-Jobs sperren den Prozessor lange Sekunden und manchmal Minuten von anderen Aufgaben ab. Vor allem in der Prozessdatenverarbeitung können derartig lange 'Zwangspausen' selten hingenommen werden. Einige Beispiele:

1.1 Der RAM-Speicher hat sich mit Meßdaten gefüllt, sie werden auf Cassette abgelegt. Daten, die im Verlauf der Cassettenausgabe anfallen, müssen ebenfalls erfaßt werden und dürfen nicht verlorengehen.

1.2 Mehrere Peripheriegeräte einschließlich Cassettenspeicher sollen gleichzeitig arbeiten.

1.3 Mittels Prozessor-Interrupts in genau gleichen Zeitabständen mißt der Prozessor Zeitintervalle oder die Uhrzeit. Diese Zeitttakt-Interrupts darf der Prozessor während der Cassetten-E/A nicht 'überhören'.

1.4 Das Bandzählwerk des Cassettenrecorders schickt Impulse an den Prozessor, der durch Aufwärts- oder Abwärtszählen einer internen Zählvariablen die augenblickliche Bandposition ableitet. Während der Cassetten-E/A bewegt sich das Band, also muß der Prozessor auch während der E/A die Variable aktualisieren können.

2. Eine Interrupt-getriebene Cassettenein-/-ausgabe blockiert den Prozessor nicht.

2.1 Einen Ausweg über Hardware bieten Mehrprozessor-Systeme oder die Verlagerung der Intelligenz in die Terminals. Das beschriebene Verfahren löst die Schwierigkeit durch Software, genauer gesagt durch Timer-Interrupts [1].

2.2 Grob skizziert läuft die Ausgabe auf Cassette folgendermaßen ab:

Das Hauptprogramm lädt die Parameter für Datenbeginn, -umfang, Satzkennnummer und dergl. (vgl. Orientierungs- und Kopfteil, 3.5) in die dafür reservierten Speicherzellen und ruft dann die Subroutine AUSG auf.

> Sie setzt die Variable STATUS auf 'Ausgabe in Arbeit', stellt den Byte-Timer (vgl. 2.5) auf beispielsweise 10 msek und schaltet den Cassettenrecorder auf Aufnahme. Ende Subroutine.

10 Sekunden lang arbeitet das Hauptprogramm weiter. dann ist der Byte-Timer abgelaufen

> und verursacht einen Interrupt: Die Interruptroutine stellt den Byte-Timer wieder auf 10 msek und legt mittels der OUTBTH-Subroutine das erste Byte auf das Magnetband. Dies beansprucht ca. 6,7 msek. Ende Interrupt-Routine.

3,3 msek lang weiter im Hauptprogramm, dann

> Interrupt vom Byte-Timer: Byte-Timer auf 10 msek stellen, zweites Byte auf Band legen (6,7 msek), Ende Interrupt-Routine

1) Als Einführung in das Interrupt-Programmieren vgl. etwa Löhr, Roland: 'Interrupt-Demonstrationsprogramme für das VIA 6522', 65xx MICRO MAG, 10, 1979, S. 24-28

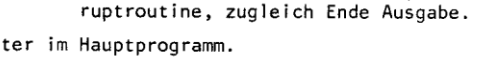
3,3 msek weiter im Hauptprogramm

... (dito)

Interrupt vom Byte-Timer: Letztes Byte auf Band legen (6,7 msek), Variable STATUS auf 'Ein-/Ausgabe fertig' setzen, Ende Interruptroutine, zugleich Ende Ausgabe.

Weiter im Hauptprogramm.

Das Cassettenband trägt schließlich folgende Magnetisierung:



wobei beim Einlesen die ersten Bytes der Synchronisierung dienen (vgl. Dateiaufbau, 3.5).

Am Wert Der STATUSvariablen kann der Prozessor zu jeder Zeit erkennen, ob die Ausgabe oder Eingabe aktiv oder (regulär oder fehlerhaft) abgeschlossen ist.

2.3 Bei der Eingabe kann der Prozessor die Daten erst dann erkennen, sobald er sowohl in Takt als auch in Phase mit den abgelegten Bytes liest. Die Taktfrequenz ist ihm bekannt (im Beipiel 10 msek), die Phase findet der mit Hilfe der Synchronisationsbytes:

Das Hauptprogramm lädt die Parameter für die Eingabe und ruft die Subroutine EING auf.

Sie setzt STATUS auf 'Eingabe in Arbeit', stellt den Byte-Timer auf 10 msek und startet die Cassettenrecorder-Wiedergabe. Ende Subroutine.

10 msek weiter im Hauptprogramm:

Der abgelaufene Byte-Timer meldet einen Interrupt an. Der Prozessor versucht nun, Eintragungen auf dem Magnetband zu finden, ggfs. darin ein Synchronisationsbyte zu erkennen. Gelingt ihm das nicht innerhalb von z.B. 8 msek (der Byte-Timer dient als Zeitmesser), wird der Byte-Timer auf 10 msek gestellt. Ende Interrupt-Routine.

10 msek später im Hauptprogramm:

Die Versuche wiederholen sich solange, bis gilt:

Interrupt vom Byte-Timer: Der Prozessor versucht ein Synchronisationsbyte zu erkennen und hat innerhalb von 8 msek Erfolg, Daraufhin setzt er den Byte-Timer auf 3,2 msek (knapp weniger als die 3,3 msek-Lücke zwischen dem Ende dieses und dem Beginn des folgenden Bytes). Ende Interrupt-Routine.

3,2 msek später im Hauptprogramm

Interrupt vom Byte-Timer: Byte-Beginn suchen (für ca. 0,1 msek), den Byte-Timer auf 9,9 msek stellen (also etwas kürzer als der 10 msek-Abstand von zwei Byte-Anfängen), das Byte lesen (dies dauert 6,7 msek), Ende Interrupt-Routine.

3,2 msek weiter im Hauptprogramm:

... Die Byte-Eingabe wiederholgt sich: Erst liest der Prozessor mehrmals Synchronisationsbytes ein, dann Dateikopf, Daten usw., schließlich

Interrupt vom Byte-Timer: Letztes Byte lesen, STATUS auf 'Ein-/Ausgabe fertig' setzen, Ende Interrupt-Routine, zugleich Ende Eingabe.

Weiter im Hauptprogramm.

2.4 Das Hauptprogramm darf den durch die Byte-Timer-Interrupts gesetzten Takt nicht dadurch stören, daß es diese Interrupts mit dem Befehl SEI (Set Interrupt Disable Status) sperrt. Umgekehrt sind Byte-E/A-Routinen regelmäßig zeitkritisch und brauchen einen Schutz vor sie überlagernden Interrupts (eben durch SEI); dies bedeutet, daß im beschriebenen Verfahren eine Interruptbearbeitung mitunter um längstens die Zeitspanne einer Byte-E/A wartet (im Beispiel ≤ 6,7 msek).

2.5 Das Interrupt-getriebene E/A-Verfahren funktioniert unabhängig davon, welche Subroutine zur Abspeicherung der einzelnen Bytes der Anwender wählt, er kann sie an seinen Anforderungen hinsichtlich Fehlerraten, Geschwindigkeit usw. ausrichten. Hat er sich auf eine Subroutine festgelegt, dann liegt auch der maximale Zeitbedarf für das Abspeichern eines Bytes (im Beispiel gleichbleibend 6,7 msek), die höchstzulässige Taktfrequenz des Byte-Timers (1/0.0067 sek = 150 Hz) und damit die größtmögliche Übertragungsrate (150 Bytes/sek) fest. Je langsamer der Anwender den Byte-Timer-Takt wählt (im Beispiel 10 msek, 100 Hz), um so längere Lücken läßt er zwischen den einzelnen Bytes (3,3 msek), in denen sich der Prozessor dem Hauptprogramm widmen kann, aber um so mehr drückt er auch die Übertragungsrate.

Erfahrungsgemäß erfordert die Interruptabarbeitung, vergleichen zur Byte-E/A, vernachlässigbar wenig Zeit, so daß der Anwender die Übertragungsrate bedenkenlos bis knapp unter ihr Maximum hochsetzen darf.

3. Eine programmierte Anwendung

3.1 Das folgende Programm zur Datenein- und -ausgabe auf Cassette arbeitet auf jedem 6502-System mit Cassetteninterface und Timer, wobei die Ausgabe einen (im folgenden 'Byte-Timer'genannt), die Eingabe einen zweiten (den 'Bit-Timer') braucht [2]. Es ist dem Betriebssystem eines Prozess-DV-Systems auf der Basis des SYM-1 entnommen. Nach Modifikation der SYM-spezifischen Adressen (sie sind im Programm im Kommentarteil durch SYM gekennzeichnet) läßt es sich auf andere 6502-Rechner übertragen.

3.2 Im Zusammenspiel mit einem fernsteuerbaren Cassettenrecorder [3], evtl. zusätzlich ausgestattet mit einem Zählwerk, das Impulse abschickt (vgl. Beispiel 1.4) oder das sich gar elektronisch abfragen läßt, kann man dem Rechner die Verwaltung und Handhabung seiner Banddateien überlassen: Bibliothek führen, unbeschriebene Bandlücken oder Dateien im Schnelllauf suchen, ein-/auslesen, bei Lesefehlern die Eingabe selbständig wiederholen und dergl..

3.3 Das Umschlagen ('transition') eines bestimmten Bits (im SYM-1 ist das Schreibbit A402/3. Bit) von 0 auf 1 und von 1 auf 0 überträgt das Cassetteninterface von diesem Bit auf das Magnetband. Die zeitlichen Abstände dieser Wechsel enthalten die zu schreibende Information: Ein Umschlag nach 731 Takten=Mikrosekunden steht für Bit=0, ein Umschlag nach 364 µsek und ein weiterer Umschlag nach 367 µsek steht für Bit=1. Dabei ist allein der Umschlag bedeutsam, seine Herkunftsrichtung (ob von High oder Low) spielt keine Rolle. Bitwert 0 und Bitwert 1 werden beide innerhalb von 731 µsek übertragen. Beispiel: Das Byte 16 (hex)=00010110 (binär) schreibt der Prozessor wie in Abb. 1 und 2 dargestellt. [4]

---

2) der 6532-Baustein enthält beispielsweise einen, der 6522-Baustein zwei Timer.

3) Zum Beipiel die Laufwerke der Mini-Digital-Cassettenrecorder von Philips 'zum Preis einer guten Pocket-Kamera' (lt. Werbung). Das Überbrücken von sieben Funktionstasten durch je einen Transistor und das Anzapfen der Zählwerksimpulse verwandelt das 'Stereo Tape Deck Universum System 8000' von

 (Fortsetzung nächste Seite)

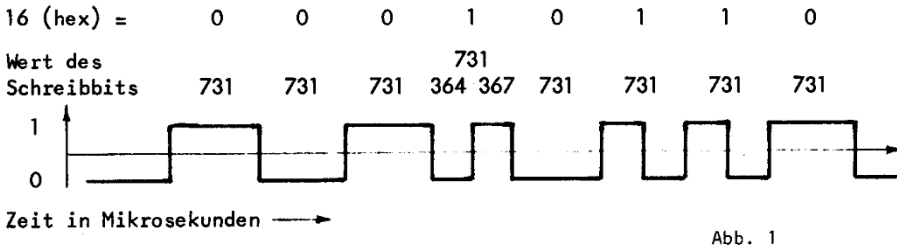


Abb. 1

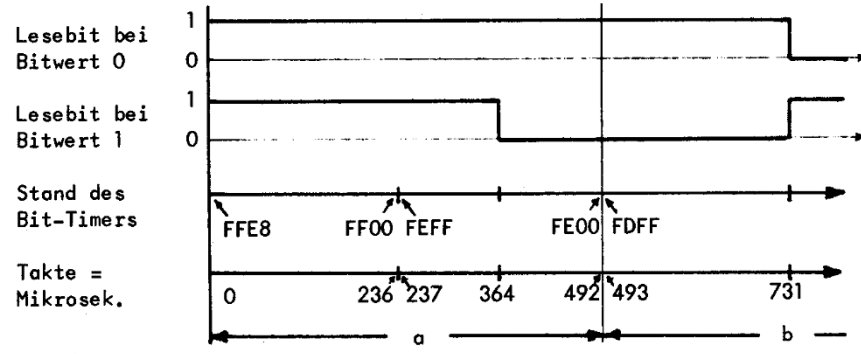


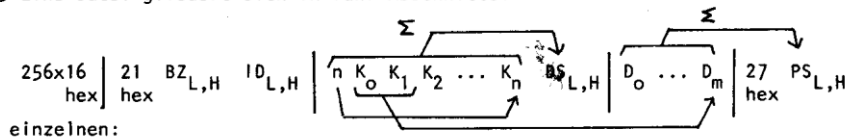
Abb. 2

Beim Lesen von Band lädt das Cassetteninterface den sich laufend ändernden Wert des auf das Band gelegten Schreibbits in das Lesebit (beim SYM-1 Bit 6 in Adresse A000), und der Prozessor filtert aus den zeitlichen Abständen zwischen den Wertwechseln das gesuchte Byte heraus: Das Byte beginnt, sobald das Lesebit erstmals umschlägt. Die Subroutine WCHSL stellt dann den Bit-Timer auf ungefähr 364 + 367/2 µsek. Hat das Lesebit nach dieser Zeitspanne den gleichen Wert wie zu Beginn der Zeitspanne, so stand auf dem Band der Bitwert 0, andernfalls der Bitwert 1. Mit dem nächsten Umschlag des Lesebits wird der Bit-Timer wieder gestellt usw.. Aus acht [5] aufeinanderfolgenden Bits liefert die Subroutine ZCHNLS das zu lesende Byte.

3.4 Vom Byte-Timer-Interrupt bis zur Rückkehr ins Hauptprogramm verstreichen etwa 6700 µsek. Demnach liegt die höchste Schreib-/Lesegeschwindigkeit bei 150 Bytes/sek. Mit einem beispielsweise 10000 µsek-Takt des Byte-Timers (=Variable BYTETT) - also einer Schreib-/Lesegeschwindigkeit von 100 Bytes /sek - bleiben für das Hauptprogramm in jeder Sekunde einhundert Arbeitslücken übrig. Aus der Adresse BYTETT lädt sich der Prozessor den Takt in Vielfachen von 100 (hex)=256 (dez) aus. Im Beispiel 27 (hex). 10000 (dez) ∿ 9984 (dez) = 256 (dez) x 39 (dez) = 100 (hex) x 27 (hex).

Fortsetzung zu 3): Quelle (mit Uhr und Zeitschalter DM 698,-) in eine vollautomatische Bandstation. Dessen ursprüngliche Tonaufnahme- und -wiedergabefähigkeit wird dadurch nicht beeinträchtigt. - 4) Dieselbe Information enthielte eine an der Zeitachse gespiegelte Kurve - 5) Die SYM-Routine OUTBTH arbeitet mit 9 Bit. In unserem Zusammenhang hat das keine Bedeutung. Bit 9 = konstant 0.

3.5 Eine Datei gliedert sich in fünf Abschnitte:



Im einzelnen:

a) Synchronisierung: 256 Synchronisationsbytes mit hex 16.

b) Orientierung: hex 21 = ASCII '!'
BZL, BZH: der augenblickliche Stand des Bandzählwerkes
IDL, IDH: Dateikenn-Nummer.

c) Kopf: K_0K_1=m : Anzahl der Datenbytes
K_2, ..., K_m: Zur freien Verfügung des Anwenders,
n: Länge-2 desjenigen Teils im Kopfabschnitt, den der Anwender festsetzt; n ≥ 1.
PSL, PSH: Prüfsumme aus den Bytes n, K_0, ..., K_n.

d) Daten: D_0, D_1, ..., D_m

e) Schluß: hex 25 = ASCII '/'
PSL, PSH: Prüfsumme aus den Datenbytes, ein Übertrag in ein drittes Prüfsummenbyte wird nicht vorgenommen.

Diese abschnittsweise Gliederung findet sich angenähert in der Ein- und Ausgaberoutine wieder.

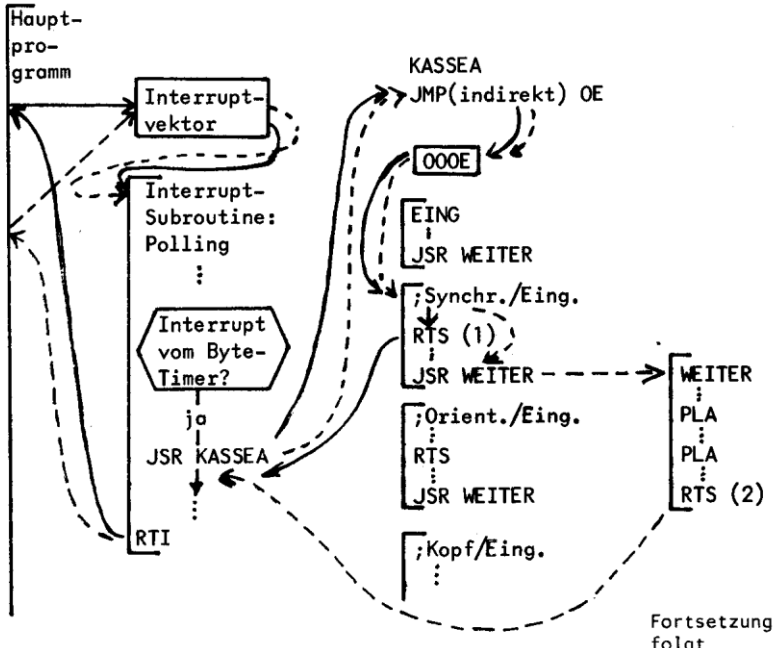


Abb. 3: Verlauf der Eingabe

GESCHACHTELTE INTERRUPTS

Heft 10 enthielt zwei Interrupt-Demonstrationsprogramme. Es war dort jeweils nur eine Interruptquelle (Timer bzw. Steuerleitung) zugelassen. Wie soll man sich nun verhalten, wenn sich mehrere Quellen unabhängig voneinander, konkurrierend zum Interrupt melden?

Es geht also nicht so sehr um das Interrupt-Polling, die Bestimmung der Interruptquelle (z.B. besprochen in Heft 5, Seite 43), sondern um das Wiederzulassen eines weiteren Interrupts, nachdem man sich schon in einer Interruptverarbeitung befindet. Erinnern wir uns an die Darstellungen im Programmierhandbuch: Mit dem Eintritt in einen Interrupt wird im Prozessorstatus automatisch das Interrupt-Flag gesetzt. Das ist gleichbedeutend, als wenn extra ein SEI (Set Interrupt Disable) programmiert worden wäre. Erst das RTI löscht das Interrupt-Flag, wiederum automatisch (den Befehl CLI - Clear Interrupt Disable - implizierend).

Während der Bearbeitung eines IRQ-Interrupts kann damit kein zweiter IRQ-Interrupt wirksam werden. Lediglich der mit Priorität versehene (nicht maskierbare) NMI-Interrupt unterbricht einen IRQ-Interrupt zwingend und sofort. Diese Priorität wird man in Anwendersystemen verwirklichen, der NMI hat dafür einen eigenen Pin an der CPU. In den Entwicklungssystemen sind die Steuerleitungen, Timer und Schieberegister der Interfacebausteine logisch ODER an den IRQ-Interrupt angeschlossen.

Nach dem Design unserer Interfacebausteine wird das IRQ-Signal solange auf Low gehalten, bis die Interruptquelle ihren speziellen Service erhalten hat. Service heißt hier: Der Anwender hat dem Interrupt das 'vermutlich Richtige' folgen lassen, er hat den zu einer Steuerleitung gehörenden Port gelesen oder neu beschrieben oder den verursachenden Timer ausgelesen oder mit einer neuen Vorgabe geladen. Erst damit geht die Interruptleitung wieder auf High. Wenn die Freigabe eines weiteren Interrupts mit CLI vor diesem Service erfolgt, so wird der Programmzähler sofort wieder mit dem Interruptvektor geladen, weil IRQ noch Low ist, also auch, wenn kein zweites Interrupt-Errignis eingetreten ist. Die fatalen Folgen sind absehbar: Der Stack wird immer tiefer umgegraben, bis es schließlich kein Zurückfinden mehr gibt.

Bei konkurrierenden Interrupts muß es also Philosophie sein, zunächst die zur Zeit bearbeitete Interruptquelle durch Service zu befriedigen und ihr IRQ-Signal abzustellen (LDA bzw. STA am Port, Timer), ehe man einen weiteren Interrupt mit CLI zuläßt. Die Abarbeitung des ersten Interrupts mag sich danach noch mit unempfindlichen Passagen fortsetzen. Tritt jetzt der konkurrierende Interrupt ein, so gelangt er bevorrechtigt zur Abarbeitung. Der Interrupthandler muß ihn entsprechend verzweigen.

Bei konkurrierenden Interrupts wird man die Prioritäten nach den betrieblichen Erfordernissen setzen. Ein anschauliches Beispiel ist eine von einem Timer angetriebene Echtzeituhr, die zwecks Vermeidung von Gangdifferenzen immer den schnellstmöglichen Service erhalten soll. Sie darf ein weniger zeitkritisches Interruptprogramm beliebig unterbrechen. Bei einer solchen Konstellation wird man sich also nicht damit zufrieden geben, daß die Interrupts im zeitlichen Nacheinander abgearbeitet werden.

Eine ähnliche Priorität wird man vorsehen, wenn z.B. von einem schnellen Datensender über eine Steuerleitung neue bereitstehende Daten gemeldet werden.

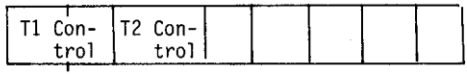
Unser Beispiel verwendet eine kleine Stoppuhr, die vom Timer 1 des Anwender-

VIA im AIM 65 angetrieben wird. Sie zeigt volle Sekunden < 60 seit Programmbeginn (Taste F3). Das Prinzip läßt sich beliebig verfeinern.

Für die Interrupte des Timers 2 wird nur ein Programmrumpf dargestellt, der mit beliebigen nützlichen Tätigkeiten ausgefüllt werden kann.

Im Vordergrund läuft eine ziemlich leere Warteschleife mit GETKEY. Wenn die ESCAPE-Taste betätigt wird, kommt die Stoppuhr zum Halten. Es gehört zu den Besonderheiten von GETKEY, daß die Routine erst nach einem Tastendruck verlassen wird. Im Vordergrund kann man sich also während ihres Ablaufes nicht mit der Abfrage weiterer Bedingungen im Prozessor befassen. Das wird erst möglich wenn man solche Routinen durch Interrupt z.B des Timers T2 regelmäßig aufbricht und dabei die notwendigen Prüfungen ausführt.

Die Stoppuhr (T1) ist auf 50 msek = 1 Jiffy eingerichtet. Die Interrupts des Timers T2 erfolgen nach den betrieblichen Erfordernissen in beliebig kürzeren oder längeren Zeiträumen. Die Funktionsweise der Timer (T1 = free running, T2 = one shot mode) wird durch initialisierendes Einschreiben in das ACR (Auxiliary Control Register) gesteuert:



Bit	7	6	5	4	3	2	1	0	
									Beispiele:
	0	1	x	x	x	x	x	x	T1 = free running
	x	x	0	x	x	x	x	x	T2 = one shot interval
									x = don't care

T1 wird bei diesem Modus nach jedem Nulldurchgang aus den Vorspeicherlatches automatisch nachgeladen, man muß lediglich sein Interrupt-Flag z.B. durch Lesen des Counters Low löschen.

T2 hat ein Vorspeicher-Latch nur für den Zähler Low. Ein neuer Durchgang wird durch Beschreiben des Counters High angestoßen. Dabei wird auch das T2-Interrupt-Flag gelöscht.

Der initialisierende Start der Timer erfolgt durch STA-Befehle in die jeweiligen Counter High.

Das IER (Interrupt Enable Register) hat folgende Besonderheit: Wenn das führende Bit (Nr. 7) beim Beschreiben '1' ist, so wird an den nachfolgenden mit '1' besetzten Bitpositionen Interrupt zugelassen. Ist das führende Bit '0', so wird an den nachfolgenden mit '1' besetzten Positionen die Interruptmöglichkeit abgestellt:

Bitposition	7	6	5	4	3	2	1	0
IER	Set/Clear	T1	T2	CB1	CB2	SR	CA1	CA2

```
Beispiele: LDA #$A0
           STA IER     INTERRUPT-ENABLE FÜR T2

           LDA #$20
           STA IER     INTERRUPT-DISABLE FÜR T2
```

Das Setzen der Flags im Interrupt Flag Register (IFR) bleibt unabhängig vom Interrrupt Enable und erfolgt durch die zugeordnete Signalquelle, wenn die im ACR oder PCR programmierte Bedingung eingetreten ist.

Bei der Auswahl von Steuerleitungen und Timern für die Programmierung von Interruptereignissen vergewissere man sich, daß man die korrekte Systembedienung an den Systeminterfaces nicht stört.

Bei der Entwicklung dieses Programmes stieß der Verfasser auf einen zeitraubenden Fehler im Rockwell-Datenblatt zum VIA 6522 (Doc. No. 29650 N 40 von Aug. 1977) Zur Steuerung des Timers 2 im ACR muß es richtig heißen: Wenn Bit5=1, dann Counting Mode, wenn Bit5=0, dann Interval Timer Mode. Im Hardware-Handbuch für den AIM 65 steht es bereits richtig.

```
--------------------
KONKURRIERENDE INTERRUPTS
--------------------
SYMBOLE
==0000 CR=$0D
==0000 TIM2=2
       FUER 512 CYCLES
==0000 ESC=$1B
==0000 T1CL=$A004
==0000 T1CH=$A005
==0000 T1LL=$A006
==0000 T2LL=$A008
==0000 T2CL=T2LL
==0000 T2CH=$A009
==0000 ACR=$A00B
==0000 IFR=$A00D
==0000 IER=$A00E
==0000 JIFFY=$EC
==0000 T1EN=$40
==0000 T2EN=$20
==0000 INTEN=$80
==0000 IRQV4=$A400
==0000 CURPO2=$A415
==0000 OUTDIS=$EF05
==0000 GETKEY=$EC40
==0000 KF3=$112
==0000 NUMA=$EA46
--------------------
ZEROPAGE
==0000
       *=0
==0000 JIFFYS
       *=*+1
==0001 SEK
       *=*+1
--------------------
INITIALISIERUNG
==0002
       *=KF3
==0112
4C000C JMP VORDER
```

```
--------------------
VORDERGRUNDPROGRAMM
==0115
       *=$C00
==0C00 VORDER
A942   LDA #<INTER      Interruptvektor
8D00A4 STA IRQV4        laden
A90C   LDA #>INTER
8D01A4 STA IRQV4+1
A90D   LDA #CR          Clear Display
2005EF JSR OUTDIS
A940   LDA #T1EN
==0C11
8D0BA0 STA ACR          T1 free running
A900   LDA #0
8501   STA SEK          Vorgabe   Sek.
8D08A0 STA T2LL
A912   LDA #18          Displayansteuerung
8D15A4 STA CURPO2
A501   LDA SEK
==0C22
2046EA JSR NUMA         Anzeige 00 Sek.
A902   LDA #TIM2        Laden T2
8D09A0 STA T2CH
A94E   LDA #$4E 50 MSEK
8D06A0 STA T1LL         Laden T1
A9C3   LDA #$C3
8D05A0 STA T1CH START T1
==0C34
A9E0   LDA #T1EN+T2EN+INTEN
8D0EA0 STA IER          Enable Interrupt
58     CLI
==0C3A WAIT
2040EC JSR GETKEY       Warteschleife
C91B   CMP #ESC         ESC ?
D0F9   BNE WAIT
00     BRK
```

++

KLEINANZEIGE

```
;-------------------
;INTERRUPTHANDLER
;
==0C42 INTER
48     PHA              Rette Akku
2C0DA0 BIT IFR          Abfrage der Interruptquelle im IFR
5028   BVC DOWHAT       Verzweige, wenn Interrupt nicht von T1 kommt
E600   INC JIFFYS       Interruptbearbeitung für T1: 20 x Hilfszähler
D01E   BNE OUT          Hochzählen, verzweigen, wenn 00 nicht erreicht
A9EC   LDA #JIFFY
8500   STA JIFFYS       Vorgabe nachladen
F8     SED              Dezimalmodus für Addition
18     CLC
==0C52
A501   LDA SEK          +1 auf den Sekundenzähler
6901   ADC #1
8501   STA SEK
C960   CMP #$60         60 Sekunden erreicht?
D004   BNE SHOW         nein, nur anzeigen
A900   LDA #0           60 Sek. = 00 Sek.
8501   STA SEK
==0C60 SHOW
A912   LDA #18          Anzeige ab 19. Stelle im Display
8D15A4 STA CURPO2
A501   LDA SEK
2046EA JSR NUMA         1 Byte = 2 Anzeigestellen in der aktiven Ausgabe-
==0C6A OUT              einheit
AD04A0 LDA T1CL         Lesen des Timers = Löschen des Interrupt-Flags
68     PLA
D8     CLD
40     RTI
;
;-------------------
;T2-HANDLING
;
==0C70 DOWHAT
A902   LDA #TIM2        Zurücksetzen des Interruptflags von T2 im
8D09A0 STA T2CH         Schnellgang. Lesen des Zählers Low
58     CLI              Freigabe anderer Interruptquellen
EA     NOP STELLVERTREND Hier kann eine Bearbeitungsroutine stehen
68     PLA
40     RTI
       END
 ERRORS= 0000
```

R. L.

****

FLOPPY DISK FÜR AIM 65: Die GWK-Elektronik, Herzogenrath, stellt auf der Hannover-Messe in Halle 12, 2. OG., Stand 21/22 ihr neues Mini-Floppy-Disk-System vor. 1-4 Laufwerke sind an AIM 65/PC100 anschließbar. Es handelt sich um softsektorierte Disketten mit je 98 kB. Zum Befehlssatz gehören SAVE und LOAD unter BASIC, Memory Dump (User), Initialisieren, Katalog, Formatabfrage, RENAME, SCRATCH, DELETE, RECOVERY of FILE, Packen, Kopieren, Testen, E/A in Editor und Assembler. Zu den weiteren Exponaten gehört die GWK-Gehäuseserie ebenso wie das neue Video-Interface.

Verschiedene Erweiterungskarten für AIM 65/PC100 zur Selbstbestückung bietet jetzt auch MCSD D. Kiesenberg an: Postfach 579, 46 Dortmund 1.

Michael Zimmermann, Eberstädter Str. 170, 6102 Pfungstadt

Anschluss von numerischer Anzeige und Tastatur an einen 6500-Mikrocomputer (2)

(Fortsetzung zu Abschnitt 4: Erzeugung von Leuchtmustern auf der Anzeige)

Das nachfolgende Programm 3 gibt gepackte Daten aus.

```
--------------------
PROGRAM 3
;GET CHARACTERS TO BE DISPLAYED

==0228 GET
       *=*
A003   LDY #$03
;SET COUNT FOR PACKED
A207   LDX #$07
;FOR DISPLAY BUFFER
==022C G10
       *=*
B110   LDA (PTR),Y
;LOAD PACKED CHARACTER
290F   AND #$0F
;SEPARATE LOW ORDER NIBBLE
204002 JSR HCONV
;AND CONVERT IT TO LED-PATTERN
B110   LDA (PTR),Y
;RELOAD CHARACTER
4A     LSR A
;AND SEPARATE HIGH ORDER NIBBLE
4A     LSR A
4A     LSR A
4A     LSR A
204002 JSR HCONV
;CONVERT IT
==023C
88     DEY
;DECREMENT PACKED COUNTER
10ED   BPL G10
;IF NOT DONE GO BACK
60     RTS DONE, RETURN
```

```
--------------------
CONVERT HEXADECIMAL VALUE

==0240 HCONV
       *=*
8E4F02 STX XSAV
;SAVE X FOR LATER USE
AA     TAX
;BRING ACCU TO INDEX
BD5002 LDA TAB,X
;AND LOAD PATTERN
AE4F02 LDX XSAV
;RELOAD X(DISPLAY POSITION)
9500   STA BUF,X
;AND STORE PATTERN AT POSITION
CA     DEX
;DECREMENT DISPLAY POSITION
60     RTS AND RETURN
==024F XSAV
00     .BYT $00
;LED-PATTERNS
==0250 TAB
C0     .BYT $C0,$F9,$A4,$B0,$99,$92,
           $82,$F8
;0-7
F9A4
B099
9282
F8
80     .BYT $80,$98,$88,$83,$C6,$A1
           $86,$8E
;8-F
9888
83C6
A186
8E
 ERRORS= 0000
```

5. *Teilweise decodierte Tastatur*

Bei der vollständig unkodierten Tastatur haben wir haben wir sogleich auf deren Schwachstelle hingewiesen. Wie wir diese beheben können und Ein-/Ausgabeleitungen für andere Zwecke frei bekommen, soll jetzt erörtert werden.

Bei der vollständig unkodierten Tastatur wurden die einzelnen Ziffern durch ein Schieberegister angesteuert. Diese Funktion soll hier durch einen

Zähler übernommen werden, dessen Ausgang zur Anstuerung der Ziffern decodiert wird. Da die Erzeugung der Anzeigemuster weiterhin durch Software erfolgt, soll hier von einer teilweise decodierten Anzeige gesprochen werden.

Für die Leuchtmuster wird weiterhin der A-Port benutzt, der Zähler ist in den unteren 3 Stellen des B-Port realisiert, diese Bits werden extern über einen 3-zu-8 Decoder gesendet. Hierdurch wird wiederum jeweils nur eine Ziffer angesteuert. Die oberen 5 Bit des B-Port stehen für andere Anwendungen zur Verfügung, z.B. für Eingaben. Die sich hieraus ergebenden Möglichkeiten sollen an anderer Stelle diskutiert werden.

Der generelle Aufbau der Hardware ist Abb. 6 zu entnehmen. Das Ansteuerprogramm unterscheidet sich nur in Initialisierung und Displayteil vom 1. Beispiel, deswegen sind in Programm 4 auch nur diese Teile dargestellt. Weitere Details können den Kommentaren entnommen werden. Im Prinzip ist es möglich, mit dieser Schaltung bei entsprechenden Decodern und Programmmodifikationen die Anzeige auf mehr Stellen, z.B. 16 zu erweitern, am Aufbau von Hard- und Software ändert sich dabei grundsätzlich nichts. Entsprechend unserer Vorgabe wollen wir uns aber mit 8 Stellen begnügen.

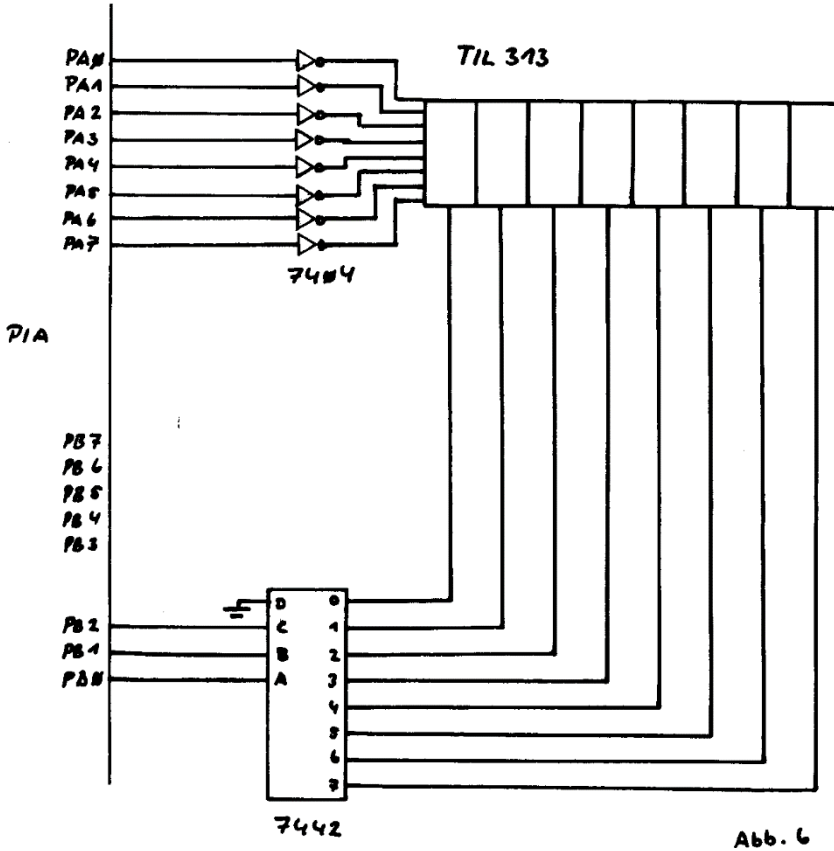


Abb. 6

```
--------------------
PROGRAMM 4, INITIALISIERE PIA

==020F INIT
       *=*
A900   LDA #$00
;SET CRA/CRB TO DDRA/DDRB
8D0190 STA CRA
8D0390 STA CRB
A9FF   LDA #$FF
;AND SET DIRECTION OUT
8D0090 STA DDRA
A907   LDA #$07
;USE ONLY LOW ORDER BITS
8D0290 STA DDRB
==0221
A904   LDA #$04
;NOW ADDRESS DATA REGISTER
8D0190 STA CRA
8D0390 STA CRB
60     RTS
       .OPT LIS
```

```
--------------------
DISPLAY BUFFER

==022B DISP
A207   LDX #$07
;SET POINTER TO BUFFER
==022D D20
B500   LDA BUF,X
:GET BUFFER CHARACTER
8D0090 STA DRA
;AND PUT INTO LEDS
8E0290 STX DRB
;ADDRESS DISPLAY
A0FF   LDY #$FF
;NOW DELAY
==0237 D30
       *=*
88 DEY
D0FD   BNE D30
A9FF   LDA #$FF
;SIGN OFF LEDS
8D0090 STA DRA
CA DEX
;NOW NEXT CHARACTER
10EB   BPL D20
;TO NEXT DISPLAY
60     RTS WE ARE THRU
```

6. Vollständig decodierte Anzeige

In vielen Fällen ist eine rein numerische Anzeige für die spezifischen Belange des Anwenders vollkommen ausreichend, und besondere Siebensegmentmuster, die über die numerischen Zeichen hinausgehen, sind in keinem Fall erforderlich. In diesen Fällen ist es möglich und es kann sinnvoll sein, die eigentliche Zeichenerzeugung durch einen entsprechenden Baustein, z.B. den 7448 vorzunehmen. Der Vorteil, den wir mit dieser verminderten Flexibilität im Zeichenvorrat erkaufen, besteht darin, daß lediglich 1 PIA-Port für eine 8-stellige Anzeige erforderlich ist.

Eine Möglichkeit für einen Anschluß ist aus Abb. 7 zu entnehmen; zu beachten ist, daß der 7448 ein geringes Fan-out hat, es sind deswegen evtl. Pull-up-Widerstände vorzusehen, um im Multiplexbetrieb ein hinreichend kontrastreiches Muster für jede Anzeige zu erhalten.

Die gesamte Anzeige wird über den B-Port bedient, die Software erfordert hierfür größere Eingriffe, deswegen ist das Programm 5 wieder vollständig abgedruckt. Eine Umwertung der Zeichen über Software ist hier nicht erforderlich, vielmehr sollten hier nur numerische gepackte Daten angezeigt werden, die durch eine GET-Routine entpackt im vorderen Halbbyte des Puffers abgelegt werden. Anschließend wird der Pufferinhalt bei der Ausgabe mit der Zeichenposition im niederen Halbbyte verknüpft und an den B-Port angelegt.

Die jetzt erforderliche Dekodierung, sowohl der Ziffernposition als auch des Anzeigewertes erfolgt durch externe Bauteile, die auf den Port geschaltet sind. Allein das Multiplexing erfolgt unter Steuerung der Software.

Mit diesen Formen der Anzeige haben wir mehrere Möglichkeiten kennengelernt, der Entwickler sollte damit in der Lage sein, eine seinen Anwendungen entsprechende Lösung zu finden.

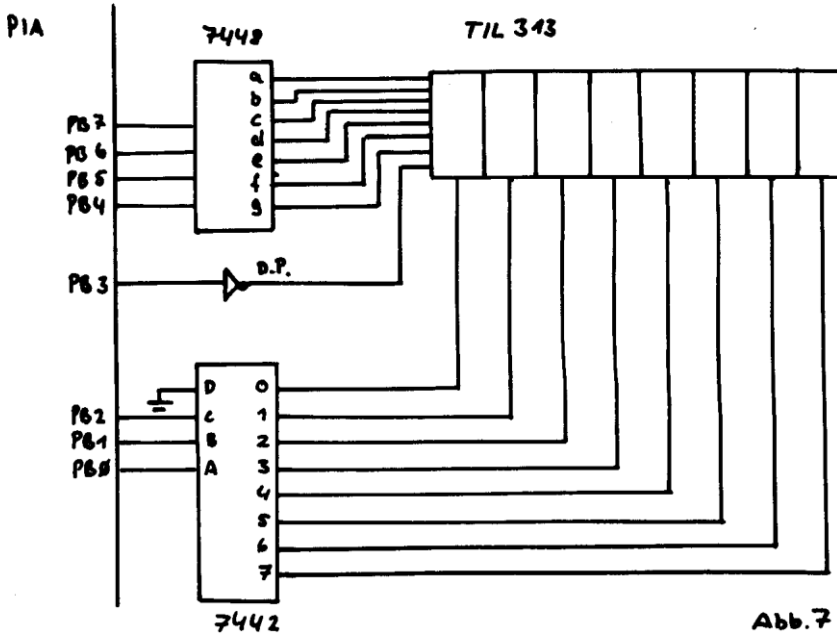


Abb.7

```
--------------------
INITIALIZE PIA  - PROGRAMM 5

==020F INIT
       *=*
A900   LDA #$00
;SET CRA/CRB TO DDRA/DDRB
8D0390 STA CRB
A9FF   LDA #$FF
;AND SET DIRECTION OUT
8D0290 STA DDRB
A904   LDA #$04
;NOW ADDRESS DATAREGISTER
8D0390 STA CRB
60     RTS

--------------------
GET CHARACTERS TO BE DISPLAYED

==021F GET
       *=*
;SET INITIAL VALUES FOR POINTERS
A207   LDX #$07
A003   LDY #$03
==0223 G10
       *=*
;GET PACKED VALUE
B110   LDA (PTR),Y
;AND USE LOW ORDER NIBBLE
0A     ASL A
0A     ASL A
0A     ASL A
0A     ASL A
;MASK OFF DECIMAL POINT
0908   ORA #$08
;AND STORE IT IN BUFFER
9500   STA BUF,X
CA     DEX
;GET AGAIN
B110   LDA (PTR),Y
;AND USE HIGH ORDER NIBBLE
29F0   SNF #$F0
0908   ORA #$08
==0234
9500   STA BUF,X
CA     DEX
88     DEY
10E9   BPL G10
60     RTS

--------------------
DISPLAY BUFFER

==023B DISP
       *=*
A207   LDX #$07
;SET POINTER TO BUFFER
==023D D20
8A     TXA
;GET POSITION
1500   ORA BUF,X
```

```
;AND CHARACTER
8D0290 STA DRB
;PUT INTO B-PORT
A0FF   LDY #$FF
;NOW DELAY
==0245 D3Ø
       *=*
88     DEY
D0FD   BNE D3Ø
A9FF   LDA #$FF
;SIGN OFF LEDS
8D0290 STA DRB
CA     DEX
10ED   BPL D2Ø
60     RTS
;IF WE ARE THRU RETURN
 ERRORS= 0000
```

7. Erste Versuche mit einer Tastatur

Die vorherigen Abschnitte waren von dem Bestreben geprägt, mehr und mehr Ein- und Ausgabeleitungen des Peripheriebausteines für andere Anwendungen frei zu bekommen. Jetzt wollen wir die Früchte dieser Bemühungen ernten und uns dem Anschluß von Tastaturen an diese Leitungen zuwenden. Weiterhin soll das Zusammenspiel zwischen Tastatur und Anzeige untersucht werden.

Der einfachste, gleichzeitig aber leitungsintensivste Lösung für einen Tastaturanschluß besteht darin, jede Ein-/Ausgabeleitung des PIA mit einem Schalter zu bestücken, der diesen Eingang in geschlossenem Zustand gegen Erde zieht. Neben dem PIA wären hier nur noch Pull-up-Widerstände erforderlich. Nachteilig ist jedoch, daß mit einem Port nur 8 Schalter bedient werden können. Für größere Tastaturen sind nach diesem Vorschlag also mehrere PIAs erforderlich. Daher soll diese Lösung nicht weiter betrachtet werden.

Eine derartige eindimensionale Tastatur kann jedoch mit einem einfachen 4-zu16 Decoder von einem einzigen PIA-Port bedient werden. Abb. 8 zeigt Einzelheiten des Anschlusses und Aufbaues, Programm 6 beinhaltet die Bedienung. In die unteren4 Bit des Ports werden nacheinander die 16 möglichen Zustände eingeschrieben und durch Abfrage von Bit 7 geprüft, ob der entsprechende Schalter geschlossen ist. Vorteilhaft ist hierbei, daß der Inhalt der unteren Portstellen als Binärwert der betätigten Taste interpretiert werden kann, eine weitere Transformation ist nicht erforderlich.

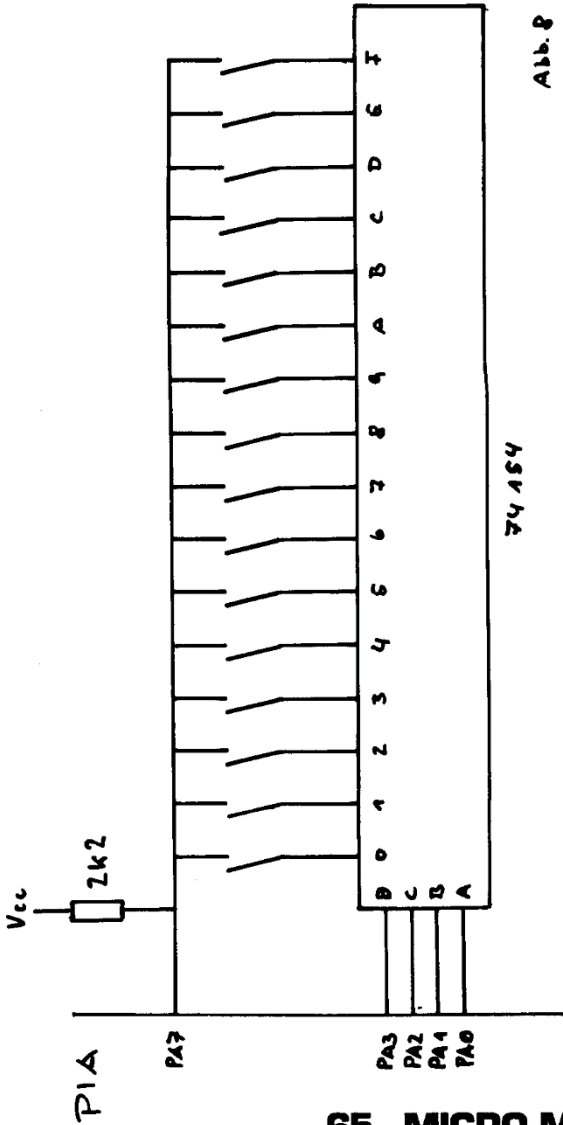
Wir betrachten das Bedienungsprogramm einmal näher, um hier einige grundsätzliche Prinzipien der Tastaturansteuerung zu erkennen. Die Abfrage der Tastatur erfolgt jeweils nach einem Displaydurchgang, es werden alle Schalter nacheinander abgeprüft. Wurde im Tastaturdurchlauf kein geschlossener Schalter gefunden, so wird der Status auf $FF gesetzt, wurde ein betätigter Schalter erkannt, so wird zuerst getestet, ob dieser Zustand bereits im letzten Durchlauf anlag. Ist dies der Fall, so unterbleiben weitere Aktionen. Wurde aber ein neuer Schalter betätigt, so wird der entsprechende Wert in den Status gestellt. Handelt es sich um einen numerischen Wert, so wird dieser zugleich in den Displaybuffer gespeichert. Funktionsschalter lösen - wie hier nur angedeutet - hingegen weitere Maßnahmen aus.

Sicher funktioniert eine derartige Tastatur sehr einfach, wenn nur ein einziger Schalter betätigt ist. Was aber, wenn bei schneller Eingabe mehrere Tasten gleichzeitig gedrückt werden? Der einfachste Ausweg aus diesem Dilemma ist zu postulieren, daß nicht sein kann, was nicht sein darf, und eine derartige Doppeltastung solange zu übergehen, bis nur noch ein Schalter gedrückt ist. Bei diesem, für eine schnelle Betätigung sehr wenig anwenderfreundlichen Verfahren, spricht man vom N-Key-Lockout.

Das entgegengesetzte Verfahren wird als N-Key-Rollover bezeichnet. Hierbei

werden unabhängig von ihrer Anzahl alle Tastenbetätigungen erkannt, entweder beim Drücken oder beim Freigeben des einzelnen Schalters. Dies bedeutet jedoch, daß jeder Taste eine Diode in Reihe geschaltet sein muß, um Ecktastungen auszuschalten, ein Umstand, der diesem sicher sehr anwenderfreundlichen Verfahren Grenzen setzt und hier nicht weiter betrachtet werden soll.

Wir wollen uns hier auf einen 2-Key-Rollover beschränken, bei dem zumindest 2 Tasten sicher getrennt werden. Im vorliegenden Programm 6 für eine eindimensionale Tastatur wird eine höhere Taste beim Betätigen, eine niedere beim Freigeben der gleichzeitig gedrückten niederen bzw. höheren Taste erkannt. Eine derartige Trennung dürfte in den meisten Fällen vollauf genügen.



Einzelheiten der Programmierung können den Kommentaren im Listing entnommen werden, hier wird das Zusammenspiel von linearer Tastatur und vollständig decodiertem Display dargestellt. - Noch nicht betrachtet wurde das Schalterprellen, hervorgerufen durch Auf- und Abschwingvorgänge in mechanischen Elementen eines jeden Schalters. Diese Effekte werden hier durch die Art der Tastaturansprache eliminiert, da die Schalter nur nach jedem vollständigen Displaydurchlauf, der ca. 1200 Zyklen benötigt, abgefragt werden, genug Zeit, um einen stabilen Zustand zu erreichen.

```
--------------------
PROGRAMM6 - ADVANCED
;DISPLAY DRIVER

==0000 DRA
       =$9000
;DATA REGISTER A
==0000 DDRA
       =DRA
;DATA DIRECTION REGISTER A
==0000 CRA
       =DRA+1
;CONTROL REGISTER A
```

```
==0000 DRB
       =DRA+2
;DATA REGISTER B
==0000DDRB
       =DRB
;DATA DIRECTION REGISTER B
==0000 CRB
       =DRB+1
==0000 BUF
       =$00 DISPLAYBUFFER
==0000 SHFT
       =$08
;POSITION AFTER BUFFER
==0000 LKEY
       =$09 LAST KEY
==0000 PTR
       =$10 POINTER TO DISPLAY
--------------------
MAIN PROGRAM
==0000
       *=$200
==0200
201802 JSR INIT
;INITIALIZE PIA
203302 JSR GET
;GET STORAGE TO DISPLAY
==0206 M1Ø
203402 JSR DISP
;DISPLAY BUFFER
204A02 JSR KEY
;CHECK FOR KEYBOARD ENTRY
90F8   BCC M1Ø
;HERE WE CAN PROCESS FUNCTION KEYS
E00F   CPX #$0F
D0F4   BNE M1Ø
207D02 JSR CLR
4C0602 JMP M1Ø
--------------------
INITIALIZE PIA

==0218 INIT
       *=*
A900   LDA #$00
;SET CRA/CRB TO DDRA/DDRB
8D0190 STA CRA
8D0390 STA CRB
A97F   LDA #$7F
;AND SET DIRECTION OUT
8D0090 STA DDRA
A9FF   LDA #$FF
8D0290 STA DDRB
==022A
A904   LDA #$04
;NOW ADDRESS DATAREGISTER
8D0190 STA CRA
8D0390 STA CRB
60     RTS
--------------------
GET CHARACTERS TO BE DISPLAYED
==0233 GET
       *=*
;SEE NEXT CHAPTER FOR DETAILS
60     RTS
--------------------
DISPLAY BUFFER

==0234 DISP
A207   LDX #$07
;SET POINTER TO BUFFER
==0236 D2Ø
8A     TXA
;GET POSITION
1500   ORA BUF,X
;AND OVERLAY WITH CHARACTER
8D0290 STA DRB
;AND OUTPUT
AOFF   LDY #$FF
;NOW DELAY
==023E D3Ø
88     DEY
D0FD   BNE D3Ø
A9FF   LDA #$FF
;SIGN OFF LEDS
8D0290 STA DRB
CA     DEX
;GO TO NEXT POSITION
10ED   BPL D2Ø
;TO NEXT DISPLAY
60     RTS
;RETURN WHEN TRU
--------------------
CHECK FOR KEYBOARD ENTRY

==024A KEY
A20F   LDX #$0F
:SET KEY COUNT
==024C K2Ø
8E0090 STX DRA
;AND OUTPUT IT
2C0090 BIT DRA
;IS INPUT
1008   BPL K3Ø
;PROCESS INPUT
CA     DEX
;DECREASE KEY COUNT
10F5   BPL K2Ø
;LOOK FOR NEXT KEY
8609   STX LKEY
;IF NO KEY, SET $FF TO LAST KEY
18     CLC
9020   BCC K99
==025C-K3Ø
E409   CPX LKEY
;CHECK WITH LAST KEY
18 CLC
```

```
F01B   BEQ K99
;INPUT DID NOT CHANGE
8609   STX LKEY
;DEPOSIT CHARACTER AT LAST KEY
E009   CPX #$09
B015   BCS K99
8A     TXA
;MOVE TO ACCU
0A     ASL A
;AND ADJUST TO LEFT
0A     ASL A
0A     ASL A
0A     ASL A
==026C
0908   ORA #$08 MASK OFF D.P.
8508   STA SHFT
;AND DEPOSIT IN POSITION BEHIND DISPLAY
A200   LDX #$00
;NOW SHIFT DISPLAY TO LEFT
==0272 K40
B501   LDA BUF+1,X
9500   STA BUF,X
E8 INX
E009   CPX #$09
;BUFFER AND SHFT
D0F7   BNE K40
18     CLC
==027C K99
60     RTS
---------------------
CLEAR DISPLAY
==027D CLR
       *=*
A207   LDX #$07
A9FF   LDA #$FF
==0281 CL10
       *=*
9500   STA BUF,X
CA     DEX
10FB   BPL CL10
60     RTS
 ERRORS= 0000
```

8. Tastaturanordnung

Bei Durchsprache der eindimensionalen Tastatur wurde noch nichts über die Anordnung der Tasten zueinander gesagt, sondern lediglich auf die hard- und softwaremäßigen Lagen zur Erkennung eines Schalterdruckes eingegangen. Da das Layout einer Tastatur von der Bedienung weitgehend unabhängig ist, soll dies hier nachgeholt werden und damit gleichzeitig eine Grundlage für weitere Tastaturformen gelegt werden. Dabei wollen wir unser Ziel nicht aus den Augen verlieren, nämlich Anzeige und Tastatur für ein numerisches Datenerfassungssystem.

Für die Erfassung numerischer Daten hat sich die Blocktastatur bewährt. Wie auch bei den Taschenrechnern sind 2 weitere Tasten in den Block integriert. ihnen sind meist die Lösch- und die Dezimalpunktfunktion zugeordnet. Zusätzlich zum numerischen Block und etwas abgesetzt von diesem sind Funktionstasten vorzusehen, mit denen gewünschte Verarbeitungen angestoßen oder Werte identifiziert werden können. Eine Einteilung dieser Schalter in Momenttasten, die nur einmalig im Moment ihrer Betätigung wirksam werden, und Rasttasten, die nach Betätigung dauernd aktiv bleiben, ist zweckmäßig. Beide Gruppen sollten voneinander abgesetzt sein, damit ihre Funktion jeweils klar erkennbar wird.

Eine Doppelbelegung von Tasten, wie sie bei Taschenrechnern üblich ist, muß hier wegen der damit verbundenen geringeren Eingabegeschwindigkeit abgelehnt werden. Ziel unserer Betrachtungen soll eine Anordnung nach Abb. 9 sein, die nach Erfahrung des Verfassers in den meisten Fällen vollauf genügen dürfte, alle Lösungen sollen hieran gemessen werden. Neben dem Numerik-Block mit Lösch- und Dezimaltaste sind 8 Momenttasten A-D und V-Y sowie 4 Rasttasten R-U vorgesehen, letztere am Rande des gesamten Feldes, um neben ihnen LEDs anzubringen, die den Tastenstatus angeben. Von diesem Ziel ist die eindimensionale Tastatur noch weit entfernt, da sie hier lediglich 4 eigentliche Funktionen erlaubt.

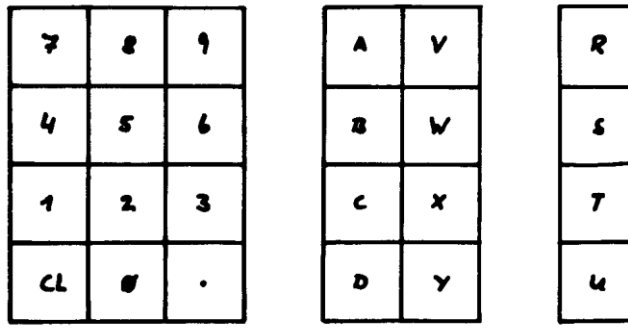


ABB. 9
TASTATUR-
ANORDNUNG

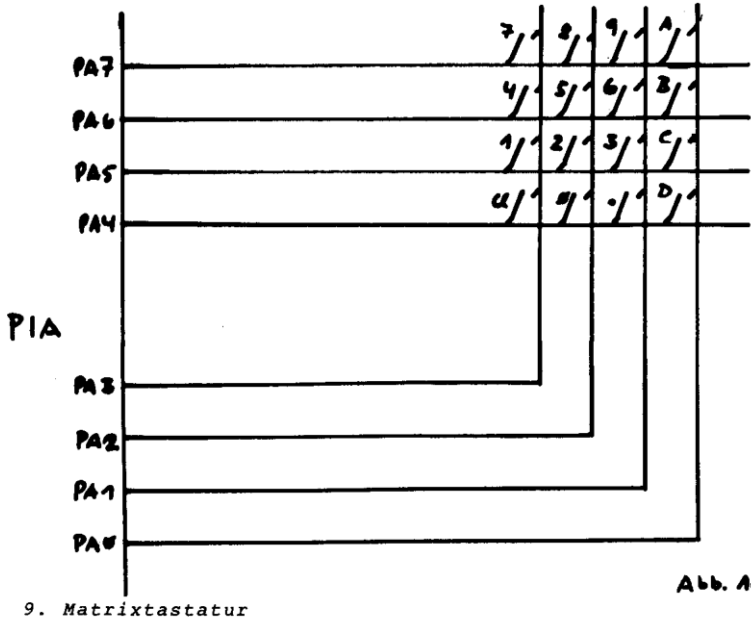


Abb. 10

9. Matrixtastatur

Bei der Besprechung der linearen Tastatur blieb noch ein Tatbestand unberücksichtigt: Von jedem Schalter muß eine separate Leiterbahn zumindest zum Decoder vorgesehen werden. Dies kann zu Problemen beim Layout der Tastaturplatine führen. Eine Abhilfe schafft hier eine Matrixanordnung, bei der jede Taste im Kreuzungspunkt von Zeile und Spalte einer Matrix angeordnet ist. Durch Anlagen von Null an die Spalten bei gleichzeitigem Hochziehen der Zeilen gegen die Versorgungsspannung kann ein geschlossener Schalter, der eine Verbindung zwischen einer Zeile und einer Spalte bewirkt, daran erkannt werden, daß eine oder mehrere Zeilen ebenfalls an Null liegen, da sie durch die gedrückte Taste auf dieses Niveau gezogen werden. - Durch Anlegen von Null an alle Spalten nacheinander und Testen der Zeileneingänge läßt sich die gedrückte Taste in der Matrix eindeutig lokalisieren, lediglich bei Mehrfachtastungen ergeben sich keine klaren Ergebnisse. Welcher Dezimalwert oder Steuercode der gedrückten Taste ent-

spricht, wird durch Vergleich mit einer Tabelle festgestellt, da jede Tastenposition ein eindeutiges, typisches Bitmuster hat. Wird das eingelesene Bitmuster nicht in der Tabelle gefunden, so liegt eine Mehrfachtastung vor.

Ein umgekehrtes Vorgehen, bei dem das Bitmuster als Index für den Wert benutzt wird, scheint nicht sinnvoll, da bei dieser Anschlußform für 16 Tasten 256 Bitmuster möglich sind. Eine derartige Transformation würde wegen der Größe der Tabelle ausscheiden. - Der Aufbau der Tastatur ist Abb. 10 zu entnehmen. Einzelheiten der Ansteuerung und die oben beschriebenen Schritte zur Tastenlokalisierung finden sich in Programm 7. Hier wurde wieder die vollständig codierte Anzeige zu Grunde gelegt.

```
--------------------
PROGRAMM 7: ADVANCED DISPLAYDRIVER

==0000 DRA
       =$9000
;DATA REGISTER A
==0000 DDRA
       =DRA
;DATA DIRECTION REGISTER A
==0000 CRA
       =DRA+1
==0000 DRB
       =DRA+2
==0000 DDRB=DRB
==0000 CRB
       =DRB+1
==0000 BUF
       =$00 DISPLAYBUFFER
==0000 SHFT
       =$08
;POSITION AFTER BUFFER
==0000 LKEY
       =$09
;LAST KEY
==0000 PTR
       =$10 POINTER TO DISPLAY
--------------------
MAI PROGRAM

==0000 *=$200
==0200
201802 JSR INIT
;INITIALIZE PIA
203302 JSR GET
;GET STORAGE TO DISPLAY
==0206 M1Ø
203402 JSR DISP
;DISPLAY BUFFER
204A02 JSR KEY
;CHECK FOR KEYBOARD ENTRY
90F8   BCC M1Ø
;HERE WE MAY PROCESS FUNCTION KEYS
C9F0   CMP #$F0
D0F4   BNE M1Ø
20BC02 JSR CLR
4C0602 JMP M1Ø
```

```
--------------------
INITIALIZE PIA

==0218 INIT
A900   LDA #$00
;SET CRA/CRB TO DDRA/DDRB
8D0190 STA CRA
8D0390 STA CRB
A90F   LDA #$0F
;AND SET DIRECTION OUT
8D0090 STA DDRA
A9FF   LDA #$FF
8D0290 STA DDRB
==022A
A904   LDA #$04
;NOW ADDRESS DATAREGISTER
8D0190 STA CRA
8D0390 STA CRB
60     RTS
--------------------
GET CHARACTERS TO BE DISPLAYED
==0233 GET
;SEE NEXT CHAPTER FOR DETAILS
60     RTS
--------------------
DISPLAY BUFFER

==0234 DISP
A207   LDX #$07
;SET POINTER TO BUFFER
==0236 D2Ø
8A     TXA
;GET POSITION
1500   ORA BUF,X
;AND OVERLAY WITH CHARACTER
8D0290 STA DRB
;AND OUTPUT
A0FF   LDY #$FF
;NOW DELAY
==023E D3Ø
88     DEY
D0FD   BNE D3Ø
A9FF   LDA #$FF
;SIGN OFF LEDS
8D0290 STA DRB
CA     DEX
```

```
;GO TO NEXT POSITION
10ED    BPL D2Ø
;TO NEXT DISPLAY
60      RTS WE ARE THRU
--------------------
CHECK FOR KEYBOARD ENTRY
==024A KEY
A200    LDX #$00
;SET OUTPUT LOW
8E0090 STX DRA
AD0090 LDA DRA
;IS ANY KEY CLOSED
C9F0    CMP #$F0
F040    BEQ K9Ø
;NO  KEY, RETURN
A907    LDA #$07
;SEARCH FOR CLOSED KEY
8D0090 STA DRA
==025B K2Ø
AD0090 LDA DRA
29F0    AND #$F0
C9F0    CMP #$F0
;IS KEY IN THIS COLUMN
D007    BNE K3Ø
;YES, PROCESS TABLE
4E0090 LSR DRA
;GO TO NEXT COLUMN
9002    BCC K3Ø
;ALL COLUMNS, RETURN
B0F0    BCS K2Ø
==026B K3Ø
AD0090 LDA DRA
;LOAD KEYPATTERN
A20F    LDX #$0F
;AND NUMBER OF KEYS
==0270 K4Ø
;SEARCH IN TABLE
DD9C02 CMP TABKEY,X
F005    BEQ K5Ø
;FOUND PATTERN, GET VALUE
CA      DEX
10F8    BPL K4Ø
;GO TO NEXT PATTERN
301C    BMI K9Ø
;FOUND NOTHING, RETURN
==027A K5Ø
;NOW GET VALUE
BDAC02 LDA TABVAL,X
C509    CMP LKEY
;AND COMPARE LAST KEY
F019    BEQ K95
;SAME, RETURN
8509    STA LKEY
;NOT THE SAME, DEPOSIT
C999    CMP #$99
;CHECK NUMERIC
B014    BCS K99
;FUNCTION KEY
8508    STA SHFT
;NOW SHIFT IN INTO DISPLAY
A200    LDX #$00
==028B K8Ø
B501    LDA BUF+1,X
9500    STA BUF,X
E8      INX
E009    CPX #$09
D0F7    BNE K8Ø
F004    BEQ K95

==0296 K9Ø
;NO KEY
A9FF    LDA #$FF
8509    STA LKEY
;RESET LKEY
==029A K95
;RETURN WITH CARRY CLEARED
18      CLC
==029B K99
;RETURN ANYWAY
60      RTS
--------------------
TABLE OF KEY-PATTERNS
==029C TABKEY
EE      .BYT $EE, $ED, $EB,$E7,
EDEB    $DE,$DD,$DB,$D7
E7DE
DDDB
D7
BE      .BYT $BE,$BD,$BB,$B7,
BDBB    $7E,$7D,$7B,$77
B77E
7D7B
77
==02AC TABVAL ;TABLE OF VALUES
F4      .BYT $F4, $E0,$00,$F0,
E000    $F3,$30,$20,$10
F0F3
3020
10
F2      .BYT $F2,$60,$50,$40,
6050    $F1,$90,$80,$70
40F1
9080
70
--------------------
CLEAR DISPLAY
==02BC CLR
A207    LDX #$07
A9FF    LDA #$FF
==02C0 CL1Ø
9500    STA BUF,X
CA      DEX
10FB    BPL CL1Ø
60      RTS
        .END
 ERRORS= 0000
```

WIRD FORTGESETZT

AIM Spezial (7)

Von den Lesern gingen wieder zahlreiche nützliche Vorschläge ein. Herr Holger Schröder, Fischergasse 10, 69 Heidelberg vereinfachte die Routinen RDRUB (E95F) und CUREAD (FE83) und ergänzte sie durch durch die Routine KLEIN zur Erkennung von Kleinbuchstaben:

Kleinschreibung für AIM 65

```
PHXY=$EB9E
PLXY=$EBAC
GETKEY=$EC40
OUTDIS=$EF05  ;EF02 WENN DILINK BENÖTIGT
PSLS=$E7DC
OUTDD1=$EF7B

*=$D00
RUB     JSR PSLS
RDRUB   JSR CUREAD
        CMP #$7F
        BNE ECHO
        DEY
        BPL RUB
        INY
        BEQ RDRUB
ECHO    CMP #$0D
        BEQ RETURN  <PHA ERSTE 20 CHAR.
        CMP #$5F    <PHA AUF DISPLAY
        BCC OUTD        IMMER GROSS
        AND #$5F
OUTD    JSR OUTDIS
        PLA
RETURN  RTS

CUREAD  TXA
        PHA
        LDX $A415
        CPX #$20
        BCS PAT2A
        LDA #$DE
        JSR OUTDD1
PAT2A   PLA
        TAX
        JSR PHXY
        JSR GETKEY
        JSR KLEIN
        JSR PLXY
        AND #$7F
        RTS

KLEIN   PHA
        TXA
        BNE GROS SHIFT ODER CRTL?
        PLA      JA: DANN RTS
        PHA
        AND #$40
        BEQ GROS ALPHA-CHAR.?
        PLA      NEIN, DANN RTS
        ORA #$20 WANDLE IN KLEIN-
        PHA      BUCHSTABEN
GROS    PLA
        RTS

        *=$108
        .WOR RDRUB
        ;USER INPUT FUNCTION
```

Das Bandformat des AIM 65 wurde schon einmal in Heft 10 angesprochen. Herr Michael Krägeloh, Schobertweg 51, 8520 Erlangen sandte:

Verschiebliches Laden im AIM-Format

Verschiebliches Laden ist leicht mit Hilfe einer User Function möglich. Dazu kann man die Routine ab E2E6 mit nur wenigen Veränderungen übernehmen (Programmausdruck s.u.). In der Originalroutine wird ab E2F7 bis einschl. E300 die Adresse vom Buffer geholt und nach A41C/A41D gebracht. Anstatt diese aus dem Buffer geholte Adresse einzuspeichern, habe ich die Routine "TO" (E7A7) ganz am Anfang in meinem Programm verwandt. Dadurch kann eine beliebige Startadresse definiert werden, ab der das zu lesende Programm abgelegt wird. Diese Adresse wird so natürlich nur einmal bestimmt und nicht nach jedem Block. Das macht nichts aus, solange man ein zusammenhängendes Stück lädt, da ADDR und ADDR+1 (A41C/1D) in STBYTE (E413) automatisch erhöht werden. Das ganze funktioniert nicht, wenn man unter einem Filenamen Daten von verschiedenen Speicherbereichen auf Band abgespeichert hat (Anm. d. Hrsg.: mit MORE?=Y und neuer Adresse). Unabhängig von eventuellen neuen Adressen, ab denen Daten eingelesen werden sollen, schreibt das modifizierte Programm alles einfach hintereinander ab der zu Anfang defi-

nierten Adresse. Mit MORE? =Y angehängte Dumps landen also schlicht in den folgenden Speicherzellen. - Das Programm rechnet nicht um, führt also kein RELOCATE aus.

Die Routine funktioniert nicht für das KIM-Format auf dem AIM, da für dieses eine eigene Adreßabfrage stattfindet.

```
VERSCHIEBLICH LADEN          *=0
0000 20 JSR E385  NCKERR     JSR CKERR
0003 20 JSR E7A7  BEGINN     JSR TO
0006 20 JSR E848  NLOAD      JSR WHEREI
0009 20 JSR E993  NLOAD1     JSR INALL
000C C9 CMP #3B              CMP #';
000E D0 BNE 0009             BNE NLOAD1
0010 20 JSR EB4D             JSR CLRCK
0013 20 JSR E54B             JSR CHEKAR
0016 AA TAX                  TAX
0017 20 JSR E54B             JSR CHEKAR
001A 20 JSR E54B             JSR CHEKAR
001D 8A TXA                  TXA
001E F0 BEQ 003B             BEQ NOLOAD4
0020 20 JSR E3FD  NOLOAD2    JSR RBYTE
0023 20 JSR E413             JSR STBYTE
0026 CA DEX                  DEX
0027 D0 BNE 0020             BNE NLOAD2
0029 20 JSR E3FD             JSR RBYTE
002C CD CMP A41F             CMP CKSUM+1
002F D0 BNE 0000             BNE NCKERR
0031 20 JSR E3FD             JSR RBYTE
0034 CD CMP A41E             CMP CKSUM
0037 D0 BNE 0000             BNE NCKERR
0039 F0 BEQ 0009             BEQ NLOAD1
003B 20 JSR E321  NLOAD4     JSR LOAD4
003E 4C JMP E182             JMP START

010C 4C JMP 0003  KEYF1      JMP BEGINN

START DES PROGRAMMES
MIT FUNKTIONSTASTE F1
```


SYS-Befehl im AIM-BASIC

Herrn Hermann Kneissel, Am Frauenberg 32 in 6430 Bad Hersfeld schafft folgenden Ersatz für den im AIM-BASIC fehlenden SYS-Befehl - direkter Einsprung in ein Maschinenprogramm mit dem Aufruf A=USR(Adresse):

```
LDA C6  POINTER RETTEN
PHA
LDA C7
PHA      C6,C7 AUF MOMENTAN
LDA 85  BEARBEITETE ZEILE SETZEN
STA C6
LDA 86
STA C7
JSR 00BF  SUCHE NACH USR-BEFEHL
CMP #B1
BNE F9
JSR 00BF  SUCHE DIE 1. ZIFFER DER
BCS FB    ADRESSE
JSR B7DA  UMRECHNEN IN HEXA
PLA       POINTER WIEDERHERSTELLEN
STA C7
PLA
STA C6
JMP (0014) SPRUNG INS PROGRAMM
```

Das Programm ab B7DA dient dazu, die BASIC-Zeilennummer zu errechnen. Es wandelt dazu die Ziffern, die auf C6/C7 zeigen, in eine Hexa-Zahl um. Falls die Zahl größer als $FFFF wird, erfolgt eine Fehlermeldung.

Christian Streicher, Wittelsbacherstr. 24, 8034 Germering

N U M B R (AIM)

E: This utility routine generates line numbers for the AIM text editor. The routine is called by typing IN=U. It will echo with FROM= to receive the starting line number. To leave the program and to continue with the standard editor commands two CRs must be typed.

Das Programm dient der Vorgabe von Zeilennummern im AIM-Text-Editor. Es wird bei der Initalisierung des Editors oder auf das Kommando 'R' (READ) mit IN=U gestartet. Anschließend fragt das Programm mit FROM nach der Startzeilennummer. Mit zwei aufeinanderfolgenden CR verläßt man den Zeilenvorgabemodus. Der Zeilenzähler steht nun hinter der letzten geschriebenen Zeile.

Die Zeilennummernvorgabe ist auf einen Abstand von 10 voreingestellt. Sie kann jedoch durch Änderung des in Zeile 1F74 stehenden LDX #1 (Dekade) und des in Zeile 1FCD stehenden ADC #1 (Schrittweite) beliebig eingestellt werden.

Generell kann das Programm auch zur Durchnumerierung beliebiger Texte verwendet werden. Es wäre also auch denkbar, ein Assembler-Listing mit Zeilennummern zu versehen. Hierzu muß lediglich der in Zeile 1FA3 stehende JSR RDRUB-Befehl geändert werden. Im Zusammenhang mit dem in Heft 9 veröffentlichten Programm TPASM von Herrn Steder wären hier noch einige interessante Möglichkeiten gegeben.

```
0000          NUMBR
0000          COPYRIGHT BY C. STREICHER, VERS. 1.2  3/80
0000                  *=0
0000 INPUT            *=*+6
0006 LASTX            *=*+1
0007 FLG              *=*+1
0008 ROTA             *=*+1
0009 SAVE             *=*+1
000A RDRUB            =$E95F
000A OUTPUT           =$E97A
000A CRLOW            =$EA13
000A HEX              =$EA7D
000A                  *=$108
0108          061F    .WOR ANF
010A                  *=$1F00
1F00 LASTN            *=*+6
1F06 ANF      8409    STY SAVE        SAVE CUR. POS.
1F08          A507    LDA FLG         HERE WE SEE WHAT IS TO BE DONE
1F0A          C988    CMP #$88
1F0C          D003    BNE *+5         QUIT SEQUENCE?
1F0F          4CC01F  JMP END3
1F11          C980    CMP #$80        NUMBER OUTPUT?
1F13          F068    BEQ OUT1
1F15          C940    CMP #$40        GENERATE A SPC?
1F17          D003    BNE *+5
1F19          4C961F  JMP OUT4
1F1C          C900    CMP #0          GENERATE A NUMBER?
1F1C          F04D    BEQ NUBR
1F20          C9FF    CMP #$FF        TEXT INPUT?
1F22          D003    BNE *+5
```

```
1F24          4CA31F  JMP OUT3
1F27          2013EA  JSR CRLOW       OR INITIALIZATION
1F2A          A200    LDX #0
1F2C          8E001F  STX LASTN       CLEAR LSD&MSD
1F2F          8E051F  STX LASTN+5
1F32 PRNT     BDEB1F  LDA TEXT,X      PRINT 'FROM='
1F35          207AE9  JSR OUTPUT
1F38          E8      INX
1F39          E006    CPX #6
1F3B          D0F5    BNE PRNT
1F3D          A000    LDY #0
1F3F INP      205FE9  JSR RDRUB       GET FIRST LINE NUMBER
1F42          207DEA  JSR HEX
1F45          B005    BCS RET         IS IT A 'CR'?
1F47          20E11F  JSR STORE       SAVE NUMBER
1F4A          D0F3    BNE INP         AND GET NEXT NUMBER
1F4C RET      88      DEY             A200   LDX #0
1F4F SORT     B500    LDA INPUT,X     NOW SORT IT
1F51          99001F  STA LASTN,Y     LSD FIRST
1F54          E8      INX
1F55          88      DEY
1F56          10F7    BPL SORT
1F58 FILL     E006    CPX #6          AND FILL UP THE REST
1F5A          F009    BEQ OK          WITH ZEROS
1F5C          A900    LDA #0
1F5E          9D001F  STA LASTN,X
1F61          E8      INX
1F62          4C581F  JMP FILL
1F65 OK       CE011F  DEC LASTN+1     SET NUMBER MINUS 10
1F68          A900    LDA #0          (ONLY FOR THE FIRST CALL)
1F6A          8507    STA FLG
1F6C          60      RTS
1F6D NUBR     A280    LDX #$80        GENERATE NEXT NUMBER
1F6F          8607    STX FLG
1F71          2013EA  JSR CRLOW
1F74          A201    LDX #1
1F76          20C91F  JSR COMP
1F79          A205    LDX #5
1F7B          8606    STX LASTX
1F7D OUT1     A606    LDX LASTX       OUTPUT 1 DIGIT
1F7F          CA      DEX
1F80          BD001F  LDA LASTN,X     CONVERT IT TO ASCII
1F83          18      CLC
1F84          6930    ADC #$30
1F86          207AE9  JSR OUTPUT
1F89          E000    CPX #0
1F8B          D004    BNE OUT2
1F8D          A040    LDY #$40        GENERATE A 'SPC' AFTER
1F8F          8407    STY FLG         LAST NUMBER
1F91 OUT2     8606    STX LASTX
1F93          A409    LDY SAVE
1F95          60      RTS
1F96 OUT4     A205    LDX #5          OUTPUT OF ONE 'SPC'
1F98          BDEB1F  LDA TEXT,X
1F9B          207AE9  JSR OUTPUT
1F9E          A2FF    LDX #$FF
1FA0          8607    STX FLG
1FA2          60      RTS
```

```
1FA3 OUT3   205FE9  JSR RDRUB       GET A CHAR. FROM KEYBOARD
1FA6        C90D    CMP #$0D        IS IT A 'CR'?
1FA8        D008    BNE END
1FAA        C508    CMP ROTA        IS IT THE FIRST 'CR'?
1FAC        F007    BEQ END2
1FAE        A200    LDX #0          YES, SO BE READY FOR
1FB0        8607    STX FLG         THE NEXT NUMBER
1FB2 END    8508    STA ROTA
1FB4        60      RTS
1FB5 END2   A988    LDA #$88        NEXT TIME GO BACK TO EDITOR
1FB7        8507    STA FLG
1FB9        8508    STA ROTA
1FBB        A409    LDY SAVE
1FBD END4   A90D    LDA #$0D
1FBF        60      RTS
1FC0 END3   A977    LDA #$77        IT WAS ONLY ONE 'CR'
1FC2        8507    STA FLG         SO IGNORE IT
1FC4        A000    LDY $0
1FC6        4CBD1F  JMP END4
1FC9        --- SUBROUTINE TO GENERATE NEXT NUMBER ---
1FC9 COMP   BD001F  LDA LASTN,X
1FCC        18      CLC
1FCD        6901    ADC #1
1FCF        C90A    CMP #$A
1FD1        900A    BCC LOW
1FD3        A900    LDA #0
1FD5        20DD1F  JSR LOW
1FD8        E8      INX
1FD9        20C91F  JSR COMP
1FDC        60      RTS
1FDD LOW    9D001F  STA LASTN,X
1FE0        60      RTS
1FE1        SUBROUTINE TO STORE STARTNUMBER
1FE1 STORE  8408    STY ROTA
1FE3        A608    LDX ROTA
1FE5        9500    STA INPUT,X
1FE7        C8      INY
1FE8        C006    CPY #6
1FEA        60      RTS
1FEB TEXT   4652    .BYT 'FROM= '
1FF1                .END

            SYMBOLTABLE
            INPUT   0000            FILL    1F58
            LASTX   0006            OK      1F65
            FLG     0007            NUBR    1F6D
            ROTA    0008            OUT1    1F7D
            SAVE    0009            OUT2    1F91
            RDRUB   E95F            OUT4    1F96
            OUTPUT  E97A            OUT3    1FA3
            CRLOW   EA13            END     1FB2
            HEX     EA7D            END2    1FB5
            LASTN   1F00            END4    1FBD
            ANF     1F06            END3    1FC0
            PRNT    1F32            COMP    1FC9
            INP     1F3F            LOW     1FDD
            RET     1F4C            STORE   1FE1
            SORT    1F4F            TEXT    1FEB
```

SOFTWARE FÜR PET & CBM

INGA

Sind Sie es leid immer gegen Ihren PET zu gewinnen? Dann ist INGA richtig für Sie. INGA steht für INtelligent GAme. INGA ist nicht nur ein weiteres Spielprogramm. INGA ist ein 16k Machinenprogramm, das nach neusten Erkenntnissen der künstlichen Intelligenz entwickelt wurde. Gespielt wird nach den Regeln von REVERSI. Schon auf Intelligenzstufe 1 bei Antwortzeiten von 2 Sekunden denkt INGA 5-11 Züge voraus und ist kaum zu schlagen. Hervorragende Graphik, beliebiges Eingeben/Ändern der Brettposition, Seitenwechsel. Disk/Cassette: DM 78.-

TYPIST

EDITOR Jeder Drucker ist anders! Auf Anforderung erhalten Sie unverbindlich einen Fragebogen zugeschickt, der es uns ermöglicht die richtige Version von TYPIST für Sie auszusuchen. Spezielle Version für Kugelkopf vorhanden. TYPIST ist einfach zu benutzen und doch vielseitig. Auch dieser Text wurde mit TYPIST editiert. Einfaches Einfügen, Löschen, Mischen von Texten mit Cassette oder Diskette, automatischer Randausgleich links und rechts. Preis je nach Version: DM 100.- bis DM 200.-

MATRA

TRACE MAschinensprachen TRAce. MATRA zeigt die Befehle, Operanden, Register, Flags und Adressen des laufenden Programms. Die Ablaufgeschwindigkeit läßt sich einstellen. Das sollte Standart sein für ein Trace-Programm. Das besondere an MATRA ist, daß es sich nicht durch Interrupts oder Stack-Operationen aus der Ruhe bringen läßt und kein Byte in der Zero-Page belegt. Auch der Ablauf von BASIC kann in Maschinensprache beobachtet werden. Wird für mehrere Speicherbereiche geliefert. Cassette: DM 40.-

FALA

COMPILER VORANKÜNDIGUNG !

FALA steht ur FAst LAnguge. FALA ist 100 - 300 MAL SCHNELLER ALS BASIC

Es handelt sich um eine Teilmenge von BASIC, erweitert um Elemente aus PASCAL und PL/1. Einfacher Umstieg von BASIC. FALA-Programme lassen sich mit einem Binder aus einer Bibliothek zusammenstellen. Ausser einem Editor gehört zum Lieferumfang ein 3-Adress-Macro-Assembler mit Operationen für beliebig viele Bytes (auch einzeln lieferbar) . Für alle 2001/3001 mit 32k & Disk (auch COMPU/THINK). ca. DM 800.-

BITTE STETS GERÄTETYP ANGEBEN PET/CBM UND DISKETTE ODER CASSETTE

ALLE PREISE SIND ENDPREISE INCL. VERSAND PER NACHNAHME !

SOFTWARE Wöst

MAX-PLANCK-STR 41A
7515 LINKENHEIM
TEL.: 07247/5031

MMC · Mertes Microcomputer Systeme GmbH

bietet an:

AIM 65 *KIM-1* SYM

Intelligente FLOPPY DISK

Hardware :
- Mutterkarte, anschlußfertig an AIM 65 mit 4 Eurobus-Steckplätzen, Adress- und Datenpufferung RAM- und EPROM-Steckplätzen
- 48 K (bzw. 32 K bei Teilbest.) quasi-statische RAM-Europakarte mit wahlfreier Adressbereichsbelegung
- Floppy-disc-Controller für 4 Laufwerke
- BASF 6106 Minidisc-Laufwerke
- Netzgeräte zum Betrieb der Erweiterungen einschließlich Disc-Drives

Software :
Floppy-Betriebssystem FDOS III mit:
- dynamischer Platzverwaltung
- sequentiellen und random Dateien
- variabler Satzlänge
- eigenem Befehlsprozessor, daher konfliktfreie Zusammenarbeit mit BASIC, Assembler, Editor und Monitor
- Dateibefehle direkt in BASIC programmierbar
- Disketten-Initialisierung gemäß IBM 3740

Preise:

1.	Komplettsystem mit 2 Laufwerken im 19" Gehäuse, betriebsbereit für AIM 65/PC 100 incl. 32 K RAM-Erweiterung	DM 4.750,00
	Einzelmodule:	
2.1	Mutterkarte	DM 395,00
2.2	Controllerkarte	DM 765,00
2.3	48 K-RAM	DM 1.385,00
2.4	32 K-RAM	DM 1.195,00
2.5	16 K-RAM, stat.	DM 880,00
2.6	FDOS III auf Diskette	DM 275,00
2.7	BASF 6106, Minifloppy ab 10 Stück	DM 680,00 DM 640,00
2.8	Netzgerät 5 V 2A, 12V 3A, 5V,0,5A	DM 245,00
3.	L 8810-2 6502-CPU mit V24-Schnittstelle 1 K stat. RAM, 4 x 2708, 6522, 6850, DMA-fähig, voll gebuffert	DM 498,00

MMC Mertes Microcomputer Systeme GmbH

Telefon 02233/68028

5030 Hürth-Efferen
Hahnenstraße 16

GWK
GESELLSCHAFT
FÜR TECHNISCHE ELEKTRONIK mbH.

D 5120 Herzogenrath
Asternstr. 2
Tel.: 02406/62394

NEU! NEU! NEU! NEU! NEU!

PC 100, Microcomputer von Siemens

Komplettgerät im Gehäuse.Ausgestattet mit 4K Byte RAM,
BASIC Interpreter und Netzteil.Garantiezeit beträgt 6 Monate.

1.990,--DM +MW

VIDEOMONITOR, Sanyo DM5912CX

Datenmonitor mit grünleuchtendem 12" Bildschirm.Stellt 24 Zeilen
mit je 80 Zeichen dar.Bandbreite 18 MHz.Professionelle Qualität.

798,--DM +MW

MATRIXDRUCKER ,TX80

Groß und Kleinschreibung,95 ASCII Charakter plus 64 Grafik
Charakter.Normalschrift und Breitschrift.2 Zeilenabstände
wählbar.Verstellbare Tractorführung für Papierbreiten bis 254mm.
80 Zeichen pro Zeile,150 Zeichen pro Sekunde.
Schnittstelle:8 BIT Parallel plus Handshake Control.
Lieferung inclusive SOFTWARE zur Ansteuerung über den
USER Output Handler bei Anschluss an User VIA von AIM 65 und
PC 100.

1.948,--DM +MW

Alle Preise zuzüglich MWST und Versandspesen.

GWK EURO BOARD EXPANSION SYSTEM

Microcomputer AIM 65,1 K Byte RAM		875,--	988,75 DM
Microcomputer AIM 65,4 K Byte RAM		1o5o,--	1186,5o DM
Resident ASSEMBLER,4K,in ROM		23o,--	259,9o DM
BASIC,Interpreter,8K,in ROM's		28o,--	316,4o DM
BASIC ERWEITERUNG,2K,auf Cassette		145,--	163,85 DM
BASIC ERWEITERUNG,2K,in EPROM,steckbar in Z 24		28o,--	316,4o DM
BASIC LISTING,ca 70 Seiten DIN A4,gut kommentiert		45,--	5o,85 DM
AIM 65 ANWENDERHANDBUCH,deutsch		35,--	39,55 DM
AIM 65 MANUALS,englisch	je	15,--	16,95 DM
AIM 65 SCHALTPLANPOSTER		5,--	5,65 DM
THERMOPAPIER,approved,schwarzdruckend,lagerfähig,50m		6,5o	7,35 DM
STECKERLEISTEN,44 Pole,nach MIL-C-21 097		12,5o	14,13 DM
MOTHER BOARD,12 Steckplätze f.Expansion,3 f.Appl. direkt an AIM steckbar,oder Einbau in 19" Gehäuse.Voll gepuffert.		485,--	548,o5 DM
ADAPTER BOARD,Busplatine ohne Pufferung,9 Steckplätze		295,--	333,35 DM
RAM BOARD, 3 x 4 K Byte statisches RAM,unabhängig voneinander beliebig adressierbar,write protectable.		945,--	1o67,85 DM
EPROM BOARD,12 K Byte,für die 1K EPROM's 2708 oder 2758.		395,--	446,35 DM
EPROM PROGRAMMER BOARD,mit res.Betriebsprogramm,Softwareumschaltung für 1K,2K und 4K Eprom's mit einer Versorgungsspannung.		795,--	898,35 DM
VIA/PIA BOARD,32 I/O Kanäle,beliebig adressierbar.		335,--	378,55 DM
PROTOTYP BOARD, Experimentierplatine zum Wrappen oder Fädeln.Durchdachte Aufteilung,über 1800 Rasterbohrungen.		75,--	84,75 DM
PET INTERFACE,ermöglicht den Anschluss der GWK BOARD's an den PET.Pufferung,Rückgewinnung der oberen Adressbits.		425,--	48o,25 DM
POWER SUPPLY BOARD,hochwertiges Schaltnetzteil incl.Trafo.			
5V/3A;26V/o,7A		315,--	355,95 DM
5V/3A,Notstromversorgung;26V/o,7A;+/- 12V/o,7A		45o,--	5o8,5o DM
5V/6A,Notstromversorgung;26V/o,7A;+/- 12V/o,7A		595,--	672,35 DM
GEHÄUSE,für AIM,Kunststoff		147,--	166,11 DM
GEHÄUSE,für AIM,Metall mit Platz für Netzteil u.kl.Erweiter.		345,--	389,85 DM
PULTGEHÄUSE,für AIM und System,Metall,mit 19" Baugruppenträger.		595,--	672,35 DM
Diverse LSI Chips:65oo Famile,EPROM's,RAM's,usw		bitte anfragen.	
Alle Preise zuzüglich Versandspesen.		zzgl MWST	incl.MWST

GWK GESELLSCHAFT FÜR TECHNISCHE ELEKTRONIK mbH.

D 5120 Herzogenrath
Asternstr. 2
Tel.: 02406/62394

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

HERAUSGEBER:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
2070 AHRENSBURG
☎ (04102) 55 816

BEZUGSBEDINGUNGEN: Abonnement ab laufender Ausgabe für 6 Hefte DM 45,- (Inlandsendpreis). Ausland/Foreign via surface mail DM 50,-, USA air $30. Die früher erschienenen Ausgaben dieser Zeitschrift sind nachlieferbar. Einzelpreis für Nos. 1-6 DM 7,-/St., Nos. 7-10 DM 7.80/St. Ein Sammelband für die früheren Ausgaben ist vorgesehen. Richten Sie bitte Ihre Überweisungen/Schecks an Roland Löhr, Konto 20/01121 bei der Vereins- und Westbank in Ahrensburg, BLZ 200 300 00. Zuschreibung auch über das Postscheckkonto der Vereinsbank: PSchA Hamburg 2244-207.

SERVICE AIM 65 - PC 100

Anwenderhandbuch für AIM 65 in deutscher Sprache. Original Rockwell-Handbuch mit zahlreichen Verbesserungen gegenüber der englischen Vorlage. Ab Lager lieferbar. Endpreis DM 32,10

Thermopapier für AIM 65/PC 100 in kontrastreicher Spitzenqualität. Qualitätsfreigabe durch Rockwell, Kalifornien. Packung mit 8 Großrollen, zus. 520 m, 57 mm breit, preiswert. Packung DM 50,85

Vorstehende Preise gelten für Vorkasse. Nachnahme + DM 1,50. Bezug beim Herausgeber.

BUCHANGEBOT

Camp, Smay, Triska: Microprocessor Systems Engineering. Lehrbuch für 65xx und speziell für AIM 65, besprochen auf Seite 11 in diesem Heft. DM 64,- + Versandkostenanteil DM 2,-.

PROGRAMMIERWORKSHOPS 65XX

9. und 10. Mai 1980 (Fr/Sa): Fortgeschrittenen-Workshop in Friedrichsdorf bei Frankfurt. Programmierung in Assembler-Sprache am Gerät (wahlweise AIM 65/PC100, PET, CBM oder APPLE mit Assembler), Interfaceprogrammierung.
16. und 17. Mai 1980 (Fr/Sa) Einführendes Programmierseminar in Ahrensburg bei Hamburg: Übung des Befehlsvorrates, der Adressierungsarten, Programmiertechniken, Hardware der 65er-Prozessoren.
Anmeldung und Sonderprospekt beim Herausgeber.