65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7,80 Nr. 13 Juni 1980

Nach der Hannover-Messe

und in Anbetracht einer recht stürmischen Entwicklung in unserer Prozessorfamilie sollte einmal kurz zur Besinnung innegehalten werden. Hardwaremäßig und mit erweiterten Betriebsprogrammen und BASIC-Befehlen sind Commodore mit der Serie 8000 und das Haus Siemens mit dem PC 1000 in Erscheinung getreten. Diese Systeme werden im einzelnen besprochen.

Auf der Softwareseite finden wir sowohl von diesen Firmen wie aber auch von privaten Softarehäusern sehr wertvolle Programmpakete. Commodore schiebt jetzt einen sehr leistungsfähigen Assembler/Text-Editor für Maschinensprache nach, der auch eine Library-Anweisung kennt. Für CBM dürfte PASCAL sehr bald verfügbar sein. - Ebenso für CBM steht in verschiedenen Versionen ein Wordprocessor für die Textverarbeitung mit einer Leistungsfähigkeit zur Verfügung, die man sonst nur auf guten professionellen Textverarbeitungssystemen findet. - Der freie Markt bietet für den CBM überdies zahlreiche Hilfen zur maschinensprachlichen Programmierung an, wie NEWTIM S, TRAMON oder den Macro-Assembler von Moser.

Mit der Kombination von z.B. einem Assembler und NEWTIM S wird der CBM zu einem Entwicklungssystem mit einer überdurchschnittlichen Ausrüstung für die Programmierung in Maschinensprache. Hinzu kommen bekanntermaßen ein leistungsfähiges BASIC und systemkompatible E/A-Einheiten wie Floppy Disk, Printer und Plotter.

b.w.

Inhaltsverzeichnis

Das Konzept des AIM 65, der hierzulande in eineinhalb Jahren etwa 5000mal ausgeliefert wurde und damit zu den meistverkauften Systemen zählt, war von Anfang an anders. Er wurde vorwiegend als Entwicklungssystem entworfen und vom Hersteller mit einem leistungsfähigen Monitorprogramm, einem Assembler und umfangreicher Dokumentation versehen, so daß der freie Markt hierfür kein zusätzliches Angebot schaffen mußte. Solcher Bedarf trat ein, um offensichtliche Begrenzungen des BASIC zu überwinden und um geeignete Hardware bereitzustellen, Netzteil, Gehäuse, Video-Interface, breitformatiger Drucker, RAM-Erweiterungen, Floppy Disk und z.T. Cassettensysteme. Viele AIMs bleiben damit auch als Entwicklungssystem in stetiger Entwicklung, und man dürfte bei den Betreibern eine enorme Konfigurationsvielfalt finden.

Die Ausbaubedürftigkeit des AIM 65 veranlaßte das Haus Siemens, aus diesem System den betriebsfertigen PC 100 zu schaffen. Die beachtliche Zahl ausgelieferter PC 100 spricht dafür, daß auch unser europäischer Markt zunehmend Fertiggeräte braucht. Neuerdings bietet auch die Firma KEB Computer in Berlin Fertigsysteme auf der Basis des Rockwell-Gerätes an und geht damit auf diesen Trend ein.

Beobachtungen auf der Hannover-Messe und bei anderen Gelegenheiten weisen unsere kleinen Computer als Fertiggeräte in großer Stückzahl auf Anwendungen hin, die wir im Zusammenhang mit Stichworten wie Tischcomputer, Textsystem, Messen und Regeln, Meßwerterfassung usw. zuvorderst gar nicht so sehr bedenken, nämlich auf Anwendungen in der dezentralen Datenerfassung und in der Datenübertragung. In zahlreichen Betrieben werden die kleinen preiswerten Systeme bereits als eine Art Fernschreiber für den Informationsaustausch zwischen Arbeitsplätzen betrieben, in anderen verwalten sie die Zugangs- und Anwesenheitskontrolle an Toren und Türen. Uns sie finden Verwendung als Terminals an größeren Rechnern.

Fertiggeräte werden ebenso auch im Ausbildungswesen benötigt. Ihr Einsatz in berufs- und allgemeinbildenden Schulen steht erst ganz am Anfang und ist noch von der Initiative einzelner abhängig.

Wenn man die vorhandenen und angekündigten Systeme, den rapiden Zuwachs an Software und die erkennbar noch offenstehenden Felder der Anwendung betrachtet, dann kommt man zu dem Schluß, daß unsere 8-Bit-Rechner erst mit der Arbeit beginnen. In dieser Sicht sind die vergangenen Jahre der notwendige Vorlauf gewesen. Und wo die zentrale CPU die Arbeit nicht mehr allein schafft, da wird man sie durch intelligente Interfacebausteine und durch Peripheriegeräte mit eignem 8-Bit-Prozessor entlasten.

Hinzu kommt, daß die Arbeitsbreite von 8 Bit optimal allen Verarbeitungen und Übertragungen von Texten entspricht (7-8 Bit ASCII). Die Prozessoren erreichen hier häufig eine Überlegenheit gegenüber Maschinen mit größerer Arbeitsbreite.

FORTSETZUNG AUF SEITE 51

****

DIREKTABONNENTEN ERHALTEN 2 EXEMPLARE DES HEFTES NR. 13 MIT DER BITTE UM WEITERLEITUNG UND EMPFEHLUNG DIESER ZEITSCHRIFT BEI IHNEN BEKANNTEN SYSTEMBETREIBERN.

Christian Streicher, Wittelsbacher Str. 24, 8034 Germering

Daten Ein-/Ausgabe für das AIM-BASIC

Das AIM-BASIC hat gegenüber den meisten anderen gebräuchlichen BASIC-Interpretern den Nachteil, daß es nicht ohne weiteres möglich ist, Datensätze vom Tonband einzulesen, bzw. auf Tonband abzuspeichern. Um die zu diesem Zweck benötigten Routinen in das AIM-BASIC zu implementieren, benutzt der Autor eine geänderte Version des in Heft 11 veröffentlichten Programmes 'USCOM'. Die Änderung dieses Programms besteht lediglich darin, daß die vom Anwender definierten Befehle auch im Programmlaufmodus abgearbeitet werden können.

Beim näheren Studium des BASIC-Interpreters stellte der Autor fest, bei der Abarbeitung der PRINT-, GET- Und INPUT-Routinen ein Unterprogramm durchlaufen wird, das das INFLG ($A412) und das OUTFLG in $A413 abfragt. Die Überlegung, daß eine Datenausgabe auf Band durch Verändern des betreffenden Flags möglich ist, bestätigte sich nach den ersten Experimenten mit POKE als richtig. Allerdings wurde bei dieser Art der Datenaufzeichnung weder die Routine TAPOUT noch die Routine DUMPTA durchlaufen, so daß die Tape-Header nicht berücksichtigt wurden. Es lag daher nahe, die in BASIC allgemein üblichen Befehle OPEN und CLOSE zur Bandsteuerung einzufügen.

Die Syntax der Befehle ist wie folgt:' :OPN$="logische Gerätenummer, Input (oder Output), Filename" ' für OPEN und ' :CLS ' für CLOSE. Der Befehl 'OPN' wird vom Steuerprogramm sowohl als Befehl als auch als Informationszuweisung bearbeitet. Dabei werden die nach dem Befehl stehenden Steuerinformationen mit Hilfe der in Zelle $B814 beginnenden BASIC-Routine der Stringvariablen PN$ zugewiesen. Auf diese Weise ist es möglich, die Steueranweisungen auch dynamisch zuzuweisen. Die Syntax ist dann dieselbe wie bei der Zuweisung einer Stringvariablen (Komma nicht vergessen!).

Die Abspeicherung der Daten erfolgt im ASCII-Code und wird vom CLOSE-Befehl mit zwei 'CR' abgeschlossen, so daß eine Bearbeitung der Daten auch vom EDITOR aus möglich ist. Abgespeichert werden dürfen alle in BASIC üblichen Variablenarten, auch in gemischter Reihenfolge.

Die Definition der log. Gerätenummern ist 0 für Tape Nr. 1 und 1 für Tape Nr. 2. Entspricht die eingegebene Zahl nicht der eines Tapes, so erfolgt Abbruch mit 'UF-ERROR'.

Das Steuerprogramm darf nur nach dem Initialisieren des BASIC geladen werden.

```
0000                *=$CA
00CA        BOEF    BCS EXTRA     ;SN-ERROR AUF ATN-VEKTOR
00CC                *=$BB
00BB EXTRA  4C0A3E  JMP ANF       ;ATN-VEKTOR AUF STARTADR.
00BE
00BE INFLG          =$A412
00BE OUTFLG         =$A413
00BE                *=$3E00
3E00        ;---- VARIABLEN ----
3E00 GOTO           *=*+2
3E02 SAVY           *=*+1
3E03 SAVX           *=*+1
3E04 SAVA           *=*+1
3E05 LAC6           *=*+1
```

```
3E06 LASTX  00       .BYT 0
3E07 V1              *=*+1
3E08 V2              *=*+1
3E09 V3              *=*+1
3E0A         ;----PROGRAMMANFANG----
3E0A ANF    08       PHP
3E0B        8D043E   STA SAVA
3E0E        8E033E   STX SAVX
3E11        8C023E   STY SAVY      ;REGISTERINHALTE RETTEN
3E14        AE053E   LDX LAC6
3E17        E4C6     CPX $C6       ;PRÜFE INPUT-BUFFER-VEKTOR
3E19        F01D     BEQ BEG       ;OB REM-BEFEHL BEREITS
3E1B        E000     CPX #0        ;ABGEARBEITET IST
3E1D        F019     BEQ BEG       ; ODER KEIN USER-COMMAND
3E1F        A4C6     LDY $C6       ; VORLIEGT
3E21        8C093E   STY V3        ;$C6 RETTEN
3E24        86C6     STX $C6       ;$C6 AUF REM-BEFEHL
3E26        AE073E   LDX V1
3E29        BDCE3E   LDA COMM,X    ;BIT 1 DES USR-COMM ÜBER
3E2C        A000     LDY#$0        ;REM-ANWEISUNG KOPIEREN
3E2E        8C053E   STY LAC6
3E31        91C6     STA ($C6),Y
3E33        AD093E   LDA V3        ;$C6 WIEDER HERSTELLEN
3E36        85C6     STA $C6
3E38 BEG    AE063E   LDX LASTX
3E3B        F011     BEQ LOOP1     ;GÜLTIGER BEFEHL SCHON
3E3D        E8       INX           ;GEFUNDEN?
3E3E        8E063E   STX LASTX
3E41        E002     CPX#$2
3E43        D056     BNE END       ;BUCHSTABE 2 IM INPUT-
3E45        A200     LDX#$0        ;BUFFER
3E47        8E063E   STX LASTX     ;PROGRAMMSTATUS LÖSCHEN
3E4A        28       PLP
3E4B        6C003E   JMP (GOTO)    ;USER-BEFEHL AUSFÜHREN
3E4E LOOP1  DDCE3E   CMP COMM,X    ;USER-BEFEHL VEREINBART?
3E51        F013     BEQ NEXT1
3E53        A97F     LDA #$7F      ;DELIMITER FÜR LETZTEN
3E55        A003     LDY#$3        ;USER-BEFEHL
3E57 LOOP2  DDCE3E   CMP COMM,X    ;PRÜFE DIE NÄCHSTEN 4
3E5A        F037     BEQ ERR       ;ZEICHEN
3E5C        E8       INX           ;AUF ERR, WENN BEFEHL
3E5D        88       DEY           ;NICHT VORHANDEN
3E5E        10F7     BPL LOOP2
3E60        AD043E   LDA SAVA
3E63        4C4E3E   JMP LOOP1     ;NÄCHSTEN USER-BEFEHL
3E66 NEXT1  8E073E   STX V1        ;TESTEN
3E69        A002     LDY #2
3E6B        B1C6     LDA ($C6),Y
3E6D        8D083E   STA V2        ;3. BYTE AUS DEM INPUT-BUFFER
3E70        88       DEY           ;RETTEN
3E71        B1C6     LDA ($C6),Y
3E73        E8       INX           ;BYTE 2 MITEINANDER VER-
3E74        DDCE3E   CMP COMM,X    ;GLEICHEN
3E77        F009     BEQ NEXT2
3E79        AD043E   LDA SAVA
3E7C        E8       INX           ;WENN UNGLEICH:
3E7D        E8       INX           ;TESTE NÄCHSTEN USER-BEFEHL
3E7E        E8       INX
```

```
3E7F        4C4E3E  JMP LOOP1
3E82 NEXT2  E8      INX
3E83        AD083E  LDA V2        ;BYTE 3 MITEINANDER VERGLEICHEN
3E86        DDCE3E  CMP COMM,X
3E89        F01B    BEQ GOOD
3E8B        AD043E  LDA SAVA
3E8E        E8      INX           ;WENN UNGLEICH: TESTE NACHSTEN
3E8F        E8      INX           ;USER-BEFEHL
3E90        4C4E3E  JMP LOOP1
3E93 ERR    A200    LDX #0
3E95        8E053E  STX LAC6
3E98        8E063E  STX LASTX     ;BEFEHL NICHT GEFUNDEN
3E9B END    AD043E  LDA SAVA
3E9E        AE033E  LDX SAVX
3EA1        AC023E  LDY SAVY
3EA4        28      PLP
3EA5        60      RTS           ;ZURUCK AN BASIC-INTERPRETER
3EA6 GOOD   AD073E  LDA V1
3EA9        18      CLC           ;USER-ADRESSE BERECHNEN
3EAA        4A      LSR A
3EAB        AA      TAX           ;USER-ADRESSE RETTEN
3EAC        BDDB3E  LDA USADD,X
3EAF        8D003E  STA GOTO
3EB2        BDDC3E  LDA USADD+1,X
3EB5        8D013E  STA GOTO+1
3EB8        A000    LDY #0        ;REM-ANWEISUNG AN BASIC
3EBA        A98E    LDA #$8E      ;INTERPRETER
3EBC        8D043E  STA SAVA
3EBF        91C6    STA ($C6),Y
3EC1        A5C6    LDA $C6
3EC3        8D053E  STA LAC6
3EC3        A901    LDA #1        ;USER- KOMMANDO GULTIG
3EC8        8D063E  STA LASTX
3ECB        4C9B3E  JMP END
3ECE        ;USER DEFINED COMMANDS
3ECE
3ECE COMM   4154    .BYT 'ATN ','OPN ','CLS ',$7F
3ED2        4F50
3ED6        434C
3EDA        7F
3EDB        ;USER ADRESSEN
3EDB USADD  E13E    .WOR ATN,OPN,CLS
3EDD        E33E
3EDF        5A3F
3EE1
3EE1 ATN    60      RTS
3EE2
3EE2 INOUT          *=*+1
3EE3 HEX            =$EA7D        ;ADRESSEN IM MONITOR-PROGRAMM
3EE3 LOADTA         =$E32F        ;DES AIM
3EE3 DUMPTA         =$E56F
3EE3 CRLF           =$E9F0
3EE3 DU11           =$E50A
3EE3
3EE3 OPN    201488  JSR $B814     ;ABSPEICHERN DER
3EE6        A000    LDY #0        ;STEUERANWEISUNGEN
3EE8        B198    LDA ($98),Y   ;STRINGLÄNGE IN Y-REGISTER
3EEA        AA      TAX
```

```
3EEB          C8        INY
3EEC          B198      LDA ($98),Y  ;STRINGADRESSE
3EEE          8596      STA $96      ;ZWISCHENSPEICHERN
3EF0          C8        INY
3EF1          B198      LDA ($98),Y
3EF3          8597      STA $97
3EF5          A000      LDY #0
3EF7          B196      LDA ($96),Y  ;LOGISCHE GERÄTE-NR. BERECHNEN
3EF9          207DEA    JSR HEX
3EFC          C902      CMP #2
3EFE          B03B      BCS NOTAP    ;KEINE TAPE-#
3F00          8D34A4    STA $A434    ;TAPE EINSCHALTEN
3F03          8D35A4    STA $A435
3F06          202B3F    JSR NEXT     ;I/O-ANWEISUNG HOLEN
3F09          8DE23E    STA INOUT    ↑ UND ZWISCHENSPEICHERN
3F0C          A900      LDA #0
3F0E          8D033E    STA SAVY+1
3F11 LOOP3    202B3F    JSR NEXT     ;FILENAME IN 'NAME'
3F14          8C023E    STY SAVY     ;ABSPEICHERN
3F17          AC033E    LDY SAVY+1
3F1A          992EA4    STA $A42E,Y
3F1D          C8        INY
3F1E          8C033E    STY SAVY+1
3F21          C005      CPY #5
3F23          F01B      BEQ M2
3F25          AC023E    LDY SAVY
3F28          4C113F    JMP LOOP3
3F2B
3F2B NEXT     C8        INY          ;NÄCHSTEN CHAR. HOLEN
3F2C          B196      LDA ($96),Y
3F2E          CA        DEX
3F2F          3007      BMI M1
3F31          F005      BEQ M1
3F33          C930      CMP #$30
3F35          90F4      BCC NEXT
3F37          60        RTS
3F38 M1       A920      LDA #$20     ;MIT 'SPC' FÜLLEN
3F3A          60        RTS          ;FÜR FILENAME
3F3B
3F3B NOTAP    A220      LDX #$20     ;LOG. GERÄTE-NR. ENTSPRICHT
3F3D          4C59B2    JMP $B259    ;KEINER TAPE-#
3F40 M2       ADE23E    LDA INOUT    ;INPUT ODER OUTPUT?
3F43          C94F      CMP #"O
3F45          D003      BNE *+5      ;KEIN OUTPUT
3F47          4C523F    JMP TAPOUT
3F4A          A954      LDA #'T      ;VOM TAPE LESEN
3F4C          8D12A4    STA INFLG
3F4F          4C2FE3    JMP LOADTA
3F52
3F52 TAPOUT   A954      LDA #'T      ;AUSGABE AUF TAPE
3F54          8D13A4    STA OUTFLG
3F57          4C6FE5    JMP DUMPTA
3F5A
3F5A CLS      ADE23E    LDA INOUT    ;CLOSE FILE
3F5D          C94F      CMP #'O      ;UND BLOCK AUFFÜLLEN
3F5F          D009      BNE M3
3F61          20F0E9    JSR CRLF     ;CR NUR FÜR EDITOR
3F64          20F0E9    JSR CRLF
```

```
3F67        200AE5  JSR DU11
3F6A M3     A9CF    LDA #$CF      ;TAPE AUSSCHALTEN
3F6C        2D00A8  AND $A800
3F6F        8D00A8  STA $A800     ;IN DEN PORT SCHREIBEN
3F72        A90D    LDA #$0D      ;I/O-DEVICE AUF
3F74        8D12A4  STA INFLG     ;TASTATUR ZURÜCKSETZEN
3F77        8D13A4  STA OUTFLG
3F7A        60      RTS           ;ZURÜCK ZUM BASIC
3F7B                .END
3F7B         ERRORS = 0000
```

TESTPROGRAMM FÜR EINE BASIC EIN-/AUSGABE:

```
 10 DIM A(30)
 20 FOR X=0 TO 30
 30 A(X)=SIN (3*X)
 40 NEXT
 50 :OPN$="Ø,0,SIN"
 60 FOR X=0 TO 30
 70 PRINT A(X)
 80 NEXT
 90 :CLS
100 CLEAR
110 DIM A(30)
115 :OPN$="Ø,I,SIN"
120 FOR X=0 TO 30
130 INPUT A(X)
140 NEXT
150 :CLS
160 END
```

User Defined BASIC Commands V2.1

Das vorstehende Programm 'Daten Ein-/Ausgabe für das AIM-BASIC' ist mit seinem ersten Teil die aktualisierte Version des in Heft 11 veröffentlichten USCOM. Der zweite Teil ab ATN ist ein Beispiel dafür, wie man selbstdefinierte Kommandos zur Ausführung bringt. Auf ähnliche Weise könnte man nun Datenverkehr mit einer Floppy-Disk implementieren. - Herr Streicher gibt außerdem folgende Hinweise zur jetzigen USCOM-Version:

Beim Aufruf von User-Befehlen besteht jetzt lediglich die Einschränkung, daß sie nicht als erster Befehl in einer Zeile stehen dürfen. Falls ein User-Befehl dennoch der einzige in einer Zeile sein soll, muß ihm ein ':' voranstehen. Ansonsten können die USCOM-Befehle beliebig gegeben werden, auch programmgesteuert.

Hier noch die Verbesserungen zur ersten Version: Will man z.B. zusätzliche mathematische Funktionen realisieren, so ist es sinnvoll, den User-Befehl nicht mit 'REM' (hex 8E) zu überschreiben, sondern mit 'ATN'. Man kann dann eine eingegebene Zahl oder Funktion aus dem Floating-Point-Akkumulator im BASIC-Format übernehmen. Will man die Zahl jedoch im ASCII-Code und ohne 'VARIABLE=USR(...) übernehmen, zB. bei SYS(...), dann kann die Zahl direkt nach dem User-Befehl stehen. Der Pointer auf die Variable steht in hex C6. Arithmetische Ausdrücke sind hier jedoch nicht mehr erlaubt.

Wie liest man ein Programm-Listing?

Bei Durchsicht der Zeitschriften, Bücher und Dokumentationen wird der Leser eine große Zahl verschiedener Darstellungsformen für Programme finden, die ihn anfangs etwas verwirren mögen. Daher folgende Übersicht, die keinen Anspruch auf Vollständigkeit erhebt:

1. Formale Analyse

1.1 Hexadezimale Dumps

Es handelt sich um den nackten, unerklärten Ausdruck von Speicherzelleninhalten, bei dem jedes Byte in Form von zwei Halbbytes (nibbles) hexadezimal dargestellt wird. Um eindeutige Verhältnisse zu schaffen, ist eine Vorspalte hinzugefügt. Sie enthält die (hexadezimale) Adresse des ersten Bytes in der Zeile. Je Zeile werden meistens die Inhalte von 4 Bytes abgedruckt. Beispiel:

```
ADRESSE  INHALT

0324     A9 FF 85 B4
0328     20 87 02 A0
```

Der Inhalt von 0324 ist also A9, von 0325 dann FF usw..

Aus Platzgründen werden oft auch die Inhalte von 16 Speicherzellen pro Zeile abgedruckt. - Der sehr verbreitete AIM 65 erzeugt unter M-Kommando:

```
<M>=0324 A9 FF 85 B4
< > 0328 29 87 02 A0
```

Das M und das Gleichheitszeichen werden also für die Folgezeilen unterdrückt. Weniger glücklich ist das Format unter D-Kommando (Dump). Es beginnt mit einem Semikolon, es folgen zwei hexadezimale Ziffern mit einer Mengenangabe für den Dump, dann vier Ziffern mit der Startadresse und schließlich die Inhalte. In unserem Beispiel entsprechend:

```
;080324A9FF85B4 ...
```

Nur geübte Maschinenprogrammierer können aus einem Dump ein Programm in mnemonischer Form zurückgewinnen. Dumps sind daher in erster Linie für Kontrollzwecke gedacht, nicht aber für die Darstellung und Weitergabe von Programmen - und noch weniger für belehrende Zwecke. Unter diesem Gesichtspunkt ist es zu bedauern, wenn große deutsche Zeitschriften noch immer Dumps für die Zwecke der Programmverbreitung abdrucken, zumal das damit bedingte 'blinde' Nachtippen von Ziffernfolgen wenig merkfähig und damit fehleranfällig ist.

1.2 Disassemblierte Darstellung

Die nächstbessere Darstellungsform enthält bereits Befehle in mnemonischer Notierung, wie z.B. LDA, STX usw.. Dabei erinnern wir uns, daß die Befehls- und Informationsdarstellung im Prozessor rein binär ist. Es wird also ein Hilfsprogramm benötigt, das aus diesen binären Mustern einen lesefähigen Text erzeugt. Das ist der Disassembler.

1.2.1 Ausgabeformate

Die in die Systeme implementierten Disassembler haben durchaus verschiedene Ausgabeformate. Je Zeile wird jedoch stets ein Befehl disassembliert, und alle Werte erscheinen in hexadezimaler Notierung. Am

Zeilenbeginn steht in 4 Ziffern die Adresse des ersten Befehlsbytes, das bekanntlich den Opcode enthält. Meistens folgt das eigentliche Opcode-Byte, darauf seine mnemonische Notierung als Befehl sowie der Operand in einer Schreibweise, die auch die Adressierungsart bezeichnet, also nach den oben verwendeten Vorgaben z.B.:

```
ADRESSE OPCODE MNEMON. OPERAND
                BEFEHL

0324        A9 LDA     #FF
0326        85 STA     B4
0328        20 JSR     0287
```

Andere Disassembler schreiben gleichwertig z.B.:

```
0234  LDA #FF     USW.

ODER

0324 A9 FF    LDA #FF
```

In manchen Fällen mag vor die Spalte mit den Adressen noch eine laufende Zeilennummer treten.

1.2.2 Adressierungsarten

Die Schreibweise des Operanden kennzeichnet seine Adressierungsart. Die 65xx kennen 13 Adressierungsarten. Bei den 1-byte-langen Befehlen sind Operand undAdressierungsart durch den Befehl selbst festgelegt, der Disassembler schreibt daher auch keinen Operanden an. Lediglich bei den Verschiebebefehlen ASL, LSR, ROL und ROR tritt ein 'A' als Operand hinzu, um den Unterschied zur Verschiebung im Speicher deutlich zu machen, bei denen als Operand eine Adresse erscheint.

Disassemblierte Befehle mit nur 1 Adreßbyte im Operanden sind Zeropage-Befehle, solche mit 2 Adreßbytes sind absolute Befehle. Ein Zahlenkreuz (#) vor dem Operanden bedeutet Direktoperand, sein Wert wird also dem Programmspeicher aus der dem Opcode folgenden Zelle entnommen.

Ein Komma im Operanden bedeutet indizierte Adressierung, also: ADRESSE,X und ADRESSE,Y.

Eine Klammer um den Operanden bedeutet indirekte Adressierung, also

6C JMP (0398)	Indirekter Sprung. Lade den Programmzähler mit dem Inhalt der Speicherzellen 0398,0399.
B1 LDA (045),Y	Indirekte Adressierung, mit Y indiziert: Lade Akku mit dem Inhalt einer Zelle, deren Adresse sich aus dem Inhalt des Pointers (0045,0046)+Y ergibt.
A1 LDA (45,X)	Indirekte Adressierung, indiziert mit X. Hole die in 45+X und 45+X+1 hinterlegte Adresse des zu ladenden Bytes und lade Akku.

Bei den Verzweigungsbefehlen haben die Disassembler zweierlei Notierungen des Operanden: Entweder es ist bereits in zwei Bytes das Verzweigungsziel ausgerechnet

```
F0 BEQ 0356
```

oder es steht da nur 1 Operandenbyte:

```
F0 BEQ 0D ,
```

dann ist es direkt dem Befehl entnommen und stellt den hexadezimalen Offset für den Programmzähler für den Fall dar, daß die Verzweigungbedingung zutrifft. Die letztere Darstellungsart ist natürlich weniger aussagekräftig.

Disassembler sind 'dumm'. Sie bemerken nicht, wenn sie nach einer Programmfolge z.B. in eine Tabelle oder in einen noch nicht geplant gefüllten Bereich hineinstolpern. Für diese können sie daher nur Unsinn ausdrucken. - Bei Programmen, die nur als hexadezimaler Dump (s.o.) vorliegen, ist der Disassembler das erste Hilfsmittel zur Programmanalyse und zu seinem Verständnis. Dem Betreiber bleibt die Mühe, die einzelnen Zeilen 'nachzukommentieren'.

1.3 Analyse von Assembler-Programmen

In Heft 7 dieser Zeitschrift gingen wir auf den AIM-Assembler als einen typischen Vertreter für die kleinen Systeme ein. Zur Erinnerung:

Ein Assembler ist ein Dienstprogramm, daß Maschinencode (Objectcode) und wahlweise auch eine Assemblerliste erzeugt. Als Programmquelle dient ein vorwiegend in mnemonischer Form zeilenweise formulierter Text (Source), der im Text-Editor angelegt wurde. Dieser Text darf daneben besondere Anweisungen und Erklärungen an das Unwandlungsprogramm sowie Kommentare enthalten, so daß ein rein symbolisches Programmieren möglich wird. Für die Maschinen-Ebene ist Assembler-Programmierung mithin der Rolls Royce, zumal der Assembler auch zahlreiche Fehler erkennt. Gleichwohl bedarf die Analyse von Assemblertexten einiger Unterscheidung und des Einlesens, denn wir finden hier noch mehr Darstellungsformen.

1.3.1 Quelltexte

Zunächst einmal halten wir Quelltext und Assemblerliste deutlich auseinander. Um einen Gedanken, ein Programm knapp zu formulieren, bedarf es zunächst nur des Quelltextes. Und so finden wir in zahlreichen Programmdarstellungen nur diesen. Häufig ist nicht einmal eine Anfangsadresse für das Programm vorgeschlagen. Ein Beispiel dafür ist in diesem Heft enthalten, die 'Interruptgetriebene Kassettenein- und -ausgabe'.

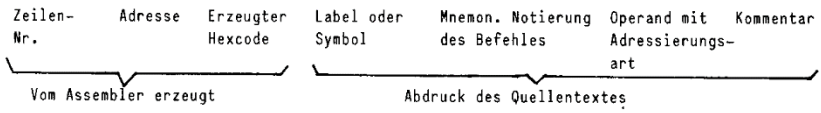
1.3.2 Assembler-Listings

Die Normalform der Darstellung und auch die aussagekräftigste ist jedoch die vom Assembler erzeugte Programmliste. Sie enthält in ihrer rechten Hälfte noch einmal das Quellenprogramm nebst Kommentaren (wir gehen von einer kolonnenweise formatierten Liste aus). Links davon ist Zeile für Zeile das erzeugte Maschinenprogramm abgedruckt, und zwar in hexadezimaler Notierung (in der Reihenfolge, in der die Bytes im Speicher abgelegt werden).

4C 02 05 ist also der Maschinencode für JMP nach 0502. Beispiele für diese einfachste Notierung pro Zeile finden wir in diesen Heft unter 'Anschluß von numerischer Anzeige und Tastatur ...'.

Verbesserte Assemblerversionen enthalten ganz links eine laufende Zeilen- oder 'Karten'-Nummer, die in manchen Versionen dem Quelltext entnommen werden konnte, in anderen erst vom Assembler hinzugefügt wurde. Eine zweite Spalte enthält unter 'LOC' (Location) Zeile für Zeile die Adresse, unter der der Opcode des erzeugten Befehls abgelegt wurde.

Stromlinienförmig wird ein Assembler eigentlich erst mit einer Symboltafel am Schluß der Assembler-Liste - möglichst alphabetisch geordnet - und vielleicht einer Cross Reference List, die alle Zeilennummern oder Programmadressen aufführt, die auf einen symbolischen Begriff Bezug nehmen. Hier das allgemeine Listenformat:



1.3.2.1 Kommentare

Was ist daneben das Besondere an Assembler-Listen? Dem Leser fällt bei veröffentlichten Programmen auf, daß sie viel Kommentare enthalten. Reine Kommentarzeilen des Quelltextes beginnen mit einem Semikolon. Es werden mitunter aber auch andere Sonderzeichen benutzt. Kommentare werden aber auch direkt an die Befehle angehängt. Ein Semikolon ist dann im allgemeinen nicht erforderlich, außer wenn es Kollisionen mit reservierten Worten der Assembler-Sprache geben könnte.

1.3.2.2 Der Zuordnungszähler = Sternadresse

Bei Kommentarzeilen fällt uns auf, daß der Inhalt in der Spalte LOC (Adresse) nicht weitergeschaltet wird. Wir kommen damit auf den Zuordnungszähler, die Sternanweisung (*=), der in einem Quellenprogramm beliebig häufig gesetzt werden kann, Z.B. *=$900. Unter dem jeweiligen Stand des Zuordnungszählers werden bei der Assemblierung die für das Maschinenprogramm erzeugten Bytes abgelegt. Mit jedem erzeugten Byte wird die Sternadresse um 1 hochgezählt, so daß sie immer den nächsten freien Programmspeicherplatz ausweist. Kommentare erzeugen keine Befehlsbytes, also wird der Zuordnungszähler auch nicht erhöht.

Assemblerprogramme setzen an Anfang die Sternadresse auf 0000. Dieser Wert wird beibehalten, bis im Quellenprogramm eine erste Zuweisung mit *= erfolgt.

Die Sternadresse bezeichnet also die Stelle, unter der der Assembler das nächste Byte ablegen soll. Sie ist nicht mit dem Programmzähler zur Ausführungszeit zu verwechseln. Wohl besteht für den praktischen Gebrauch Identität zwischen beiden bei fast allen Befehlen, nicht jedoch bei den Verzweigungsbefehlen. Das hängt mit der Prozessorstruktur zusammen (Pipelining):

```
Maschinencode  -  Quelltext
F0 03          .  BEQ *+5
```

In diesen Fällen besteht also eine Differenz von 2, um die der Programmzähler vorausläuft. Diese Zeile ist zugleich ein Beispiel dafür, daß man sich beim Programmieren auf den Zuordungszähler beziehen darf.

1.3.2.3 Zahlensysteme

Zahlen- und Wertangaben im Quelltext können im allgemeinen in den Zahlensystemen binär (%), oktal (@), dezimal (ohne) oder hexadezimal ($) notiert werden. Zur Interpretation der Ziffernfolge ist das in den Klammern angeführte Ziffernzeichen voranzustellen.

Der Ausdruck des Wertes in der Assembler-Liste erfolgt aber immer hexadezimal, z.B.:

```
A9 04    LDA #%100
```

1.3.3 Der Erklärungsteil - Symbole

In Assemblerprogrammen (Quelltexte und Listen) bemerken wir eine Zweiteilung in Erklärungs- und Programmteil.

Der Assembler erlaubt die Verwendung von Symbolen im Quelltext, die dem Erklärungsteil zuzurechnen sind. Wir können z.B. formulieren:

```
LDA #VIER, STATT LDA #04 ODER LDA #4 BZW. LDA #%100
```

Symbole, die konstante Rechen- oder Adreßwerte vertreten, erleichtern die gedankliche Arbeit des Programmierers erheblich. Dem Übersetzungsprogramm, dem Assembler, müssen diese Werte aber zunächst mitgeteilt werden, damit es mit ihnen verfahren kann. Daher der Erklärungsteil mit der Wertzuweisung durch Gleichheitszeichen an die Symbole, also z.B.:

```
VIER=$4 ODER
VIER=%00000100,
OUTPUT=$E97A ODER
OUTPUT=59770
```

Der Assembler wird diese Werte neben dem Namen in einer Symboltafel hexadezimal ablegen und bei jeder Bezugnahme von dort her verwerten.

1.3.4 Der Programmteil

1.3.4.1 Befehle und Marken

Hinsichtlich der mnemonischen Notierung der Befehle und der Adressierungsarten durch die Schreibweise der Operanden ergeben sich keine Unterschiede zum Abschnitt 1.2. Abweichungen bestehen in den Zeilen hinsichtlich der Einfügung von Kommentaren (s.o.) und zweitens hinsichtlich der Notierung von Zahlenwerten. Für diese muß das Zahlensystem gem. Prefix nach 1.3.2.3 erklärt sein. Ein Wert E97A wird nicht akzeptiert, weil das dezimale Zahlensystem die Ziffern E und A nicht kennt, nachdem das fehlende $-Zeichen eine dezimale Zahl erklären wollte. Führende Nullen müssen nicht getippt sein.

Der Programmteil bietet außer der Verwendung der erklärten Symbole aber einen weiteren Komfort: Man darf den augenblicklichen Stand des Zuordnungszählers (*) mit einem symbolischen Namen versehen, mit einer Marke, einem Label. Auch diese Marke wird in der Symboltafel verwaltet und kann für jede Art von Bezug verwendet werden, z.B. um Einsprungs- oder Verzweigungspunkte ansprechbar zu machen oder um Unterprogramme zu benamen. Der Programmierer braucht sich also nicht mehr darum zu kümmern, unter welcher Adresse der Befehl tatsächlich abgespeichert wird.

Wegen ihrer besonderen Bedeutung werden die Marken in den Assemblerlisten hervorgehoben. Gute Assembler schreiben sie in eine eigene Spalte. Bei geringer Zeilenbreite nimmt man nur die graphische Herausstellung vor, wie z.B. beim AIM 65 mit 2 Gleichheitszeichen:

```
==0200 START
```

1.3.4.2 Speicherplatzbelegung mit Werten

Einen Unterabschnitt des Programmteiles bilden fast regelmäßig auch Speicherplatzbelegungen mit Tabelleninhalten oder sonstigen Operatoren. Hierfür dienen drei Anweisungen, die alle die Sternadresse hochzählen, nämlich

.BYT, .WOR und .DBY bzw. .BYTE, .WORD und .DBYTE

Der vorangestellte Punkt kennzeichnet sie als Assembleranweisungen für die Umwandlungsphase, z.B.:

```
TABL .BYT $44,68,%01000100
```

Es werden 3 Speicherplätze belegt, alle mit hex 44. Das erste Byte ist unter der Marke TABL anzusprechen. Die Verkettung der Belegung erfolgt durch die Kommata.

Die Anweisung .WOR legt 2 Bytes in der umgekehrten Reihenfolge ab, in der sie geschrieben sind, also

ADR .WOR $E97A als Bytefolge 7AE9.

Sie eignet daher für die Hinterlegung von Adreßvektoren per Assembler. - Die Anweisung .DBY legt zwei Bytes in der Reihenfolge der Schreibweise ab.

1.3.4.3 Reservierung von Arbeitsspeicher

Bei den Symbolen des Erklärungsteiles ist davon auszugehen, daß sie während des Programmablaufes einen gleichbleibenden (konstanten) Wert des Operators oder der Adresse behalten. Nun will man aber auch Zwischenergebnisse in namentlich bezeichneten Arbeitsspeichern und Ein- und Ausgabepuffern ablegen können. Dazu bedient man sich der Sternanweisung, z.B.

```
*=$B0
ARB1 *=*+1
ARB2
ARB3 *=*+1
```

Diese drei Anweisungen reservieren 2 Bytes Arbeitsspeicher. ARB1 wird unter B0 reserviert, ohne daß dort etwas hineingeschrieben wird. Das zweite Byte unter B1 ist künftig unter dem symbolischen Namen ARB2 und wahlweise auch unter ARB3 ansprechbar. Diese Gleichstellung erfolgt, weil die Sternadresse von der einen Zeile zur anderen nicht hochgezählt wurde, so daß sich beide Marken unter der selben Sternadresse befinden.

Mit der Sternadresse wird oft auch beweglich bei der Zuweisung von Adreßkonstanten gearbeitet. Gesetzt den Fall, ein VIA-Interfacebaustein sei unter $A800 decodiert. Dann könnten wir dem Assembler erklären:

```
*=$A800
PORTB *=*+1
PORTA  *=*+1
DDRB *=*+1
usw.
```

Wenn wir nur PORTB, DDRB und das IER benötigen, so wird eleganter erklärt:

```
*=$A800
PORTB
DDRB=*+2
IER=*+$E
```

1.3.4.4 Operatoren

Die arithmetischen Operatoren + und - führen zu einer Berechnung auf der Basis des Symboles/Labels:

LDA TABL+1 lädt z.B. das zweite Byte der Kette.
JMP START+2 führt zum Programmsprung nach 0202 im Beispiel.

Die Operanden der Befehle sind hier als (Rechen-)Ausdrücke gebildet. Es können mehrere Werte/Symbole/Labels rechenhaftig verkettet werden und dürfen auch verschiedenen Zahlensystemen entstammen.

Der Operator > betrifft das höhere Byte eines Symboles/Labels, der Operator < das niedere.

1.3.4.5 ASCII-Bytes und Operanden

Auch ASCII-Strings können mit der .BYT-Anweisung abgelegt werden. Die Zeichen sind dann in Anführungszeichen enthalten, z.B.:

STRING .BYT 'BEISPIEL'

Einzelne ASCII-Bytes dürfen in der Assemblersprache auch als Direktoperanden verwendet werden, z.B.:

CMP $'S. Der Assembler erzeugt daraus C953.

1.4 Assembler- Anweisungen

Die Assemblersprache stellt nach den bisherigen Erklärungen bereits eine erhebliche Erweiterung gegenüber der reinen Befehlssprache dar, die nur mnemonische Befehle und hexadezimale Adressen verwendet (z.B. der I-Command des AIM 65). Die Assembleranweisungen *=, .BYT, .WOR, .DBY, die Operatoren, die Symbole und Labels wurden als Teile der Sprache bereits angesprochen, ebenso das Semikolon für Kommentare. Hinzu treten Anweisungen, die wir im allgemeinen nicht ausgelistet finden und die für die Interpretation eines Programmes nicht wesentlich sind. Sie betreffen die Herkunft des Quelltextes, den Speicherbereich für die Symboltafel und die Ablage des Maschinenprogrammes.

Weitere Anweisungen bemerken wir an ihren Wirkungen. Sie beginnen mit .OPT und betreffen die Assemblerliste. - Erwähnenswert ist noch die Anweisung .FILE, die die Heranziehung eines weiteren namentlichen Textfiles von Tonband, Diskette etc. bewirkt.

Die Assembler für 65xx unterliegen einer Weiterentwicklung. Für die CBM gibt es in verschiedenen Versionen einen Makro-Assembler (Moser), der die Erzeugung von Programmteilen aus hinterlegten allgemeinen Ablaufstrukturen zuläßt, die durch Parameterübergabe ausgefüllt werden (s.a. Heft 3: Makros für 65xx).

Von Commodore wird seit wenigen Wochen ein sehr leistungsfähiger Assembler nebst Text-Editor vertrieben, der an anderer Stelle besprochen wird. Er verwendet natürlich die für 65xx sehr gleichartige Assemblersprache und beinhaltet zusätzlich die Library-Funktion .LIB, mit der man Quelltext-Module aus einer Bibliothek zusätzlich heranziehen kann. Hierzu wird man sicher noch viele Programmveröffentlichungen sehen.

2. Inhaltliche Analyse

Dieser Abschnitt muß naturgemäß kürzer ausfallen, weil wir hier kein konkretes Programmlisting besprechen, sondern allgemeine

Untersuchungsgesichtspunkte mitteilen wollen. - Die vorbesprochenen Darstellungsformen 1 und 2, Dump und Disassembly, reichen für eine Analyse nicht aus. Es bedarf dort in jedem Falle der Bearbeitung und Nachkommentierung. Hierzu wird man die die Veröffentlichung begleitenden Erklärungen heranziehen, um die Bedeutung von Speicherzelleninhalten und Einsprungspunkten aufzuhellen. Bezogen auf eine fehlende Assemblerliste bedeutet das: Nachvollzug des Erklärungsteiles und der Labels.

Blockdiagramme findet man selten, auch in dieser Zeitschrift, weil ihre Anfertigung etwas graphisches Geschick und zusätzlichen Zeitaufwand erfordert. Längere Programme sollte man aber in jedem Fall in Erledigungsblöcke auflösen. Man wird dabei von oben nach unten vorgehen: Fluß des Haupt- oder Vordergrundprogrammes, Feststellung, auf welcher Kommandoebene man sich jeweils befindet.

Gute Programmiertechnik beinhaltet regelmäßig die Verwendung von Unterprogrammen. In der nächsten Stufe wird man also bestimmen, welche Dienstleistungen/Erledigungsblöcke die Unterprogramme im einzelnen erbringen. Zur rein optischen Programmgliederung ist es empfehlenswert, die Endpunkte von Unterprogrammen mit einem Markierstift farbig kenntlich zu machen.

Auch in Vordergrundprogrammen gibt es Programmsäcke, 'dead ends', Stellen von denen immer wieder nach oben oder unten gesprungen oder verzweigt wird. Der eigentliche Aussprungspunkt mag in der Mitte liegen. Diese Aussprungspunkte sollte man ebenfalls kennzeichnen, um im groben eine Vorstellung vom Programmablauf zu erhalten. In manchen Fällen mag man das Programm natürlich auch im Einzelschritt-Modus auf seinem System ausprobieren, um die Verläufe zu erkennen.

In einer weiteren Zergliederung wird man feststellen, welche Programmteile der eigentlichen Aufgabenerledigung dienen und welche der eigentlich immer notwendigen Ein- und Ausgabe. - Beim E/A wird man bemerken, daß besondere Einflüsse von der Interfacebeschaltung herrrühren. Bei der Ansprache von System-Interfaces wie Tastatur, Printer, Bildschirm, Cassette, Floppy usw.. wird man dabei die Heranziehung von Routinen aus dem Monitorprogramm bemerken. Man erkennt sie regelmäßig am angesprochenen ROM-Adressenbereich. Solche E/A-Routinen kann man im ersten Durchgang als Erledigungsblöcke betrachten, wie auch andere aus dem ROM-Bereich stammende. Das Interesse kann damit dem Haupterledigungsteil zugewandt werden.

Die Untergliederung zeigt hier meistens einen Verwaltungsteil, der Bedingungen initialisiert, Parameter bereitstellt, verändert und prüft, und einen ausführenden Teil. Z.B. bei einer Addition:

```
        LDX #$4     X ALS ADDRESSER UND ZÄHLER  }
        SED         MODUS                       } VERWALTUNG
        CLC         ADDITIONSVORBEREITUNG       }

AUSF    LDA OP1,X                               }
        ADC OP2,X                               } AUSFÜHRUNGSTEIL
        STA OP2,X                               }

        DEX                                     }
        BPL AUSF                                } VERWALTUNG
        CLD                                     }
```

Man sollte dabei zunächst die Funktion des Ausführungsteiles erkennen, in diesem Fall die Addition von OP1 und OP2 nach Ergebnisfeld OP2. Der Verwaltungsteil zeigt die Bedingungen auf: Addiere solange X positiv, d.h. addiere 5 Bytes (DO ... WHILE X IS POSITIVE).

Es gibt wohl keine allgemeingültige Empfehlung, wie man ein Programm analysiert. Eigene Programmierung und der Nachvollzug veröffentlichter Programme ergeben erst die Übung. Vorstehende bewährte Gliederungsgesichtspunkte mögen behilflich sein, in jedem Falle gehört die rein technische Kenntnis dazu, was die Notierungen in den Programmzeilen und Assembleranweisungen bedeuten.

R.L.

****

LITERATURHINWEISE

BASIC-Handbuch, Personal-Computer PC 100. Siemens AG, Ausgabe 1980/81, 80 S., Bestell-Nr. B/2237. - Deutsche Übersetzung und Bearbeitung für das im PC 100 und im AIM 65 implementierte BASIC. Das Buch entspricht dem englischen BASIC Language Reference Manual von Rockwell.

INTERACTIVE ist der Titel einer von Rockwell International Electronic Devices herausgegebenen neuen Zeitschrift. Sie gibt speziell Hinweise für AIM-Betreiber. Das erste von Eric C. Rehnke bearbeitete Heft umfaßt 12 Seiten. Erwähnenswert ist ein Checksum Program und 'Data Files for AIM BASIC'. Der Abonnementspreis ist $8 für Europa. Bestellungen an den Newsletter Editor, Rockwell International, P.O. Box 3669, RC55, USA-Anaheim, CA 92803.

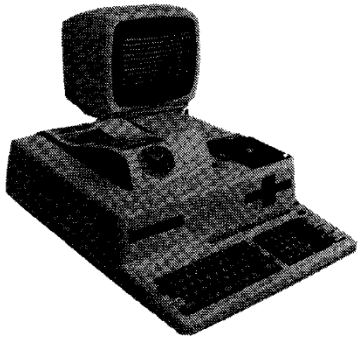
COMPUTE und COMPUTE II sind neue amerikanische Zeitschriften, die vorwiegend auf 65er Computer und -prozessoren ausgelegt sind. COMPUTE befaßt sich mit PET/CBM, APPLE und Atari, COMPUTE II zweimonatlich mit den Einplatinencomputern, wie KIM, SYM, AIM. Die ersten Ausgaben enthalten bereits zahlreiche Programme und Hinweise. Wir werden darauf noch eingehen. Diese Zeitschriften können auch hier zu je DM 40,- Endpreis für 6 Ausgaben abonniert werden, und zwar bei Ingenieurbüro Manfred Schöffel, Rohlsdorfer Weg 33, 2409 Techau, Tel.: 04504-281 und 282. Bei dieser Firma sind weiterhin die amerikanischen COMMODORE USER NEWSLETTERS zu DM 50,- im Jahr beziehbar.

Die 6502 USER NOTES von Eric C. Rehnke schließen mit Heft 17 ab und stellen ihr Erscheinen ein. Ihre Nachfolge besteht in COMPUTE und speziell in COMPUTE II.

Programme für Kleincomputer und Taschenrechner, Sonderheft Nr. 31 der FUNKSCHAU, Franzis-Verlag 1980, 84 Seiten, DM 14,80. - Neben Programmen, die den Funkamateur besonders interessieren, finden wir einen Text-Editor für KIM-1, ASCII-Tastatur für KIM und Simulation des 8080 als wesentliche neue Vorschläge.

Siemens PC 1000

Auf der Hannover-Messe 1980 präsentierte das Haus Siemens seinen PC 1000, einen aufwärtskompatiblen Personal Computer auf der Basis des 6502. Das für dieses Gerät vorhandene und noch geplante Leistungsspektrum ist beachtlich und wird ihm professionelle Anwendungen beim Rechnen, beim Textverarbeiten, beim Steuern und Regeln sowie in der Ausbildung eröffnen.



Hauptmerkmale: Komplettsystem im Gehäuse mit Mini-Floppy-Disk (116 kByte), komfortabler Dateneingabetastatur nach DIN 2136 und zusätzlichen Tastenblöcken für Zifferneingabe und Textverarbeitung, Ablaufkontrolle und 8 frei programmierbaren Funktionstasten. Arbeitsspeicher 32 kB RAM und Betriebssystem in 24 KB ROM-resident, einschl. BASIC-Interpreter und DOS. Das BASIC ist mit seinem Anweisungssatz ausserordentlich erweitert und sowohl auf Editierfähigkeit, File-Bearbeitung (mit Floppy Disk) und Graphik auf Bildschirm und Thermodrucker ausgerichtet. Hinzu treten so schöne Befehle wie RENUMBER, TRACE, ON TIME, MAT, MATPRINT, DO WHILE, ELSE, ON mit verschiedenen Bedingungen, DIGITIZE, PLOT, INIT UNIT usw..

Durch ein besonderes ROM-BAnksystem kann der PC 1000 auf bis zu 8 Megabyte zugreifen (256 Banken zu 32 kB). Dadurch ist es möglich, künftigen Erweiterungen, z.B. dem geplanten PASCAL und Anwenderprogrammen feste Adreßbereiche zuzuweisen.

Der PC 1000 hat verschiedene AV-Interfaces zum Anschluß von externen Schwarz/Weiß- und Farbsichtgeräten. Ein HF-Interface macht auch die handelsüblichen Empfänger mit ihrer geringeren Bandbreite anschließbar. Ein Multiscreen-Interface erlaubt die Speisung eines weiteren Sichtgerätes mit der Ausgabe eines getrennten Inhaltes. Die Bildschirminterfaces haben das Format 25x40 Zeichen und 287x200 Bildpunkte bzw. 25 Zeilen x80 Zeichen und erlauben frei programmierbare Zeichensätze. Das AV-Interface übernimmt auch Tonfrequenz-Modem-Funktionen.

Ein 80-Zeichen-Thermodrucker/Plotter kann als Option in das Gehäuse eingebaut werden und erlaubt Graphik.

Zu den weiteren ausrüstbaren Interfaces gehören: Parallel-Interface für externe Drucker, Joy-Stick-Anschluß, IEC-BUS-Interface, V24-Interface, Slave-Processor-Interface.

Am Design ist weiter interessant, daß die eingebauten Peripheriegeräte über einen internen IEC-Bus verbunden sind. Bei angeschlossenem IEC-Bus-Interface ist der interne Bus ein Abbild des externen, solange der PC 1000 Controller ist. Es kann aber die Kontrolle an einen externen IEC-Bus-Controller abgegeben werden. Das Interface dient dann als Buskoppler.

Die vorgenannten Spezifikationen lassen erkennen, daß das Haus Siemens mit dem PC 1000 ein eigenständiges, vielfältiges und sehr modernes Design verfolgt. Es wurde Preise von etwa DM 5.500 (ohne Bildschirm und Drucker) genannt. Lieferung etwa ab Oktober 1980.

Der CBM-Assembler

Auf der Hannover-Messe kündigte Commodore einen Floppy Disk-orientierten Assembler zur Freigabe und Lieferung nach der Messe an. Nach gegenwärtigem Stand hat sich die Freigabe jedoch noch verzögert. Zu diesem Assembler folgende vorläufige Beschreibung, der später eine ausführliche Darstellung folgen soll:

Lieferung auf Diskette (ca. DM 500,-) mit Benutzeranleitung, Text-Editor und Assemblerprogramm. Der 'Mini'-Editor hat alle guten Eigenschaften, die man zur Bearbeitung von Quelltext in Assemblersprache benötigt. Er erlaubt ein zeilenorientiertes Arbeiten am Bildschirm mit Cursor-Funktionen wie beim Eingeben von BASIC-Statements. Mit dem Befehl AUTOxx werden Zeilennummern im Abstand von xx vorgegeben. Ein RENUMBER, auch für einzelne Textbereiche, ist möglich. Die automatische Numerierung kann abgeschaltet werden. Einzelne Textzeilen werden gelöscht, indem man wie bei BASIC nur die Zeilennummer und RETURN drückt. Mit DELETE können Textblöcke von ... bis gelöscht werden. Die Formulierung einer Zeile mit einer früher schon benutzten Zeilennummer führt zur Übernahme des neuen Textes. War die Nummer der neuen Zeile noch nicht benutzt, so wird die Zeile als ein INSERT übernommen. - Ändert man einzelne Zeichen in den auf dem Bildschirm angezeigten Zeilen, so werden auch diese Änderungen unmittelbar in den Quelltext übernommen.

Der FORMAT-Befehl arbeitet wie LIST unter BASIC. Man kann den Quelltext zurückrufen, auch in Blöcken von ... bis Zeile.

Daneben gibt es zeichenkettenorientierte Befehle wie FIND String (mit oder ohne Bereichsangabe) und CHANGE String to New String (mit oder ohne Bereichsangabe und Wiederholung). Mit GET"Name" werden Quelltextmodule von Floppy Disk hinzugeladen, mit PUT auf Floppy abgespeichert.

Der Editor dient der Formulierung und Bearbeitung von Quelltexten, die in Assemblersprache zu formulieren sind. Für letztere gibt es eine bestimmte Syntax, die sich mit derjenigen anderer 6502-Assembler deckt (s. z.B. Heft 7 für den AIM 65-Assembler).

Zur Assemblersprache gehören die üblichen mnemonischen Befehlsformulierungen wie z.B. LDA, die Operandenbezeichnungen in synbolischer Schreibweise oder mit expliziter Adressierung und der durch die Schreibweise des Operanden festzulegenden Adressierungsart (z.B. Operand,X für indizierte Adressierung). Der Assembler akzeptiert ebenso im Programmablauf definierte Marken, Label, ist also voll mnemonisch und symbolisch. Ebenso verarbeitet er Rechenausdrücke, wobei Operatoren für Addition, Subtraktion, Multiplikation und Division zulässig sind.

Auch die Assembleranweisungen sind auf weite Strecken wie gewohnt: .BYTE, .WORD und .DBYTE zur Ablage von Werten im Speicher, = und * für EQUATE und Zuordnungszähler, ferner die die Assemblerliste beeinflussenden .OPT-Anweisungen, .PAGE und .SKIP.

Zur Verbindung von Quelltext-Files dienen .FILE und, man beachte, .LIB. Mit der Library-Funktion können Quelltext-Module aus einer Bibliothek entnommen werden.

Das eigentliche Assemblerprogramm wird ebenfalls von der Floppy geladen. Es arbeitet in 2 Durchgängen (passes) am Quelltext-File oder

an den verbundenen Files auf Floppy und es erzeugt unter den gegebenen Anweisungen drei Ausgabefiles, die Assemblerliste nebst Symboltafel, das Maschinenprogramm (Interface File) als ladefähiges Modul und die Fehlerliste. Bei der Assemblerliste ist vom Anwender beeinflußbar, ob sie auf das IEE-Interface (CBM-Drucker) oder auf den Userport gerichtet werden soll (Centronix Schnittstelle).

Nach dieser Beschreibung erfüllt der CBM-Assembler alle Erwartungen an ein leistungsfähiges Hilfsmittel zur symbolischen Programmierung in Maschinensprache und dürfte damit einen zunehmenden Benutzerkreis finden, zumal die Zusammenarbeit mit Floppy Disk ein zügiges Arbeiten gewährleistet.

R.L.

****

KLEINANZEIGEN DER LESER

SUCHE GEBRAUCHTEN AIM 65 kompl. m. Zubehör. Willi Gietmann, Ahornstr. 9, 4174 Issum 1, Tel.: 02835-2381.

AIM 65 4K, Alugehäuse, Netzteil, Anschlüsse herausgeführt, 5 Steckplätze VG 64-Bus, DM 1.2000,-, Frank Philipp, 05241-67480.

Handbücher für AIM 65, engl., komplette Sätze günstig DM 25,- mit User's Guide, Hardware- und Programming Manual, Monitor Listing, 2 Reference Cards, Schaltplan. H-J. Regge, Fesenfeld 57, 28 Bremen 1, Tel. 0421-71114.

****

Michael Zimmermann, Eberstädter Str. 170, 6102 Pfungstadt

Anschluß von numerischer Anzeige und Tastatur (3)

10. Erweiterte Matrixanordnung

Bei der Tasturanordnung im vorigen Beispiel haben wir, ähnlich wie bei der Anzeige, noch ohne Dekodierung gearbeitet. Dies führte dazu, daß lediglich 16 Schalter mit einem Port bedient werden können, zu wenig für unser Entwicklungsziel.

Aus diesem Grunde scheint es sinnvoll, auch hier eine Decodierung vorzusehen und die Tastaturabfrage der multiplexen; einen entsprechenden Aufbau gibt **Abb. 11.** Hier sind bereits Tastatur- und Anzeigemultiplexing verbunden. Als Display wurde die teilcodierte Form gewählt. Hier kann nicht mehr - wie imvorigen Beispiel - mit einer einzigen Abfrage erkannt werden, ob eine Taste gedrückt ist. Vielmehr kann hier nur festgestellt werden, ob ein Schalter in der gerade betrachteten Zeile betätigt wurde. Durch Ansprechen aller Zeilen nacheinander kann ermittelt werden ob und welcher Schalter der Gesamttastatur betätigt wurde.

Da dieser Durchlauf sowieso für die Anzeige erforderlich ist, wird die Zeilenabfrage einfach an die Anzeige für eine Ziffer angehängt und bei gedrücktem Key dessen Muster in MKEY gespeichert. Nach Durchlauf aller Ziffern (und damit Spalten) der Tastatur wird geprüft, ob in MKEY ein Wert steht, der in der beschriebenen Weise in Ziffer oder Kontrollzeichen umgewertet wird. - Mit dieser Schaltung der Tasten sind wir in der Lage, unserer Wunschtastatur zu entsprechen. Die folgenden Beispiele stellen nur noch Modifikationen dar.

Die Schritte zur Tastenerkennung sind in **Programm 8** nochmals nachzulesen, ebenfalls werden hier die erforderlichen Tabellen für die Bitmuster gegeben.

```
PROGRAMM 8
--------------------
ADVANCED DISPLAYDRIVER

==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
==0000 DDRB
       = DRB  ;DATA DIRECTIONB
 ==0000 CRB=DRB+1
==0000 BUF
       =$00   ;DISPLAYBUFFER
==0000 SHFT
       =$08   ;POSITION AFTER BUFFER
==0000 LKEY
       =$09   ;LAST KEY
==0000 MKEY
       =$0A   ;MOMENTARY 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 M10
       +=*
203402 JSR DISP ;DISPLAY BUFFER
205E02 JSR KEY
;CHECK FOR KEYBOARD ENTRY
90F8   BCC M10
;HERE WE CAN PROCESS FUNCTION KEYS
E00F   CPX #$0F
D0F4   BNE M10
20B402 JSR CLR
4C0602 JMP M10
```

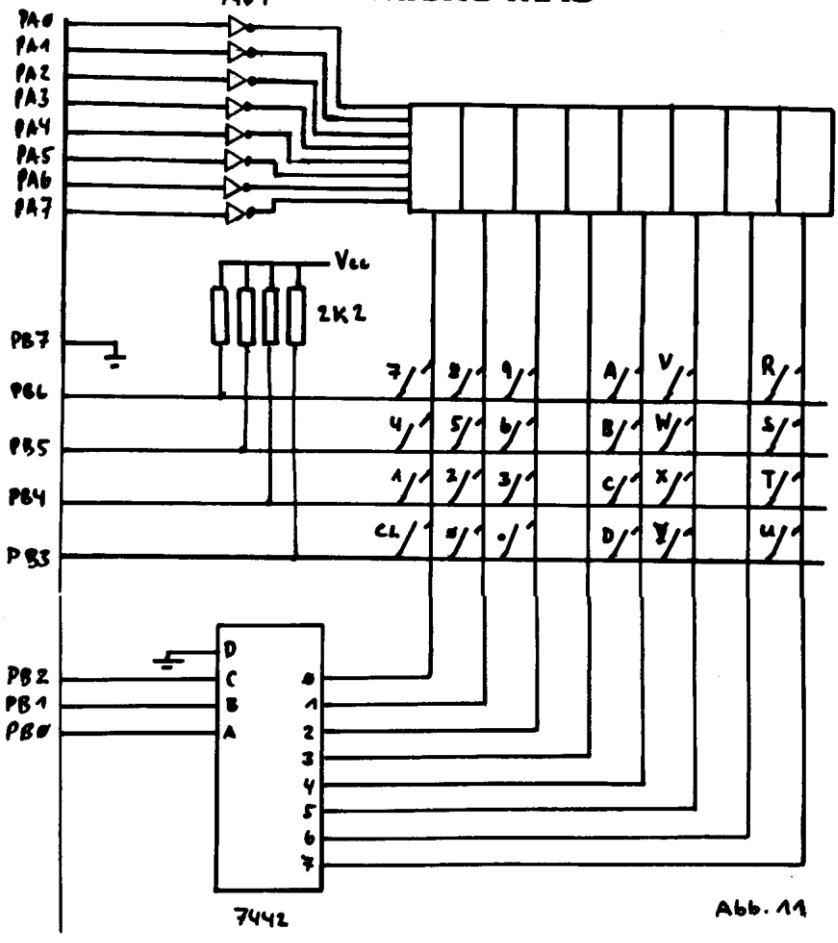


Abb. 11

```
--------------------
INITIALIZE PIA
==0218 INIT
       *=*
A900   LDA #$00
;SET CRA/CRB TO DDRA/DDRB-ACCESS
8D0190 STA CRA
8D0390 STA CRB
A9FF   LDA #$FF ;SET DIRECTION OUT
8D0090 STA DDRA
A907   LDA #$07
8D0290 STA DDRB
A904   LDA #$04
;NOW ADDRESS DATA-REGISTER
8D0190 STA CRA
8D0390 STA CRB
60     RTS
```

```
--------------------
GET CHARCTERS TO BE DISPLAYED
==0233 GET
       *=*
;SEE NEXT CHAPTER FOR DETAILS
60     RTS
--------------------
DISPLAY BUFFER
==0234 DISP
       *=*
A9FF   LDA #$FF
;STORE A UNVALID MOMENTARY KEY
850A   STA MKEY
A207   LDX #$07
;SET POINTER TO BUFFER
```

```
==023A D20
       *=*
B500   LDA BUF,X
;GET BUFFER CHARACTER
8D0090 STA DRA
;AND PUT IT INTO LEDS
8E0290 STX DRB
;STORE CHARACTER-POSITION
A0FF   LDY #$FF
;NOW DELAY
==0244 D30
       *=*
88     DEY
D0FD   BNE D30
AD0290 LDA DRB
;GET KEY ENTRY
2978   AND #$78
;CHECK FOR KEY DEPRESSED
C978   CMP #$78
F005   BEQ D90
;NO KEY, RETURN
AD0290 LDA DRB
;GET KEYPATTERN
850A   STA MKEY
;AND DEPOSIT AT MOMENTARY KEY
==0255 D90
       *=*
A9FF   LDA #$FF
;SIGN OFF LEDS
8D0090 STA DRA
CA     DEX
;GO TO NEXT CHRACTER
10DD   BPL D20
60     RTS
--------------------
CHECK FOR KEYBOARD-ENTRY

==025E KEY
       *=*
A50A   LDA MKEY
;CHECK WHICH KEY IS DEPRESSED
C9FF   CMP #$FF
;NO KEY
F029   BEQ K90
;RETURN
C509   CMP LKEY
;SAME AS LAST TIME
F029   BEQ K99
;RETURN
8509   STA LKEY
;TAKE AS LAST KEY FOR THE NEXT TIME
A217   LDX #$17
;NOW CHECK PATTERN
==026C K20
       *=*
DD9C02 CMP KEYTAB,X
F005   BEQ K50
;FOUND IN PATTERN-TABLE
CA     DEX
;CHECK NEXT PATTERN
10F8   BPL K20
3017   BMI K90
;FOUND NOTHING, RETURN
==0276 K50
       *=*
E00A   CPX #$0A
;CHECK FOR NUMERIC
B017   BCS K99
;RETURN IF FUNCTION
BD9202 LDA LEDTAB,X
;LOAD LED-PATTERN
8508   STA SHFT
;AND STORE IT INTO SHIFT
A200   LDX #$00
;NOW SHIFT NEW ENTRY INTO DISPL
==0281 K60
       *=*
B501   LDA BUF+1,X
9500   STA BUF,X
E8     INX
E009   CPX #$09
D0F7   BNE K60
18     CLC
9004   BCC K99
;RETURN WITH CARRY CLEARED
==028D K90
       *=*
A9FF   LDA #$FF
;MAKE VOID LAST KEY
8509   STA LKEY
==0291 K99
       *=*
60     RTS
--------------------
LED-PATTERNS
==0292 .BYT $C0,$F9,$A4,$B0,$99
       .BYT $92,$82,$F8,$80,$98
;TABLE OF KEY-PATTERNS
==029C KEYTAB
       .BYT $71,$68,$69,$6A,$58
       .BYT $59,$5A,$38
       .BYT $39,$3A,$74,$6C,$5C
       .BYT $3C,$72,$70
==02AC .BYT $75,$6D,$5D,$3D,$77
       .BYT $6F,$5F,$3F
--------------------
CLEAR DISPLAY

==02B4 CLR
       *=*
A207   LDX #$07
A9FF   LDA #$FF
==02B8 CL10
       *=*
9500   STA BUF,X
CA     DEX
10FB   BPL CL10
60     RTS
       .END
```

11. Verbesserte erweiterte Matrixanordnung

Am vorigen Beispiel einer Matrixanordnung kann durchaus noch die eine oder andere Verbesserung vorgenommen werden. So ist es z.B. möglich, die Erkennung einer betätigten Taste dadurch zu vereinfachen, daß alle Zeilen auf ein NAND-Gatter mit 4 Eingängen geschaltet werden. Sind alle Leitungen hoch, d.h. ist keine Taste gedrückt, so braucht die angesprochene Spalte nicht bearbeitet zu werden. Ist hingegen eine der Leitungen durch Schalterdruck auf den Decoder geschaltet und damit bei angesprochener Spalte Null, so wird das entsprechende Muster übernommen.

Abb. 12 zeigt hier die Hardware, **Programm 9** gibt die Modifikationen zum Programm für die vorige Anordnung. - Vom Entwickler muß hier entschieden werden, ob in diesem Falle die Erweiterung in der Hardware der nur geringen Einsparung in der Software angemessen ist.

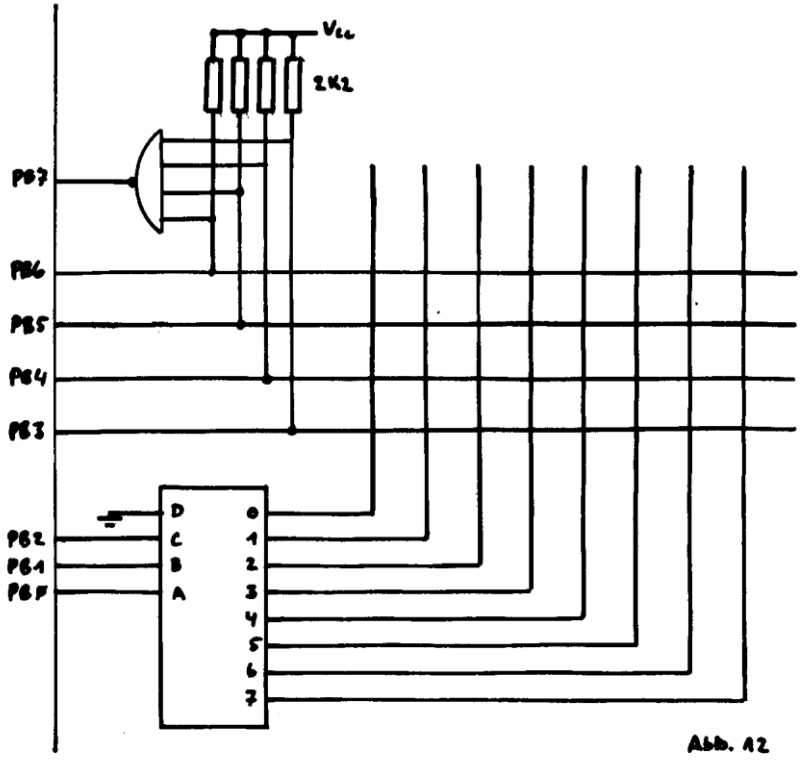


Abb. 12

PROGRAMM 9

```
--------------------
DISPLAY BUFFER
==0234 DISP
       *=*
A9FF   LDA #$FF
;MAKE VOID MOMENTARY KEY
850A   STA MKEY
A207   LDX #$07
;SET POINTER TO BUFFER
==023A D20
       *=*
B500   LDA BUF,X
;GET BUFFER CHARACTER
8D0090 STA DRA
; AND PUT IT INTO LEDS
8E0290 STX DRB
; STORE CHARACTER-POSITION
```

```
A0FF   LDY #$FF
; NOW DELAY
==0244 D30
       *=*
88     DEY
D0FD   BNE D30
2C0290 BIT DRB
; CHECK IF ANY KEY
1007   BPL D90
AD0290 LDA DRB
; GET KEYPATTERN
297F   AND #$7F
850A   STA MKEY
; AND DEPOSIT AT MOMENTARY KEY
==0253 D90
       *=*
A9FF   LDA #$FF
; SIGN OFF LEDS
8D0090 STA DRA
CA     DEX
; GO TO NEXT CHARACTER
10DF   BPL D20
60     RTS
```

12. Tastatur mit Direktumwertung

In den vorigen Beispielen war es mit vernünftigem Aufwand nur möglich, das Bitmuster des Tastendruckes durch Suchen in einer Tabelle in Ziffer oder Steuercode umzuwerten. Erstrebenswert erscheint es, dieses Muster direkt als Index für eine Umsetzung zu nehmen.

In den bisherigen Beispielen würde dies Tabellen mit 256 bzw. 128 Elementen erfordern, von denen die meisten unbelegt wären - ein Aufwand, der in keiner Relation zu der erwarteten Verbesserung steht. Hier soll ein anderer Weg beschritten werden, imdem die anliegende Zeileninformation von 4 auf 2 Bits verdichtet wird. Als Baustein hierfür dient ein Priority-Encoder. **Abb. 13** zeigt einen möglichen Aufbau.

Hierdurch sind wir in der Lage, insgesamt 32 Tasten (wir belegen hiervon nur 24) als Bitmuster in 5 Bit zu verschlüsseln. Bei diesen Relationen ist es überhaupt kein Problem, das Bitmuster direkt als Index zu benutzen, um Ziffer oder Kontrollzeichen der gedrückten Taste direkt zu erhalten. **Programm 10** bedient eine derartige Tastatur und beinhaltet die erforderlichen Tabellen zur direkten Transformation. Für die Kontrollzeichen wurden lediglich Beispielswerte verwendet.

Mit dieser Tastaturform wollen wir unsere Betrachtungen beenden. Sicher sind noch viele weitere Möglichkeiten vorstellbar, der generelle Rahmen dürfte aber durch die hier beschriebenen Beispiele abgesteckt sein.

Literatur

(1) **The First Book of KIM**
(2) CHIP, Januar 1980
(3) MCS ALPHA-1 Handbuch
(4) Elektor, November 1977

PROGRAMM 10

```
--------------------
DISPLAY BUFFER
==0234 DISP
       *=*
A9FF   LDA #$FF
; MAKE VOID MOMENTARY ENTRY
850A   STA MKEY
A207   LDX #$07
; SET POINTER TO BUFFER
==023A D20
       *=*
B500   LDA BUF,X
; GET BUFFER CHARACTER
8D0090 STA DRA
; AND PUT IT INTO LEDS
8E0290 STX DRB
; STORE CHARACTER-POSITION
A0FF   LDY #$FF
; NOW DELAY
==0244 D30
       *=*
88     DEY
D0FD   BNE D30
2C0290 BIT DRB
; CHECK IF ANY KEY
3005   BMI D90
AD0290 LDA DRB
; GET KEYPATTERN
850A   STA MKEY
; AND DEPOSIT AT MOMENTARY KEY
```

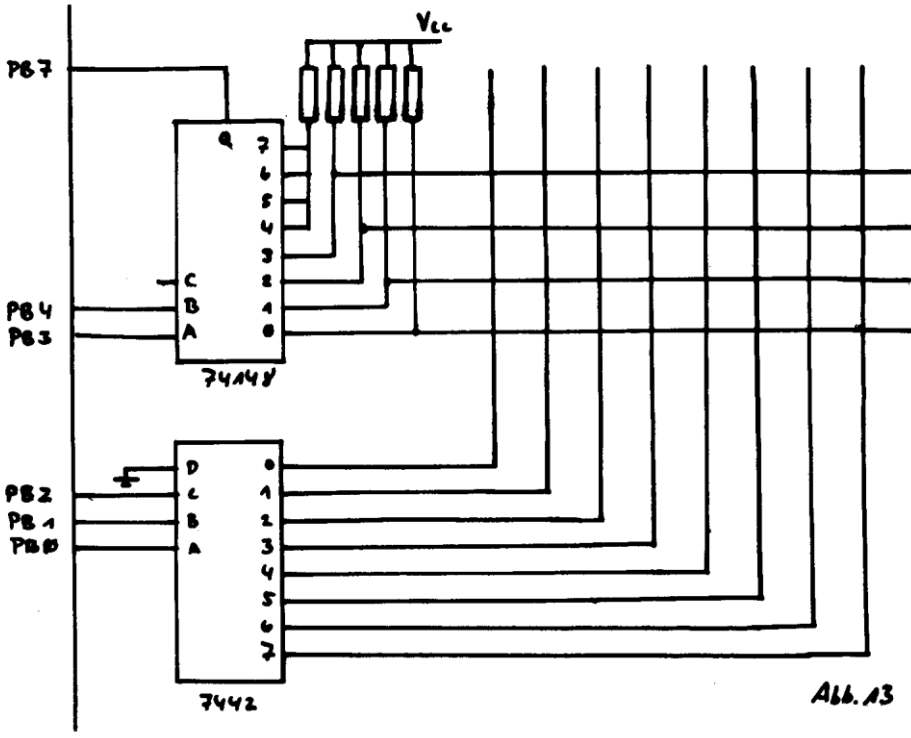


Abb. 13

```
==0251 D90
       *=*
A9FF   LDA #$FF
; SIGN OFF LEDS
8D0090 STA DRA
CA     DEX
; GO TO NEXT CHARACTER
10E1   BPL D20
60     RTS
--------------------
CHECK FOR KEYBOARD-ENTRY
==025A KEY
       *=*
A60A   LDX MKEY
; CHECK WHICH KEY IS PRESSED
E0FF   CPX #$FF
; NO KEY
F021   BEQ K90
; RETURN
E409   CPX LKEY
; SAME AS LAST TIME
F021   BEQ K99
; RETURN
8609   STX LKEY
; MAKE KEY LAST KEY FOR NEXT TIME
BD9002 LDA KEYTAB,X
; CONVERT PATTERN TO NUMERIC
C90A   CMP #$0A
; CHECK FOR NUMERIC
==026B
B018   BCS K99
; RETURN IF FUNCTION
AA     TAX
BD8602 LDA LEDTAB,X
; LOAD LED-PATTERN
8508 STA SHFT
; AND STORE IT INTO SHIFT
A200   LDX #$00
; NOW SHIFT NEW ENTRY INTO DISPLAY
==0275 K60
B501   LDA BUF+1,X
9500   STA BUF,X
E8     INX
E009   CPX #$09
D0F7   BNE K60
18     CLC
9004   BCC K99
; RETURN WITH CARRY CLEARED
```

```
==0281 K90
       *=*
A9FF   LDA #$FF
; MAKE VOID LAST KEY
8509   STA LKEY
==0285 K99
       2=2
60     RTS
--------------------
LED-PATTERNS
==0286 LEDTAB
       .BYT $C0,$F9,$A4,$B0,$99
       .BYT $92,$82,$F8,$80,$98
; TABLE OF KEY-PATTERNS
==0290 KEYTAB
       .BYT $0F,$00,$0E,$FF
       .BYT $10,$20,$FF,$30
       .BYT $01,$02,$03,$FF
       .BYT $11,$21,$FF,$31
       .BYT $04,$05,$06,$FF
       .BYT $12,$22,$FF,$32
       .BYT $07,$08,$09,$FF
       .BYT $13,$23,$FF,$33
```

NEWTIM S für CBM

Für die Programmentwicklung an den Commodore-Systemen der Serie 3000 steht jetzt ein neues leistungsfähiges Hilfsmittel zur Verfügung, der NEWTIM S. Zunächst ein Rückblick: Für den PET 2001 wurde der TIM-Monitor als Tonbandcassette geliefert. Das Programm gelangte an die Speicherstellen ab 1024 und kollidierte dort mit BASIC-Programmen. Immerhin erlaubte der TIM die Anzeige von hexadezimalen Speicherzellen- und Registerinhalten und ihre byteweise Abänderung, die Ausführung von Maschinenprogrammen sowie Laden und Abspeichern.

Bei der Serie 3000 befanden sich diese Dienstleistungen bereits im ROM des Betriebssystems und wurden dort mit SYS 1024 aktiviert. Diese Version steht dem Betreiber ständig zur Verfügung. In den vergangenen Monaten nun erlangte der NEWTIM 2 als ein freies Servicepaket mit seiner Bewährung zunehmenden Zuspruch. Die Autoren ließen jetzt den NEWTIM S folgen, der dem Entwickler weitere wertvolle Dienste zur Verfügung stellt. Unsere Besprechung behandelt diese brandneue Version und macht die dort enthaltenen neuen Utilities mit einem Stern (*) kenntlich.

Der NEWTIM S wird als ROM (*) geliefert und residiert im Bereich $9000-9FFF wobei die Cassettenpuffer nicht benutzt werden. Neu (*) ist der Einbezug der Floppy-Disk-Kurzbefehle, der sog. 'wedge', erweitert um einen SAVE-Befehl, die auch von BASIC her angesprochen werden können. Im Monitor finden wir einen Mini-Assembler, Breakpointsetzten (*), Kopieren von Speicherbereichen ohne und mit Umrechnung (Relocate), Disassemblieren, Execute to Breakpoint (*), Fill Memory, Goto, Byte/Stringsuche (jetzt bis 32 * Zeichen), Change String (*), Display als ASCII-Zeichen (*), Kontrolle auf unzulässige Opcodes, Einzelschrittausführung (*), auch als WALK mit Registeranzeige, Printer ein oder aus (*), Textzeile zum Printer (*), Registeranzeige und -veränderung, SAVE, Texteingabe in Speicher oder auf Bildschirm (*), Textausgabe auf Bildschirm (*), Dauertaste (*), Vergleichen von Speicherbereichen (*), Prozessorstatus binär anzeigen und ändern (*), IRQ zurücksetzen, arithmetische und logische Verknüpfungen mit Anzeige des Ergebnisses (*), Zahlenwandlung hex-dez-hex (*) und Mitbenutzung des NMI-Vektors.

Mit diesen Diensten wird der CBM zu einem leistungsfähigen Entwicklungssystem. Dem NEWTIM S sollte dabei ein leistungsfähiger Assembler zur Seite gestellt werden, der auch eine symbolische Programmierung zuläßt und der einen Text-Editor hat.

Bezug: Orgaplus G. Gailer KG, Fürther Str. 54-56, 8500 Nürnberg. Preis des Programms etwa DM 200,- + EPROM.

TEL. 0911-268646

Dipl.-Math. A. Quindt, Wiesbaden

Binäres Speichern von Zahlen

Der folgende Artikel beschäftigt sich mit dem Speichern von Daten auf Diskette, also mit dem Übergeben von Daten vom CBM 3001 an CBM 3040 und zurück. Insbesondere die Speicherung von Zahlen auf der Diskette soll dargestellt werden. Ein schnellerer Weg, der außerdem noch weniger Platz für die Speicherung der Zahlen auf der Diskette benötigt, wird dargestellt. Die Routinen in BASIC und Assembler werden gelistet.

Bei der Commodore-Floppy werden Zahlen in der Form von Strings gespeichert. Ein Beispiel:

Die Anweisung `PRINT#LOGICAL FILE,A`
bewirkt, daß auf der Diskette folgende Zeichen stehen, es sei A=25:
Blank, 2, 5, Blank, CR, LF
oder hexa

$20,32,35,20,0D,0A.

Für A=185987.42 steht dann:

Blank, 1, 8, 5, 9, 8, 7, ., 4, 2, Blank, CR, LF oder
hexa $20,31,38,35,39,38,37,2E,34,32,20,0D,0A.

Neben den Problemen, die beim Lesen durch das Line Feed ('LF') auftreten (FILA DATA ERROR) fällt auf, daß die Speicherung der Zahlen eine unterschiedliche Zahl an Bytes benötigt, sie ist von der Größe der Zahlen abhängig. Man benötigt, wenn man Line Feed und das Blank zwischen Carriage Return ('CR') und Zahl unterdrückt, zwischen 3 und 16 Bytes auf der Diskette.

Beispiel: A=0 mit Darstellung auf der Diskette $20,30,0D. Bei exponentieller Darstellung benötigt man für führendes Blank, 9 Ziffern, Punkt, E, Vorzeichen des Exponenten, Wert des Exponenten und Carriage Return 16 Byytes.

Es sollte einen Weg geben, eine Zahl unabhängig von ihrem numerischen Wert mit einem vorher bekannten und immer gleichen Platzbedarf auf eine Diskette zu schreiben. Das folgende Programm zeigt dazu einen Weg.

Die Zahlen werden im Computer in 5 Bytes dargestellt. Die Lösung besteht darin, exakt diese 5 Bytes auf Diskette zu schreiben und wieder zu lesen. Dazu wurde eine Übergabevariable im BASIC-Programm festgelegt. Diese hat den Namen AA und muß als erste Variable definiert werden. In den Bytes 42 und 43 swa CBM steht ein Pointer auf den Beginn der Variablen, und dieser zeigt dann direkt auf die Variable AA.

Eine abzuspeichernde Zahl muß dann in die Variable AA umgespeichert werden. Dann müssen die 5 Bytes, die den numerischen Wert enthalten, auf die Diskette geschrieben werden. - Beim Lesen müssen die 5 Byte von der Diskette geholt und in der Variablen AA an der entsprechenden Stelle gespeichert werden. Die folgenden Assembler-Programme zeigen einen Weg dazu:

```
ROUTINE OUT

$0398   LDA #$02
 039A   STA $03CB
 039D   LDA #$08
 039F   STA $D4
 03A1   JSR $F0BA       UP LISTN
 03A4   LDA #$68        SEKUNDÄRADRESSE
 03A6   STA $D3
 03A8   JSR $F128       UP SECND
 03AB   LDA $96         STATUSVARIABLE
 03AD   CMP #$01
 03AF   BEQ $16
 03B1   LDY $03CB
 03B4   LDA ($2A),Y     LADE 1 BYTE AUS AA
 03B6   JSR $F16F       UP CIOUT
 03B9   LDY $03CB
 03BC   INY
 03BD   CPY #$07
 03BF   BEQ $06
 03C1   STY $03CB
 03C4   JMP $03B4
 03C7   JSR $F183       UP UNLSN
 03CA   RTS


ROUTINE IN

$03CC   LDA #$02
 03CE   STA $03CB
 03D1   LDA #$08
 03D3   STA $D4
 03D5   JSR $F0B6       UP TALK
 03D8   LDA #$68        SEKUNDÄRADRESSE
 03DA   STA $D3
 03DC   JSR $F128       UP SECND
 03DF   LDA $96         STATUSVARIABLE
 03E1   CMP #$02
 03E3   BEQ $13
 03E5   JSR $F18C       UP ACPTR
 03E8   LDY $03CB
 03EB   STA ($2A),Y     SPEICHERE 1 BYTE IN AA
 03ED   INY
 03EE   CPY #$07
 03F0   BEQ $06
 03F2   STY $03CB
 03F5   JMP $03E5
 03F8   JSR $F17F       UP UNTLK
 03FB   RTS
```

In $03CB wird gezählt, wieviele Bytes bisher übermittelt wurden. Die Statusvariable (Name ST, Adresse $96) muß abgefragt werden, um ein sicheres Übergeben zu überprüfen. Sollte diese Variable ≠ 0 sein, muß der Vorgang wiederholt werden.

Nachfolgend wird ein BASIC-Programm gelistet, das die Übergabe von Zahlen an Floppy-Disk demonstrieren soll. Zur Kontrolle wird jede 100. erzeugte Zufallszahl auch auf den Bildschirm geschrieben.

```
520 CLR:AA=0:REM ÜBERGABEWERT
530 IN=972:OUT=920:T1=TI
540 AA=RND(-T1):REM ZUFALLSZAHLENGENERATOR STELLEN
550 OPEN15,8,15
560 PRINT#15,"I1":GOSUB 820
570 PRINT"■NUMMER    AUSGESCHR.    EINGELESENE ZAHL "
580 OPEN8,8,8,"@1:BINAERE ZAHLEN,S,W":GOSUB 820
590 FOR I=1TO1000
600 AA=RND(1)
610 AA=INT(1E8*AA+1E-3)/100
620 IF INT(I/100=I/100THEN PRINTI;TAB(10);AA
630 SYSOUT
640 IF1ANDSTTHEXX=XX+1:POKE150,0:GOTO630
650 NEXT
660 CLOSE8:GOSUB820
680 OPEN8,8,8,"1:BINAERE ZAHLEN,S,R,":GOSUB 820
690 FORI=1TO1000
700 SYSIN
710 IF2ANDSTTHENYY=YY+1:POKE150,0:GOTO700
720 IFINT(I/100)=I/100THENPRINTTAB(24);AA
730 NEXT
740 CLOSE 8: GOSUB 820
750 CLOSE 15
760 REM PROGRAMM OHNE PRINT-BEFEHLE UND OHNE FEHLERKANAL-UNTERPROGRAMM
770 REM AB ZEILENNUMMER 820
```

EIN PRINTOUT:

NUMMER	AUSGESCHR.	EINGELESENE ZAHL
100	94887.12	94887.12
200	469097.17	469097.12
...		

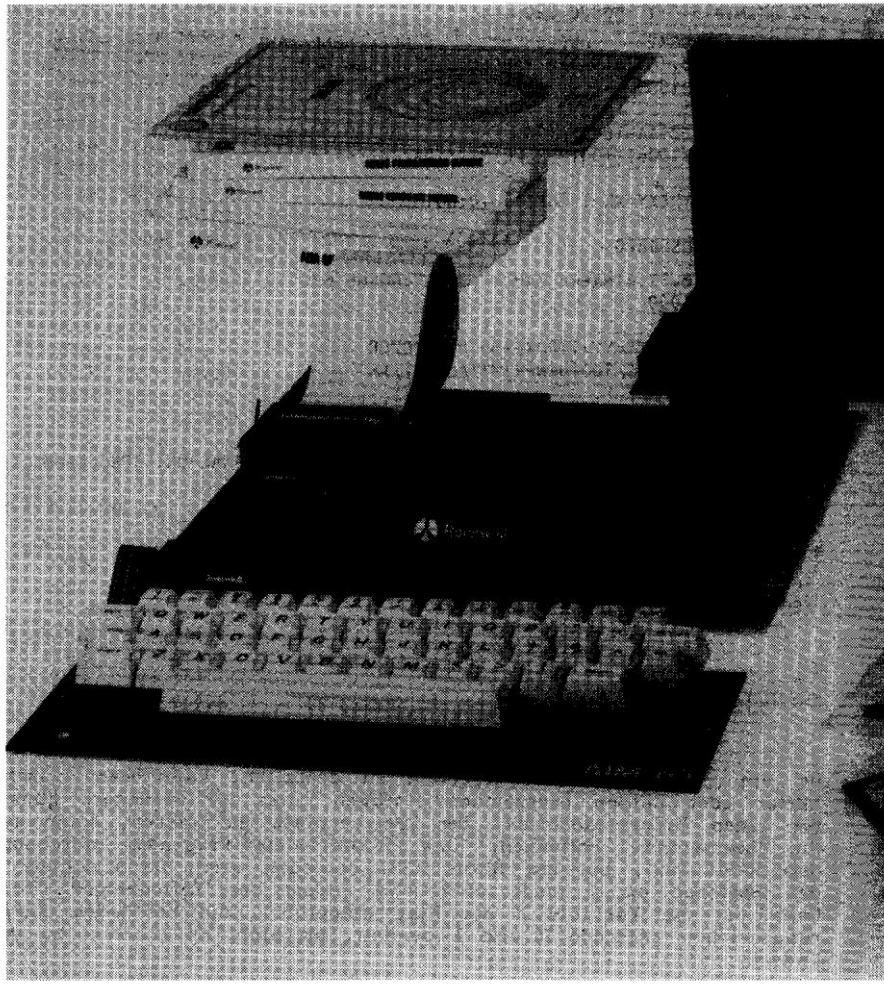
Karlheinz Lehner, Eschborn

Blank-Deleter

Um ein Programm besser leserlich zu gestalten, kann man zwischen Wörter oder Zahlen Leerzeichen einfügen. Dies ist besonders günstig für einen Druckerausdruck. Um Speicherplatz zu sparen, kann man die Leerzeichen entfernen. Das Blank-Deleter-Programm durchsucht dem BASIC-Text, bis ein Blank (32) gefunden wird. Dann schiebt es den Speicher um 1 nach unten und ändert den Zeilenzeiger (PET). Dabei ist die Variable F als Flag eingeführt, die bei jedem Anführungszeichen zwischen 0 und 1 pendelt, um nicht Ausgabetext zu verändern.

```
 1 REM BLANK DELETER VON K. LEHNER
 2 DEFFNA(X)=PEEK(X)+256*PEEK(X+1)
 4 A=1025:FORI=1TO12:A=FNA(A):NEXT
 6 Z=A+4:S=S+C:C=0:F=0
 8 IF PEEK(Z)=32ANDF=0 THENC=C+1:FOR I=Z+1TOFNA(124):X=PEEK (I):
 POKEI-1,X:NEXT:GOTO12
10 Z=Z+1
12 IF PEEK (Z)=34THENF=1-F
14 IF PEEK(Z)<>0 THEN 8
16 Z=A
```

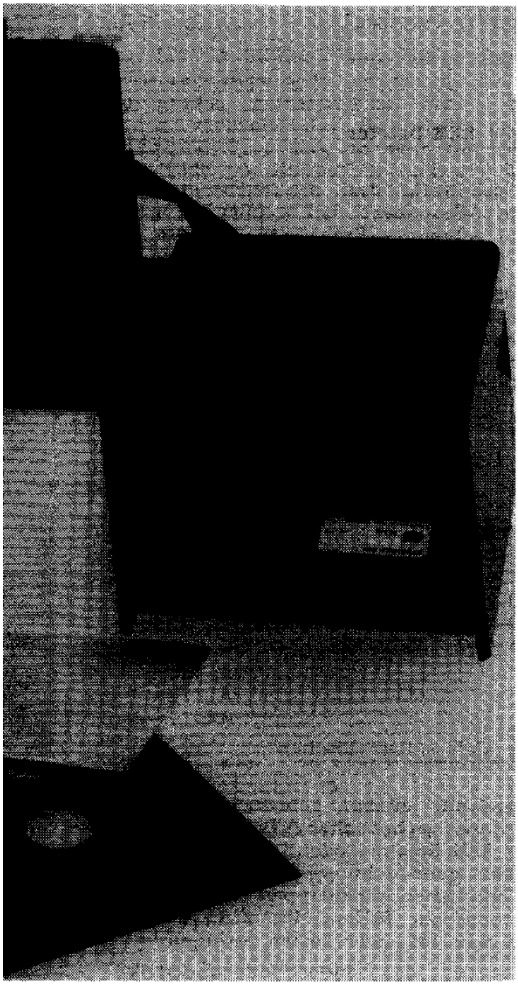
MSB bietet jetzt I



M.&R. Nedela
Tel. 07544 3575

MSB Mic
Sho
Boo

ppy zum AIM-65 an



COMPAS–
Floppy Disk Controller
DM 1050,–
(inkl. DOS auf EPROM)

Rockwell Motherboard
DM 450,–

AIM–65
1 k–RAM DM 910,–

AIM–65
4 k–RAM DM 1090,–

DAM–Floppy
2 Laufwerke DM 1998,–
(220 Volt)

Alle Preise
+ 13% MWSt

ee Postfach 1122 D-7778 Markdorf

```
18 GOSUB24:Z=FNA(Z):IF FNA(Z)<>0 THEN 18
20 A=FNA(A):IF FNA(A)<>0 THEN 6
22 Z=124:C=C+S:GOSUB 24:CLR:GOTO28:REM BZW. RUN 28
24 IF PEEK(Z)-C>0 THEN POKEZ,PEEK(Z)-C:RETURN
26 POKEZ,256+PEEK(Z)-C:POKE Z+1,PEEK(Z+1)-1:RETURN
28 HIER BEGINNT DAS HAUPTPROGRAMM
```

Verbesserter REM-Deleter

Der erste REM-Deleter hatte den Nachteil, daß die Suche nach jeder REM-Löschung beim Anfang begann. Besonders bei großen Programmen ist die Suchzeit unnötig lang. Dazu folgendes Programm, das die gegenwärtig gelöschte Zeile im Speicher 900/901 zwischenspeichert.

Übrigens: Ändert man im REM-Deleter in Zeile 8 die Zahl 143 zu 131, hat man einen DATA-Deleter. Man kann die DATA-Daten lesen, sie abspeichern und dann die Zeilen löschen. Natürlich muß man die Daten mit POKE speichern, denn eine Programmänderung löscht alle Variablen. So hat man den Vorteil, Daten mit dem Programm speichern zu können (keine Files notwendig) und kann den Speicher günstiger ausnutzen.

```
 1 REM-ENTFERNER VON K. LEHNER
 2 POKE 900,1:POKE 901,4
 4 DEFFNA(X)=PEEK(X)+256*PEEK(X+1):Z=FNA(900)
 6 IF PEEK(Z)=0 THEN IF PEEK(Z+1)=0 THEN RUN 16
 8 IF PEEK(Z+4)<>143 THENZ=FNA(Z):GOTO 6
10 POKE901,Z/256:X=PEEK(901):POKE900,Z-256*X
12 PRINT CHR$(147)FNA(Z+2):PRINT"RUN4"
14 POKE527,19:POKE 528,13:POKE529,13:POKE 525,3:END
16 HIER BEGINNT DAS HAUPTPROGRAMM
```

Programme, die sich selber löschen

Es gibt Teile von Programmen, die nur einmal gebraucht werden und die hernach nur unnütz Speicherplatz verbrauchen. Dazu gehören Vorbereitungsroutinen wie z.B. der REM-Deleter oder der Blank-Deleter, die nach getaner Arbeit nicht mehr gebraucht werden. Um sie zu löschen, braucht man nur die Zeilennummer einzutippen und die Tastaturpuffer mit einem "Carriage Return" zu füllen. Um z.B. das Blank-Deleter-Programm zu löschen, kann man folgende Zeilen anfügen:

```
28 PRINT"cs";:FORI=2TO16STEP2:PRINTI:NEXT:PRINT"RUN"30":GOTO32
30PRINT"cs";:FORI=18TO32STEP2:PRINTI:NEXT:PRINT"RUN34"
32 POKE527,19:FORI=528TO536:POKEI,13:NEXT:POKE525,10:END
34 Hier beginnt das Hauptprogramm
```

Rolf Schöne, Leonrodstr. 66, 8000 München 19

BASIC 'DATA'-Generator

Einleitung

Für das Anbinden von Unterprogrammen in Maschinensprache an BASIC-Programme gibt es neben der trivialen Möglichkeit, das Maschinenprogramm getrennt zu laden noch weitere Verfahren. Darunter ist das verbreitetste, das nach dem Assemblieren im Hexadezinalcode vorliegende Programm in den für den BASIC-Interpreter verständlichen Dezimalcode zu übersetzen und als DATA-Anweisung dem BASIC-Programm mitzugeben. Dieses hat dann mit POKE-Anweisungen den Datensatz in den Speicher zu laden, um ihn wieder zu einem lauffähigen Programm zu machen.

Die mühsame Übersetzungsarbeit kann uns der Rechner abnehmen. Wir weisen die einzelnen Bytes des Maschinenprogrammes den DATA-Anweisungen in ihrer ursprünglichen hexadezimalen Form zu und erledigendas Übersetzen in BASIC während des Ladens.

Beispiel:

```
 :
9000 REM  ***   MASCHINENPROGRAMM (HEX)
9010 DATA 200,2AF                 : REM START- UND ENDADR.
9020 DATA 8,48,8A,48,98,48        : REM PROGRAMM
9030 DATA AD,86,F6,A2,F,BD,...
 :
 :
9800 REM  ***   MASCHINENPROGRAMM LADEN (UP)
9810 READ H$ : GOSUB 9900 : S=D : REM STARTADR.
9820 READ H$ : GOSUB 9900 : E=D : REM ENDADR.
9830 FOR A=S TO E
9840 READ H$ : GOSUB 9900
9850 POKE A,D
9860 NEXT A
9870 RETURN
9880 REM
9900 REM  ***   HEX/DEZ -KONVERTIERUNG (UP)
9910 D=0 : L=LEN(H$)
9920 FOR I=L TO 1 STEP -1 : H=ASC(MID$(H$,I,1))
9930 IF H<58 AND H>47 THEN H=H-48
9940 IF H<71 AND H>64 THEN H=H-55
9950 D=D+H*16^(L-I)
9960 NEXT I
9970 RETURN
 :
```

Es bleibt jetz noch immer das lästige Abschreiben des Assembler-Listings und seine Überführung in DATA-Anweisungen. Da diese Tätigkeit streng nach einem gleichbleibenden Schema abläuft und wir schließlich über einen Rechner verfügen, der solche Arbeiten schneller und vor allem fehlerfrei erledigen kann, kommen wir zu folgender

Alternative

Wir erzeugen aud dem Objekt-Code mit Hilfe eines Generatorprogramms

direkt den BASIC-Quellcode. Der Generator hat dabei folgendes zu leisten:

a) Erzeugung einer frei wählbaren Zeilenummer für BASIC

b) Erzeugung eines frei wählbaren Zeileninkrements

c) Erzeugung des Zeilenformates

d) Übergabe von Start- und Endadresse in dezimaler Darstellung

e) dezimale Darstellung des hexadezimalen Objekt-Codes als DATA.

Durch die Forderungen d) und e) erübrigt sich dann auch das Übersetzungsprogramm in BASIC. - Ergänzend zu obigem Pflichtenkatalog sei noch bemerkt, daß Zeilennummer und -inkrement dezimal, Adressen dagegen hexadezimal vorliegen. Der für 6502-Systeme weit verbreitete BASIC-Interpreter von MICROSOFT, auf den der Verfasser sich hier bezieht, stellt die Zeilennummer jedoch hexadezimal dar. Daraus ergibt sich für den Generator als zusätzliche Forderung neben einem

f) HEX/DEZ-Konverter

ein Unterprogramm für

g) DEZ/HEX-Konversion

und Treiberprogramme für

h) Ein-/Ausgabe (Tastatur und Display).

Und um das Maß voll zu machen und das BASIC-Programm nicht über Gebühr zu belasten, wünschen wir uns ein Generatorprogramm

i) in Assembler- bzw. Maschinensprache,

das, wenn es sich bewährt , auch wert ist (aber sicher!), in einem EPROM dauerhaft gespeichert zu werden.

Erkenntnisse

Vor der Erstellung des Programms müssen wir uns Klarheit über das Format einer BASIC-Zeile allgemein und einer DATA-Zeile im besonderen verschaffen. Beim Herumsuchen im entsprechenden RAM-Bereich und in der Zero Page fällt uns auf:

1) Eine BASIC-Zeile beginnt mit einem Zeiger (HEX: LO, HI), der auf das Ende dieser Zeile deutet.

2) Danach folgt die Zeilennummer (HEX: LO, HI) und dann

3) der Inhalt der Zeile, wobei BASIC-Anweisungen bzw. BASIC-Operationen irgendwie codiert und der Rest (Variablen, Text) im ASCII-Code gespeichert sind

4) Insbesondere wird die Anweisung DATA als hex 83 dargestellt (systemabhängig).

5) Das Zeilenende wird durch 00 markiert.

6) Das Ende des Files wird durch weitere 00,00 bezeichnet.

7) In der Zero Page finden wir Speicherplätze, deren Inhalt gleich der Adresse des Zeilenbeginns der letzten Zeile und der Adresse des Fileendes ist.

Programm

Im Interpreter selbst sind zahlreiche Unterprogramme enthalten, die sich für das nun zu schreibende Programm gut verwenden lassen - wenn man ihre Wirkungsweise und Startadresse kennt. Der PET/CBM-Interpreter scheint in dieser Hinsicht schon weitgehend erforscht zu sein (. MICRO MAG, Heft 11, S. 29). Ein Generator für den PET würde daher in der Hauptsache aus einer Folge von Unterprogrammaufrufen bestehen.

Im allgemeinen Fall werden diese Informationen jedoch nicht zugänglich sein. Man könnte sich natürlich helfen, indem man seinen Interpreter disassembliert und in wochenlanger Arbeit auseinanderfieselt. Abgesehen davon, daß man dabei viel lernen kann und auch sehr viel mehr Informationen gewinnen wird, als im Moment benötigt werden, wird sich ohne zwingenden Grund wohl niemand freiwillig an diese Arbeit machen wollen. Wir brauchen ja lediglich schnell ein Werkzeug zur Erstellung von Anwendungsprogrammen. Schreiben wir also alles selbst! ...

Das fertige Programm ist im Listing gezeigt. Die Speicherplätze LZA ($AA, $AB) und MEMH ($7B, 7C ff.) sowie die Adresse 'BASIC' (0000) gelten für das 8k MICROSOFT BASIC, wie es die Firma MCS für ihr System BETA 8 (oder auch einen aufgemöbelten ALPHA 1) als MCS 6502 RESIDENT BASIC V1 REV 3.3 implementiert hat. Für andere Systeme können diese Adressen nach dem BASIC-Kaltstart und Eingeben zweier Zeilen durch PEEK-Anweisungen oder über den Monitor leicht gefunden werden, wie auch der Code (Token) für DATA (s. Listing, Zeile 1010). Die Adresen "VON" ... "INKR" liegen am besten irgendwo im Zeilenpuffer des Interpreters, wo sie ihn nicht stören können.

CHIN, CHOUT und CRLF sind Einsprungsadressen von Monitor-Unterprogrammen. Der Transfer der ASCII-Zeichen geschieht über den Akkumulator (z.B. beim KIM: GETCH, $1E5A; OUTCH, $1EA0; CRLF, $1E2F). HEXA (beim KIM: PACK, $1FAC) formt ASCII-Zeichen 0 ... F (im Akku) in Halbbytes um und schiebt diese paarweise durch INL und INL+1 (KIM: $F8 und $F9).

Die Funktion des Programms ergibt sich schon durch die Aufgabenstellung und wird hoffentlich durch den Kommentar im Listing deutlich. - Wer schon entsprechende Utility-Programme für String-Ausdruck und Zahlensystem-Konversion im EPROM hat, soll das Generatorprogramm ruhig ändern, es ist (mit Klimmzügen) in einer Page unterzubringen.

Anwendung

Das Programm wird entweder über den Monitor oder in BASIC mit einer SYS-Anweisung gestartet (hier auf $B000). Im letzteren Fall sollte die Zeile 940 durch RTS ersetzt werden. Wichtig ist, daß der BASIC-Interpreter vor Programmstart schon einmal initialisiert wurde, um die Zeigerinhalte von MEMH und LZA übernehmen zu können.

```
0010  0200          ;-----------------------------------------;
0020  0200          ;                                         ;
0030  0200          ;  GENERATOR FUER 'DATA'-ZEILEN IN BASIC  ;
0040  0200          ;                                         ;
0050  0200          ;          R. Schoene,  18.05.80          ;
0060  0200          ;-----------------------------------------;
0070  0200
```

```
0080  0200                MEMH   =$7B              ;BASIC FILE ZEIGER
0090  0200                LZA    =$AA              ;ADR. DER LETZTEN ZEILE
0100  0200
0110  0200                       *=$10             ;IM BASIC-LINEBUFFER
0120  0010                VON    *=*+2             ;RAM STARTADRESSE, ZEIGER
0130  0012                BIS    *=*+2             ;RAM ENDADRESSE
0140  0014                HEX    *=*+2             ;HEX 0...FFFF; LSB,MSB
0150  0016                BCD    *=*+3             ;DEZ 0...65535; LSB,HSB,MSB
0160  0019                OLP    *=*+2             ;ADRESSE DES ZEILENENDES
0170  001B                LINE   *=*+2             ;ZEILENNUMMER
0180  001D                PTR    *=*+2             ;AKTUELLER ZEIGER
0190  001F                INKR   *=*               ;ZEILENINKREMENT
0200  001F
0210  001F                BASIC  =0                ;BASIC WARMSTARTADRESSE
0220  001F                CHIN   =$EFA2            ;ASCII EINGABE
0230  001F                CHOUT  =$EFC9            ;ASCII AUSGABE
0240  001F                CRLF   =$EFF2            ;WAGENRUECKLAUF AUSGABE
0250  001F                HEXA   =$FA2D            ;ASCII-->HEX KONVERTER
0260  001F                INL    =$F1              ;EINGABEBUFFER FUER "HEXA"
0270  001F
0280  001F                       *=$B000
0290  B000
0300  B000  D8            GENDAT CLD
0310  B001  20 F2 EF             JSR CRLF
0320  B004  20 D9 B1             JSR STRING        ;TEXT-STRING DRUCKEN
0330  B007  56 4F                .BYTE 'VON $',0   ;RAM STARTADRESSE
0330  B00C  00
0340  B00D  20 59 B1             JSR EING
0350  B010  A5 F1                LDA INL
0360  B012  85 10                STA VON
0370  B014  A5 F2                LDA INL+1
0380  B016  85 11                STA VON+1
0390  B018  20 D9 B1             JSR STRING
0400  B01B  42 49                .BYTE 'BIS $',0   ;RAM ENDADRESSE
0400  B020  00
0410  B021  20 59 B1             JSR EING
0420  B024  A5 F1                LDA INL
0430  B026  85 12                STA BIS
0440  B028  A5 F2                LDA INL+1
0450  B02A  85 13                STA BIS+1
0460  B02C  20 D9 B1             JSR STRING
0470  B02F  49 4E                .BYTE 'INKR ',0   ;ZEILENINKREMENT
0470  B034  00
0480  B035  20 59 B1             JSR EING
0490  B038  20 6F B1             JSR DEZHEX        ;DEZ/HEX-KONVERSION
0500  B03B  A5 1B                LDA LINE
0510  B03D  85 1F                STA INKR
0520  B03F  20 D9 B1             JSR STRING
0530  B042  5A 2E                .BYTE 'Z.NR ',0   ;ERSTE ZEILENNUMMER
0530  B047  00
0540  B048  20 59 B1             JSR EING
0550  B04B  20 6F B1             JSR DEZHEX        ;DEZ/HEX-KONVERSION
0560  B04E
0570  B04E  20 9A B0             JSR KOPF          ;1. DATA-ZEILE BEGINNEN
0580  B051  A5 10                LDA VON
0590  B053  85 14                STA HEX           ;1. DATA : STARTADR.
0600  B055  A5 11                LDA VON+1
0610  B057  85 15                STA HEX+1
```

```
0620  B059  20 D4 B0          JSR DEZDIG       ;DEZ. IN DEN FILE
0630  B05C  20 17 B1          JSR KPTR         ;KOMMA EINFUEGEN
0640  B05F  A5 12             LDA BIS
0650  B061  85 14             STA HEX          ;2. DATA : ENDADR.
0660  B063  A5 13             LDA BIS+1
0670  B065  85 15             STA HEX+1
0680  B067  20 D4 B0          JSR DEZDIG       ;IN DEN FILE
0690  B06A  20 22 B1          JSR ZEND         ;1. DATA-ZEILE BEENDEN
0700  B06D
0710  B06D  20 9A B0    NZEILE JSR KOPF        ;DATA-ZEILE BEGINNEN
0720  B070
0730  B070  20 BC B0    NZEICH JSR DATA        ;DATA IN DEN FILE
0740  B073
0750  B073  A5 10             LDA VON          ;ADRESSENVERGLEICH
0760  B075  C5 12             CMP BIS
0770  B077  90 06             BCC INVON
0780  B079  A5 11             LDA VON+1
0790  B07B  C5 13             CMP BIS+1
0800  B07D  F0 15             BEQ FILEND       ;FERTIG
0810  B07F
0820  B07F  E6 10       INVON  INC VON          ;ZEIGER INKREM.
0830  B081  D0 02             BNE KOMMA
0840  B083  E6 11             INC VON+1
0850  B085
0860  B085  CA          KOMMA  DEX              ;DATA-ZAEHLER DEKREM.
0870  B086  F0 06             BEQ KKOM
0880  B088  20 17 B1          JSR KPTR         ;KOMMA EINFUEGEN
0890  B08B  4C 70 B0          JMP NZEICH       ;NAECHSTES DATA
0900  B08E  20 22 B1    KKOM  JSR ZEND         ;ZEILE ABSCHLIESSEN
0910  B091  4C 6D B0          JMP NZEILE       ;NEUE ZEILE BEGINNEN
0920  B094
0930  B094  20 22 B1    FILEND JSR ZEND        ;FILE ABSCHLIESSEN UND
0940  B097  4C 00 00          JMP BASIC        ;ZURUECK ZUM INTERPRETER
0970  B09A              ;  U N T E R P R O G R A M M E
0980  B09A
0990  B09A  A0 00       KOPF   LDY #0           ;ZEILENKOPF
1000  B09C  20 F7 B0          JSR DMEMH        ;ZEIGER DEKREM.
1010  B09F  A5 1B             LDA LINE
1020  B0A1  91 7B             STA (MEMH),Y
1030  B0A3  85 AA             STA LZA
1040  B0A5  A5 1C             LDA LINE+1
1050  B0A7  C8                INY
1060  B0A8  91 7B             STA (MEMH),Y
1070  B0AA  88                DEY
1080  B0AB  85 AB             STA LZA+1
1090  B0AD  20 07 B1          JSR IMEMH        ;AKTUELLE ADR. ERZEUGEN
1100  B0B0  A2 10             LDX #16          ;16 DATA PRO 'DATA'-ZEILE
1110  B0B2  A9 83             LDA #$83         ;"DATA"-CODE
1120  B0B4  20 19 B1          JSR ZPTR         ;FILE AUFBAUEN
1130  B0B7  A9 20             LDA #$20         ;"SPACE"
1140  B0B9  4C 19 B1          JMP ZPTR         ;UND 'RTS'
1150  B0BC
1160  B0BC  B1 10       DATA   LDA (VON),Y     ;DATA IN DEN FILE
1170  B0BE  85 14             STA HEX          ;HEX
1180  B0C0  20 8F B1          JSR HEXDEZ       ;WIRD DEZ
1190  B0C3  A5 17             LDA BCD+1
1200  B0C5  F0 03             BEQ FUEHR0
1210  B0C7  20 E8 B0          JSR BRECHT       ;HUNDERTER
```

```
1220  B0CA  A5 16     FUEHR0 LDA BCD
1230  B0CC  20 F0 B0         JSR BLINKS      ;ZEHNER
1240  B0CF  A5 16            LDA BCD
1250  B0D1  4C E8 B0         JMP BRECHT      ;EINER, 'RTS'
1260  B0D4
1270  B0D4  20 8F B1  DEZDIG JSR HEXDEZ      ;ADR. KONVERTIEREN
1280  B0D7  A2 02            LDX #2
1290  B0D9  20 E6 B0         JSR DIGITL      ;ZEHNTAUSENDER
1300  B0DC  CA               DEX
1310  B0DD  20 E1 B0         JSR DIGITH      ;TAUSENDER, HUNDERTER
1320  B0E0  CA               DEX
1330  B0E1  B5 16     DIGITH LDA BCD,X
1340  B0E3  20 F0 B0         JSR BLINKS
1350  B0E6  B5 16     DIGITL LDA BCD,X
1360  B0E8  29 0F     BRECHT AND #$0F        ;RECHTE BYTE-HAELFTE
1370  B0EA  18        ASCPTR CLC             ;BCD WIRD ASCII
1380  B0EB  69 30            ADC #$30
1390  B0ED  4C 19 B1         JMP ZPTR        ;'RTS'
1400  B0F0
1410  B0F0  4A         BLINKS LSR A          ;LINKE BYTE-HAELFTE
1420  B0F1  4A               LSR A
1430  B0F2  4A               LSR A
1440  B0F3  4A               LSR A
1450  B0F4  4C EA B0         JMP ASCPTR      ;'RTS'
1460  B0F7
1470  B0F7  A5 7B     DMEMH  LDA MEMH        ;FILE ZEIGER DEKREM.
1480  B0F9  38               SEC
1490  B0FA  E9 02            SBC #2
1500  B0FC  85 19            STA OLP         ;PLATZ FUER ZEILENENDADR.
1510  B0FE  A5 7C            LDA MEMH+1
1520  B100  85 1A            STA OLP+1
1530  B102  B0 02            BCS R1
1540  B104  C6 1A            DEC OLP+1
1550  B106  60        R1     RTS
1560  B107
1570  B107  A5 7B     IMEMH  LDA MEMH        ;AKT. FILE-ZEIGER
1580  B109  18               CLC             ;INKREMENTIEREN
1590  B10A  69 02            ADC #2
1600  B10C  85 1D            STA PTR
1610  B10E  A5 7C            LDA MEMH+1
1620  B110  85 1E            STA PTR+1
1630  B112  90 02            BCC R2
1640  B114  E6 1E            INC PTR+1
1650  B116  60        R2     RTS
1660  B117
1670  B117  A9 2C     KPTR   LDA #','        ;KOMMA BZW.
1680  B119  91 1D     ZPTR   STA (PTR),Y     ;ZEICHEN IN DEN FILE
1690  B11B  E6 1D            INC PTR         ;ZEIGER
1700  B11D  D0 02            BNE R3          ;INKREMENTIEREN
1710  B11F  E6 1E            INC PTR+1
1720  B121  60        R3     RTS
1730  B122
1740  B122  A9 00     ZEND   LDA #0          ;ZEILE BEENDEN
1750  B124  20 19 B1         JSR ZPTR
1760  B127  A5 1B            LDA LINE        ;ZEILENZAEHLER
1770  B129  18               CLC             ;INKREMENTIEREN
1780  B12A  65 1F            ADC INKR
1790  B12C  85 1B            STA LINE
```

```
1800  B12E  90 02           BCC LL
1810  B130  E6 1C           INC LINE+1
1820  B132  A5 1D     LL    LDA PTR          ;ZEILENENDADRESSE IN
1830  B134  91 19           STA (OLP),Y      ;ZEILENKOPF SCHREIBEN
1840  B136  85 AA           STA LZA
1850  B138  A5 1E           LDA PTR+1
1860  B13A  C8              INY
1870  B13B  91 19           STA (OLP),Y
1880  B13D  88              DEY              ;VIELLEICHT
1890  B13E  85 AB           STA LZA+1        ;LETZTE ZEILE
1900  B140  A9 00           LDA #0           ;UND
1910  B142  20 19 B1        JSR ZPTR         ;FILE-ENDE
1920  B145  20 19 B1        JSR ZPTR
1930  B148  A5 1D           LDA PTR          ;ZEIGER AKTUALISIEREN
1940  B14A  85 7B           STA MEMH
1950  B14C  85 7D           STA MEMH+2
1960  B14E  85 7F           STA MEMH+4
1970  B150  A5 1E           LDA PTR+1
1980  B152  85 7C           STA MEMH+1
1990  B154  85 7E           STA MEMH+3
2000  B156  85 80           STA MEMH+5
2010  B158  60              RTS
2020  B159
2030  B159  A9 00     EING  LDA #0           ;NUMERISCHE EINGABE
2040  B15B  85 F1           STA INL          ;EINGABEBUFFER
2050  B15D  85 F2           STA INL+1        ;VORBEREITEN
2060  B15F  20 A2 EF  INP   JSR CHIN         ;ZEICHEN HOLEN
2070  B162  C9 0D           CMP #$0D         ;WENN 'CR'
2080  B164  F0 06           BEQ R4           ;DANN EINGABEENDE
2090  B166  20 2D FA        JSR HEXA         ;SONST BYTE
2100  B169  4C 5F B1        JMP INP          ;AUFBAUEN
2110  B16C  4C F2 EF  R4 .  JMP CRLF         ;'RTS'
2120  B16F
2130  B16F  F8        DEZHEX SED             ;DEZIMAL/HEXADEZIMAL
2140  B170  A9 FF           LDA #$FF         ;-KONVERSION
2150  B172  85 1B           STA LINE         ;FUER
2160  B174  85 1C           STA LINE+1       ;ZEILENNUMMER-
2170  B176  E6 1B     DH1   INC LINE         ;UND -INKREMENT
2180  B178  D0 02           BNE DH2          ;EINGABE
2190  B17A  E6 1C           INC LINE+1
2200  B17C  A2 00     DH2   LDX #0
2210  B17E  A0 01           LDY #1
2220  B180  18              CLC
2230  B181  B5 F1     DH3   LDA INL,X
2240  B183  E9 00           SBC #0
2250  B185  95 F1           STA INL,X
2260  B187  E8              INX
2270  B188  88              DEY
2280  B189  10 F6           BPL DH3
2290  B188  B0 E9           BCS DH1
2300  B18D  D8              CLD
2310  B18E  60              RTS
2320  B18F
2330  B18F  86 F1     HEXDEZ STX INL         ;HEXADEZIMAL/DEZIMAL
2340  B191  A2 01           LDX #1           ;-KONVERSION
2350  B193  B5 14     HCOMPL LDA HEX,X       ;FUER DATA
2360  B195  49 FF           EOR #$FF
2370  B197  95 14           STA HEX,X        ;HEX KOMPLEMENT
```

```
2380  B199  CA          DEX
2390  B19A  10 F7       BPL HCOMPL
2400  B19C  F8          SED
2410  B19D  A2 02       LDX #2
2420  B19F  A9 99       LDA #$99
2430  B1A1  95 16  DCOMPL STA BCD,X      ;DEZ KOMPLEMENT
2440  B1A3  CA          DEX
2450  B1A4  10 FB       BPL DCOMPL
2460  B1A6  38    DEZ1  SEC
2470  B1A7  A0 02       LDY #2
2480  B1A9  A2 00       LDX #0
2490  B1AB  B5 16  DEZ2  LDA BCD,X
2500  B1AD  69 00       ADC #0
2510  B1AF  95 16       STA BCD,X
2520  B1B1  E8          INX
2530  B1B2  88          DEY
2540  B1B3  10 F6       BPL DEZ2
2550  B1B5  E6 14       INC HEX
2560  B1B7  D0 ED       BNE DEZ1
2570  B1B9  E6 15  DEZ3  INC HEX+1
2580  B1BB  F0 16       BEQ DEZ4
2590  B1BD  18          CLC             ;UEBERTRAG
2600  B1BE  A5 16       LDA BCD         ;256 ADDIEREN
2610  B1C0  69 56       ADC #$56
2620  B1C2  85 16       STA BCD
2630  B1C4  A5 17       LDA BCD+1
2640  B1C6  69 02       ADC #2
2650  B1C8  85 17       STA BCD+1
2660  B1CA  A5 18       LDA BCD+2
2670  B1CC  69 00       ADC #0
2680  B1CE  85 18       STA BCD+2
2690  B1D0  4C B9 B1    JMP DEZ3
2700  B1D3  A0 00  DEZ4  LDY #0
2710  B1D5  A6 F1       LDX INL
2720  B1D7  D8          CLD
2730  B1D8  60          RTS
2740  B1D9
2750  B1D9  68    STRING PLA             ;TEXT-STRING AUSGABE
2760  B1DA  85 14       STA HEX
2770  B1DC  68          PLA             ;STRING-ADR.
2780  B1DD  85 15       STA HEX+1
2790  B1DF  A0 00       LDY #0
2800  B1E1  C8    TEXT  INY
2810  B1E2  98          TYA
2820  B1E3  48          PHA
2830  B1E4  B1 14       LDA (HEX),Y     ;AUSGABE BEENDEN
2840  B1E6  F0 07       BEQ RAUS        ;BEI '00', SONST
2850  B1E8  20 C9 EF    JSR CHOUT       ;TEXT AUSGEBEN
2860  B1EB  68          PLA
2870  B1EC  A8          TAY
2880  B1ED  D0 F2       BNE TEXT
2890  B1EF  68    RAUS  PLA             ;RUECKSPRUNGADR.
2900  B1F0  98          TYA             ;BERECHNEN
2910  B1F1  38          SEC
2920  B1F2  65 14       ADC HEX
2930  B1F4  85 14       STA HEX
2940  B1F6  90 02       BCC R5
2950  B1F8  E6 15       INC HEX+1
```

Für die Editierung auf dem Bildschirm stehen zusätzliche Funktionen zur Verfügung: Insert/Delete Line, Scroll up, Scroll down, Erase Begin (von Zeilenanfang bis einschl. Cursor-Stellung), Erase to end of line, Bell (Piepser) vor Zeilenende, Definition eines Bildschirmfensters. Daneben gibt es für den Bildschirm den Text- und den Graphic-Mode. Bei letzterem schließen die Bildzeilen unmittelbar aneinander an, bei ersterem ergibt sich eine bessere Lesbarkeit für Texte durch den Zeilenabstand. Das deutliche, ruhige Monitorbild wird durch den Bildschirmcontroller MC6845 CRTC von Motorola erzeugt. Erste Versuche zeigen, daß man durch Beschreiben seiner Register auch beim 8032 den Bildaufbau beeinflussen kann.

Die Kapazität des Festwertspeichers ist jetzt auf 18 kB angewachsen. Dem Benutzer stehen 32 kB RAM-Speicher zur Verfügung. Das neue Bildschirmformat bedingt z.T. neue Verwaltungsroutinen und Speicheradressen für das Betriebssystem. Beim Übergang vom 3032 zum 8032 wird man daher einige Programmstellen überarbeiten müssen, in denen solche maschinenspezifischen Adressen angesprochen worden sind. In privater Initiative wird aber schon jetzt daran gearbeitet, solche Eigenheiten möglichst bald zu dokumentieren.

Das neue Mini Floppy Disk-System CBM 8050 für diese Zentraleinheit wurde über DOS 2.0 mit einem erweiterten Befehlsvorrat und 4 kB RAM-Puffer versehen. Je Laufwerk stehen 77 Spuren mit 23-29 Sektoren zur Verfügung. Je Laufwerk werden 512 kB Zeichen gespeichert, zusammen also 1 Megabyte. Man kann die Sektorenlänge frei definieren und sequentielle Dateien an ein File anfügen.

Die Speichereinheiten CBM 8061/8062 verwenden 8-Zoll IBM-kompatible Disketten (1 bzw. 2 drives), in doppelter Dichte beschrieben, und speichern je Laufwerk 1,5 Megabyte. Auch hier frei wählbare Sektorlänge und direkter Zugriff auf einzelne Bytes der Diskette.

Dis Disketten sind zur Lieferung ab Herbst angekündigt, der RechnerCBM 8032 soll ab Juli 1980 lieferbar sein.

Den neu vorgestellten Geräten ist das gelungene Bestreben Commodores abzulesen, die Systeme mehr und mehr für den professionellen Einsatz aufzurüsten, zum Angebot gehören ja ebenso der Drucker 3022 und der Plotter 3050. Weltweit sollen jetzt über 100.000 PET- und CBM-Systeme verkauft worden sein, und man rechnet mit weiteren Zuwächsen: Ab 1981 will Commodore in Braunschweig für den europäischen Markt produzieren, in der Endstufe bis zu 10.000 Geräte monatlich.

Die große Betreiberzahl zieht ein rasch zunehmendes Angebot an Betriebs- und Anwendersoftware sowie an Interfaces und Information nach sich, so daß der CBM-Betreiber heute und künftig viel schneller zu Lösungen kommen kann als früher, da ihm nur knappe Handbücher zur Verfügung standen.

Viele Betreiber besitzen noch nicht die aktuellen Commodore-Handbücher. Daher folgende Hinweise: 'CBM 2001/3001 Bedienungshandbuch' in deutscher Sprache, 137 Seiten, sorgfältig bearbeitet und editiert. Daneben als weitgehend inhaltsgleiche englische Vorlage: 'CBM User Manual', ca. 165 Seiten, für die Modelle 2001-16 und die Serie 3000. Für die bisherige Floppy 2040/3040 gibt es das 'CBM Floppy Disk User Manual', ca. 60 Seiten. Eine deutsche Übersetzung soll in Arbeit sein.

```
2960  B1FA  A0 00       R5     LDY #0          ;RUECKSPRUNG NACH
2970  B1FC  6C 14 00           JMP (HEX)       ;TEXTENDE +1
2980  B1FF
2990  B1FF              ZZZ    .END

)000 FEHLER
3YMBOLE:
ASCPTR B0EA     BASIC  0000     BCD    0016     BIS    0012
BLINKS B0F0     BRECHT B0E8     CHIN   EFA2     CHOUT  EFC9
CRLF   EFF2     DATA   B0BC     DCOMPL B1A1     DEZ1   B1A6
DEZ2   B1AB     DEZ3   B1B9     DEZ4   B1D3     DEZDIG B0D4
DEZHEX B16F     DH1    B176     DH2    B17C     DH3    B181
DIGITH B0E1     DIGITL B0E6     DMEMH  B0F7     EING   B159
FILEND B094     FUEHR0 B0CA     GENDAT B000     HCOMPL B193
HEX    0014     HEXA   FA2D     HEXDEZ B18F     IMEMH  B107
INKR   001F     INL    00F1     INP    B15F     INVON  B07F
KKOM   B08E     KOMMA  B085     KOPF   B09A     KPTR   B117
LINE   001B     LL     B132     LZA    00AA     MEMH   007B
NZEICH B070     NZEILE B06D     OLP    0019     PTR    001D
R1     B106     R2     B116     R3     B121     R4     B16C
R5     B1FA     RAUS   B1EF     STRING B1D9     TEXT   B1E1
VON    0010     ZEND   B122     ZPTR   B119     ZZZ    B1FF
```

```
VON $B000
BIS $B1FE
INKR 10
Z.NR 9000

OK
LIST
```

PROBELAUF

DIE ERSTEN ZEILEN DES GENERATORPROGRAMMES SIND IN BASIC-DATA-ZEILEN UMGESETZT WORDEN, NATÜRLICH MIT DEZIMALCODE.

```
 9000 DATA 45056,45566
 9010 DATA 216,32,242,239,32,217,177,86,79,78,32,36,00,32,89,177
 9020 DATA 165,241,133,16,165,242,133,17,32,217,177,66,73,83,32,36
 9030 DATA 00,32,89,177,165,241,133,18,165,242,133,19,32,217,177,73
```

Die neue CBM-Serie 8000

Auf der Hannover-Messe stellte Commodore seine neue Zentraleinheit CBM 8032 vor und kündigte die Disketten CBM 8050 und 8061/8062 an.

Hervorstechendes Merkmal das 8032 ist der grüne 31-cm-Bildschirm, der nunmehr 2000 Zeichen im Format 25 Zeilen x 80 Zeichen darstellt.Die neue Bildröhre machte ein verändertes Design des Gehäuses nötig. Dazu kommt das um 12 Befehle erweiterte BASIC 4.0. Die neuen Befehle integrieren Leistungen des DOS 1.2, sie sind also Floppy Disk-orientiert. Es handelt sich um APPEND (Anfügen von Daten an eine sequentielle Datei), BACKUP (vollst. Kopieren), COPY, COLLECT (Streichen nicht ordnungsgemäßer Dateien), DOPEN, DSAVE und DLOAD, DIRECTORY, HEADER (formatiere), RECORD (Einstellen auf bestimmte Stelle einer Datei), RESTORE und CONCAT (Verbinden von Dateien).

Dipl.-Math., Dipl.-Wirtsch.-Math. Georg Huber, Bonn

Interruptgetriebene Cassettenein- und -ausgabe (2)

3.6 Die Byte-Timer-Interrupts müssen den Prozessor immer zu denjenigen Abschnitten der E/A-Routine führen, die gemäß der Gliederung des Datensatzes logisch als nächste zur Bearbeitung anstehen. Verfolgen wir kurz den dynamischen Verlauf der Eingabe in Abb. 3:

Der Interrupt-Vektor zeigt ständig auf die Interrupt-Routine. Die Subroutine WEITER im Abschnitt EING lädt die Anfangsadresse des Abschnitts 'Synchronisieren/Eingabe' in den Zellen 000E, 000F. Folglich führt der erste Byte-Timer-Interrupt in den Abschnitt 'Synchronisieren/Eingabe'. Kann der Prozessor nicht synchronisieren, kehrt er über RTS(1) - RTI ins Hauptprogramm zurück (ausgezogene Linie). Alle folgenden Byte-Timer-Interrupts bewirken die gleiche Sprungfolge, solange der Prozessor nicht in Phase liest.

Ist dem Prozessor aber erstmals die Synchronisation gelungen, darf er nicht mit Synchronisationsversuchen fortfahren. Vielmehr muß er die restlichen Synchronisationsbytes überlesen und anschließend die Orientierungsdaten bearbeiten, d.h. den Programmabschnitt 'Orientierung/Eingabe' anspringen, wo diese Anweisungen stehen. Für den Abschnitt ' Synchroniseren/Eingabe' bedeutet das: Ist dem Prozessor erstmals die Synchronisation gelungen, dann kehrt er nicht über RTS(1) - RTI sondern über JSR WEITER - RTS(2) - RTI ins Hauptprogramm zurück (gestrichelte Linie in Abb. 3). Die Subroutine WEITER zieht ihre eigene Rücksprungadresse vom Stack, erhöht sie um 1 und trägt dieses Ergebnis - nämlich die Anfangsadresse des auf die Anweisung JSR WEITER unmittelbar folgenden Abschnittes 'Orientierung/Eingabe' - in die Zellen 000E/000F ein. Beim nächsten Byte-Timer-Interrupt landet der Prozessor also am Anfang des Abschnittes 'Orientierung/Eingabe' usw..

3.7 Dem Programm sind drei Speicherbereiche zugeordnet:

1. Das Programm selbst wird nicht verändert, kann also in ein PROM gebrannt sein; vgl. 3.7.2.
2. Es lagert Parameter und Zwischenergebnisse auf die Zeropage aus; vgl. 3.7.1.
3. Das auszugebende bzw. zu beschreibende D a t e n f e l d liegt im RAM.

3.7.1 Parameter und Zwischenergebnisse belegen die Zeropage-Zellen 000C bis mindestens 0021 (Ausnahme: SYM-Subroutine OUTBTH). Die Eintragung 'A' in der Spalte 'Parameter' zeigt, daß der Anwender diese Zelle vor dem Aufruf der Ausgabe-Routine mit dem passenden Parameterwert geladen haben muß; entsprechend 'E' bei der Eingabe.

ZERO-PAGE-ZELLE	PARA-METER	NAME	INHALT
0C		ALT	ZWISCHENERGEBNIS
0D		ZCHN	DITO
0E		-	ADRESSE DES NÄCHSTEN ANZUSPRINGENDEN
0F		-	ABSCHNITTES
10		PSL	PRÜFSUMME
11		PSH	"
12	A,E	AAL	ANFANGSADRESSE DES DATENBEREICHES
13	A,E	AAH	"

14		EAL	ENDADRESSE DES DATENBEREICHES
15		EAH	"
16		-	ZÄHLVARIABLE
17		-	"
18		-	"
19		STATUS	00 = ABGESCHLOSSEN/ 20 = AUSGABE IN ARBEIT/ 40 = EINGABE IN ARBEIT/ 80 = FEHLERABBRUCH
1A		21 (HEX)	'!' IN ASCII
1B		BZL	BANDZÄHLERSTAND
1C		BZH	"
1D	A,E	IDL	DATEIKENN-NUMMER
1E	A,E	IDH	"
1F	A	N	LÄNGE+2 DES VOM ~NWENDER EINZUTRAGENDEN KOPFS, 1≤N < E0
20	A	KØ	ANZAHL DER DATENBYTES DES SATZES
21	A	K1	"
22	A	K2	KEINE UNMITTELBARE BEDEUTUNG FÜR DIE
..	..	..	E/A-ROUTINE; ANWENDERABHÄNGIGE
20+N	A	KN	INTERPRETATION

Es hängt vom Bedarf des Anwenders ab, was er in die Zellen 22 ff. schreiben will, beispielsweise Angaben über die Datei: Name, Besitzer, Ersteller, Zugri berechtigung, Programm- oder Datenunterscheidung, Abspeicherdatum und dergl..

Die Einleseroutine sieht gegen Ende des Abschnitts 'Kopf/Eingabe' ein JSR STNSPR vor. Zwei Beispiele sollen erläutern, welche Vorteile ein solcher 'Seitensprung' kurz vor der Dateneingabe haben kann:

a) STNSPR überprüft den eben gelesenen Kopf. Sie kann erforderlichenfalls - etwa wegen fehlender Zugriffsberechtigung - die Eingabe abbrechen, noch ehe der Magnetkopf des Cassettenrecorders den Datenabschnitt erreicht hat.

b) STNSPR ermöglicht dynamische RAM-Speicherverwaltung: Für die Eingabe braucht der Anwender als Parameter neben der Dateikennnummer $ID_{L,H}$ nur die Anfangsadresse $AA_{L,H}$ des Datenbereiches zu spezifizieren. Den Umfang der einzulesenden Daten erfährt der Prozessor erst, nachdem er den Kopf, insbesondere $k_{o,1}$ kennt. Reicht $AA_{L,H} + K_{o,1}$ (= Endadresse nach der Dateneintragung ins RAM) in einen unzulässigen RAM-Speicherbereich hinein, kann er augenblicklich $AA_{L,H}$ herabsetzen oder notfalls die Eingabe abbrechen.

```
        ; BEI BYTE-TIMER-INTERRUPT
        ; HIERHER SPRINGEN
KASSEA  BIT 19
        BVS EA1         (FALLS EINGABE)
        LDA BYTETT      BYTE-T: BYTE-TIMER
        STA A805        LADEN, START
EA1     JMP ($E)

        ;BEGINN AUSGABE
AUSG    LDA #$20
        STA STATUS      AUSGABE IN ARBEIT
        LDA #'!
        STA 1A
        JSR STARTE
        LDA #7          SYM: DISPLAY
```

```
        STA A402      SYM: DISPLAY
        LDA #$FF      256 SYNCHR. BYTES
        STA 16
        LDA #$5       5 ORIENT. BYTES
        STA 17
        LDA 24        STM: AUFNAHME EIN BEI
        STA A400      STM: KASS. REKORDER
        JSR WEITER

        ;SYNCHRONISIERUNG, ORIENTIERUNG/
        ;AUSGABE:
        ; 256 SYNCHR. BYTES,!,BZL,BZH,IDL,IDH
        LDA 16
        BEQ SOA1
        LDA #$16      SYNCHRONISATIONSBYTE
        JSR OUTB TH   AUSGEBEN
        DEC 16
        BNE SOA2
        LDA 1         STM: AKTUELLER WERT DES
        STA BZL       BANDZÄHLERS VON 1,2 NACH
        LDA 2         BZL,BZH LADEN
        STA BZH
SOA2    RTS
SOA1    INC 18
        LDX 18
        LDA 19,X      !, BZL, BZH,IDL,IDH
        JSR OUTBH     AUSGEBEN
        LDX 18
        CPX 17
        BNE SOA2
        CLC
        LDA 1F
        ADC #$2
        STA 16
        LDA #$1
        STA 17
        JSR WEITER

        ;KOPF/AUSGABE
        ;N,KØ,K1,..KN,PSL,PSH
        INC 18
        LDX 18
        LDA 1E,X
        JSR PLUSPS
        JSR OUTBTH
        LDX 18
        CPX 16
        BEQ KA1
        RTS
KA1     LDA 17        WENN INHALT VON 17
        BEQ KA2       NICHT Ø IST, DANN DIE
        DEC 17        PRÜFSUMME AN DEN
        LDY #$0       STRING N, ..., KN
KA3     INC 16        HÄNGEN
        INX
        LDA 10,Y
        STA 1E,X
        INY
        CPY #$2
```

```
        BNE KA3
        RTS
KA2     CLC             EA=AA+(KØ,K1)
        LDA 12
        ADC 20
        STA 14
        LDA 13
        ADC 21
        STA 15
        LDA #$0         PRÜFSUMME AUF Ø
        STA PSL         ZURÜCKSTELLEN
        STA PSH
        JSR WEITER

        ; DATEN/AUSGABE
        ;DØ, ...,DM,/ (M=KØK1)
        LDX #$12
        LDY #$14
        JSR XGRGLY      SA<EA2
        BCS DA1         SPRUNG BEI NEIN
        LDY #$0
        LDA (SAL),Y
        JSR PLUSPS
        JSR OUTBTH      DATENBYTE IN (SAL,SAH)
        INC SAL         AUSGEBEN
        BNE DA2         SAL, SAH+1
        INC SAH
DA2     RTS
DA1     LDA #'/
        JSR OUTBTH
        JSR WEITER

        ;SCHLUSS AUSGABE
        ;PSL,PSH
        LDX 18
        LDA 10,X
        JSR OUTBTH
        LDX 18
        BNE SA1
        INC 18
        RTS
SA1     JSR ENDEOK

        ;BEGINN/EINGABE
EING    LDA #$40
        STA STATUS      EINGABE IN ARBEIT
        LDA #'!
        STA 1A
        JSR STARTE
        LDA #$0         BIT-T: ONE SHOT
        STA A003        BIT-T: INTERRUPT DISABLED
        LDA #$E8        BIT-T: LOW LATCH
        STA A004        BIT-T: LADEN
        LDA #$4         STM: WIEDERGABE EIN
        STA A400        STM: BEI KASS.-REKORDER
EING1   JSR WEITER

        ;SYNCHRONISIERUNG/EINGABE
        ;1 SYNCHR.-BYTE
        CLI
```

```
        LDA #$FF    - BYTE-T: BYTE-TIMER
        STA A805              LADEN, STARTEN
SE2     JSR WCHSLS
        BCC SE1
        JSR WCHSLS
SE1     ROR ZCHN
        LDA ZCHN
        CMP #$16      = SYNCHR.-BYTE
        BNE SE2
        SEC
        LDA BYTETT    BYTE-T: INTERRUPT NACH
        SBC #$19              ZEIT ZWISCHEN
        STA A805              ZWEI BYTES
        LDA #$F5      FF-F5=10 (DEZ.) SYNCHR.-
        STA 16        BYTES
        LDA #$4       4+1 ORIENT.-BYTES
        STA 17
        JSR WEITER

        ;ORIENTIERUNG/EINGABE
        ;RESTLICHE SYNCHR.-BYTES (MINDESTENS
        ;10),!, BZL,BZH,IDL,IDH
        JSR ZCHNLS
        LDX 16
        BEQ OE1
        BPL OE2
        CMP #$16      =SYNCHR.-BYTE
        BNE EING1
        INC 16
OE3     RTS
OE1     CMP #$16      =SYNCHR.-BYTE
        BEQ OE3
OE2     INC 16
        STA 1F,X
        CPX 17
        BNE OE3
        LDA 1F
        CMP #'!
        BNE EING1     FALSCHES STARTZEICHEN
        LDA 20        STM: RECHNER-INTERNEN BAND-
        STA 1         ZÄHLER AKTUALISIEREN
        LDA 21
        STA 2
        LDA IDH
        CMP 23
        BNE OE4
        LDA IDL
        CMP 22
        BEQ OE5
OE4     JMP ENDEF     FEHLERAUSGANG: DATEI
        ;             HAT FALSCHE KENN-NR.
OE5     JSR WEITER

        ;KOPF/EINGABE
        ;N,KØ, ...,KN,PSL,PSH
        JRS ZCHNLS
        LDY 18
        BNE KE1
        STA 16
```

```
        INC 16
        LDX #$0
        STX 17
        INC 18
KE1     CPY #$2
        BCS KE2
        LDX 17
        STA 1F,X
        JSR PLUSPS
        LDX 17
        CPX 16
        BNE KE3
        INC 18
KE3     INC 17
        RTS
KE2     BNE KE4
        CMP 10
        BNE KE5
        INC 18
        RTS
KE4     CMP 11
        BEQ KE6
KE5     JMP ENDEF     FEHLERAUSGANG: PRÜF-
        ;             SUMMENFEHLER KOPFTEIL
KE6     CLC
        LDA AAL       EA=AA+(KOK1)
        ADC K0
        STA EAL
        LDA AAH
        ADC K1
        STA EAH
        LDA #$0       PRÜFSUMME FÜR DATEN-
        STA PSL       BYTES AUF Ø SETZEN
        STA PSH
        JSR STNSPR    "SEITENSPRUNG" VOR
        JSR WEITER    DATENEINGABE

        ;DATEN/EINGABE
        ;DØ, ...,DM,(M=KOK1)
        JSR ZCHNLS
        LDX #$12
        LDY #$14
        JSR XGRGLY    AA<EA?
        BCS DE1
        JSR PLUSPS
        LDY #$0
        STA (AAL),Y
        INC AAL
        BNE DE2
        INC AAH
DE2     RTS
DE1     CMP #'/
        BEQ DE3
        JMP ENDEF     FEHLERAUSGANG: DATEN-
        ;             SCHLUSSZEICHEN FALSCH
DE3     JSR WEITER

        ;SCHLUSS/EINGABE
        ;PSL,PSH
```

```
        JSR ZCHNLS
        LDX 18
        BNE SE1
        CMP PSL
        BNE SE2
        INC 18
        RTS
SE1     CMP PSH
        BEQ SE3
SE2     JMP ENDEF       FEHLERAUSGANG: PRÜF-
        ;               SUMMENFEHLER DATENTEIL
SE3     JMP ENDEOK

        ;VORBESETZUNGEN
STARTE  LDA #$0         PRÜFSUMME AUF Ø
        STA PSL         ZURÜCKSTELLEN
        STA PSH
        STA A80B        BYTE-T: ONE SHOT
        STA A804                LOW LATCH=Ø
        LDA BYTETT      BYTE-TIMER STARTEN
        STA A805
        LDA #$C0        ENABLE TIMER-INTERRUPT
        STA A80E
        LDA #$3F        SYM: DISPLAY
        STA A403        SYM: DISPLAY
        RTS

        ; DIE ADRESSE  DER AUF DEN AUFRUF VON
        ;'WEITER' FOLGENDEN ANWEISUNG
        ;IN E,F LADEN
WEITER  LDA #$0         ZÄHLER AUF Ø
        STA 18
        PLA
        STA E
        PLA
        STA F
        INC E
        BNE WTR1
        INC F
WTR1    RTS

        ;A AUF DIE PRÜFSUMME ADDIEREN
PLUSPS  PHA
        CLC
        ADC  PSL
        STA  PSH
        BCC PLPS1
        INC  PSH
PLPS1   PLA
        RTS

        ;C-FLAG SETZEN, FALLS DER INHALT
        ;DER ZERO-PAGE-ZELLEN X,X+1
        ;GRÖSSER/GLEICH DEM INHALT DER
        ;ZERO-PAGE-ZELLEN Y,Y+1 IST.
XGRGLY  PHA
        LDA 1,X
        CMP 1,Y
        BCC XGGY1
        BNE XGGY1
```

```
        LDA Ø,X
        CMP Ø,Y
XGGY1   PLA
        RTS

        ;EIN ZEICHEN VOM BAND IN DEN
        ;AKKUMULATOR LESEN
ZCHNLS  LDX #$8
        JSR WCHSL
        LDA BYTET     BYTE-T: BYTE-TIMER-TAKT
        SEC           AUF KNAPP UNTER
        SBC #$1       SCHREIB-TAKT
        STA A805      STELLEN
ZL3     JSR WCHSL
        BCC ZL1
        JSR WCHSL
        BCC ZL2
        ROR ZCHN
        DEX
        BNE ZL3
        LDA ZCHN
        RTS
ZL2     PLA           FEHLERAUSGANG:
        PLA           UNZULÄSSIGE
        JMP ENDEF     BITDARSTELLUNG

        ;WECHSEL DES LESEBITS
WCHSL   LDY #$FF
WSL1    LDA A000      SYM: LESEBIT
        AND #$40      SYM: BEI
        CMP ALT       SYM: A000/BIT 6
        BEQ WSL1
        STA ALT
        LDA A005      BIT-T:VERSTRICHENE ZEIT
        STY A005      AUF FFE8 STELLEN
        CMP #$FE      IN A IST BIT-ZEIT
        RTS
        ;NACH DEM UMSCHLAG DES LESEBITS LÄUFT
        ;DER BIT-TIMER VON FFE8 (LOW LATCH, VGL.
        ;VORBESETZUNG) ABWÄRTS. ES ERFOLGT DER
        :NÄCHSTE UMSCHLAG INNERHALB WENIGER ALS
        ;492 MIKROSEKUNDEN, WIRD DAS C-FLAG GE-
        ;LÖSCHT (ZEICHNUNG 2, BEREICH A), SONST
        ;GESETZT (BEREICH B)

        ;WECHSEL DES LESEBITS BEI
        ;SYNCHRONISIEREN, FUNKTION
      ;WIE WCHSL, ZUSÄTZLICH ABBRUCH
        ;DES LESEVERSUCHES NACH 1F00
        ;MIKROSEKUNDEN
WCHSLS  LDY #$FF
WSLS2   LDA A805      BYTE-T: AKTUELLER STAND
        CMP #$E0
        BCS WSLS1
        LDA BYTETT    BYTE-T: NEU LADEN
        STA A805      UND STARTEN
        PLA
        PLA
        RTS
WSLS1   LDA A000      SYM: LESEBIT DES
```

```
            AND #$40      KASSETTENINTERFACE
            CMP ALT       BEI A000/BIT 6
            BEQ WSLS2
            STA ALT
            LDA A005      BIT-T: VERSTRICHENE
            STY A005      AUF FFE8 STELLEN
            CMP #$FE      IN A IST BIT-T-ZEIT
            RTS

            ;EINGABE UND AUSGABE ABSCHLIESSEN
      ENDEF LDA #$80      FEHLERENDE
            BNE ENDE1
      ENDEOK LDA #$00     OK-ENDE
      ENDE1 STA STATUS
            LDA #$40      BYTE-T: INTERRUPT
            STA A80E      DISABLE
            RTS

            ;SYM-MONITOR 8F17-8F42:
            ;OUTBTH - NO CLOCK
            ;A, X DESTROYED
            ;MUST RESIDE ON ONE PAGE
            ;TIMING CRITICAL
209   OUTBTH LDX #9       8 BITS + START-BIT
C39A6        STY TEMP2
5FC          STA CHAR
D02A4        LDA TAPOUT   GET PREVIOUS LEVEL
5FC   GETBIT LSR CHAR
908          EOR #TPBIT
D02A4        STA TAPOUT   INVERT LEVEL
      ;HERE STARTS FIRST 416 µSEC PERIOD
047          LDY #TM1500
3     A416   DEY          TIME FOR THIS LOOP
DFD          BNE A416     IS 5Y-1
011          BCC NOFLIP   NO FLIP IF BIT ZERO
908          EOR #TPBIT   BIT IS ONE - INVERT
D02A4        STA TAPOUT   OUTPUT
      ;END OF FIRST 416 µSEC PERIOD
046   B416   LDY #TM1500-1
3     B416B  DEY          LENGTH OF LOOP IS 5Y-1
DFD          BNE B416B
A            DEX
DE5          BNE GETBIT   GET NEXT BIT (LAST IS Ø
C39A6        LDY TEMP2    START BIT) - BY 9 BIT LSR
D            RTS
A     NOFLIP NOP          TIMING
DF1          BCC B416     (ALWAYS)
```

****

Fortsetzung von Seite 2: Nach der Hannover-Messe

Das schließt das Vordringen der 16-Bit-CPUs nicht aus. Vordringlich wird man sich aber hier fragen müssen, wie weit Anwendung, Kosten und Entwicklungshilfsmittel im Einzelfall schon zueinander passen.

Man nehme den Pseudo 16-Bit-Prozessor MC6809 von Motorola. Er stellt eine familienkompatible Weiterentwicklung dar, die sich viele 6502-Betreiber schon oft gewünscht haben (User- und Systemstack, beliebig viele Zeropages durch Vorbefehl, erweiterter und leistungsfähiger Be-

fehlsvorrat mit neuen Adressierungsarten, schnellerer Takt). Die in der 6809 liegenden Leistungsverbesserungen stehen schon jetzt zur Verfügung, ihre Nutzung liegt mit dem üblichen Zeitfaktor aber erst in der Zukunft.

Damit ist kurz auf die Hardwareentwicklung in der 65er Familie einzugehen: Eine CPU 6516, einen Pseudo 16-Bit-Prozessor von Rockwell und Synertec wird es nicht mehr geben. Anderslautende Meldungen wurden von Syntertec in Amerika öffentlich zurückgewiesen. Der Herausgeber erhielt von der deutschen Synertec-Vertretung, Bitronic in München, gleichlautende Nachricht. Die familienkompatible Weiterentwicklung bei CPUs liegt damit beim 6809.

Als neue 65er Bausteine sind von Synertec lieferbar der SY6545 CRT-Controller, das ASCIA SY6551 und ab Juli der Floppy Disk Controller SY1791.

AIM-Spezial (8)

In früheren Heften berichteten wir bereits über eine Eigenwilligkeit des AIM-Monitorprogrammes im Magnetbandbetrieb. Bei bestimmten Datenmengen kann nicht einwandfrei geladen werden. Herr StDir. Rix aus Neumünster gibt dazu folgende Erklärungen: Die Tape-Laderoutine kehrt nicht einwandfrei zum Monitor zurück, wenn der letzte Block im Maschinenformat mehr als N=76 Bytes umfaßt. Der Fehler wird von der Routine LOAD4 (Loc $E321) verursacht. Wird nach erkannter letzter Aufzeichnung dorthin verzweigt, so ist die Anzahl der Aufzeichnungen (C3C2 C1cØ) bereits gelesen worden, es bleiben also nur noch 3 Bytes (Kontrollsumme und CR) zu lesen. Tatsächlich liest LOAD4 aber noch 6 Bytes. Da die gelesenen Bytes nicht geprüft oder sonstwie weiterverarbeitet werden, stört das solange nicht, als die Obergrenze 80 nicht erreicht wird. Umfaßt der letzte Bolck 77 oder mehr Bytes, so macht LOAD4 den Versuch, 80 oder mehr Bytes zu lesen, und das bewirkt (Loc. ED41) den Versuch, einen weiteren Bandblock zu laden. Bei einem Band mit weiteren Aufzeichnungen führt dies trotz richtig geladenem Programm zum ERROR-Abbruch, sobald ein neuer Bandblock identifiziert wird. - Folgen auf dem Band keine weiteren Aufzeichnungen, kann nur mit RESET zum Monitor zurückgekehrt werden.

Hierzu einige Beispiele. Die folgenden Dateien werden mit beliebigem Inhalt mit Dump-Befehl aufgezeichnet:

```
F=FILE1 FROM=200 TO=229 MORE?Y FROM=10C TO=10E MORE?N
F=FILE2 FROM=200 TO=228 MORE?Y FROM=10C TO=10E MORE?N
F=FILE3 FROM=200 TO=227 MORE?Y FROM=10C TO=10E MORE?N
F=FILE4 FROM=200 TO=226 MORE?Y FROM=10C TO=10E MORE?N
F=FILE5 FROM=200 TO=225 MORE?Y FROM=10c TO=10E MORE?N
```

FILE1: Zwei Bandblöcke, 2. Block enthält nur 1 Byte Maschinendaten
FILE2: Ein Bandblock, keine Füll-Nullen
FILE3: Ein Bandblock, eine Füll-Null
FILE4: Ein Bandblock, zwei Füll-Nullen
FILE5: Ein Bandblock, drei Füll-Nullen.

Alle Dateien werden richtig in den Speicher geladen aber nur bei FILE1 und FILE5 wird der Ladevorgang korrekt beendet. - Der Fehler

kann durch Aufzeichnen einiger überzähliger Speicherstellen oder Einfügen von NOP in ein Programm überspielt wrden, löst aber zunächst doch eine Fehlersuche im Kassettenteil aus, insbesondere da der VERIFY-Befehl in allen Fällen korrekt arbeitet.

In Heft 10 wurde eine Schaltungsveränderung für den AIM 65 beschrieben, mit der ein COLD RESET erzeugt werden kann, ohne die Stromversorgung zu unterbrechen. Datenverluste können so vermieden werden. Die genannte Veränderung wurde beim Herausgeber erfolgreich nachvollzogen. Herr Volkmar Klos aus Bochum geht einen viel einfacheren Software-Weg: "Ich ändere vor DILINK-Experimenten grundsätzlich die Speicherstelle $A402 (Low Byte des NMI2-Vektors) von 7B auf 7C (oder 7E), um beim Drücken der RESET-Taste auf jeden Fall einen 'kalten Reset' zu erhalten, ohne die Versorgungsspannung abschalten zu müssen (siehe Anwenderhandbuch Seite 7-37, letzter Absatz) Dabei tritt im 'STEP'-Mode bei Command (V)ON nur ein kleiner Schönheitsfehler auf: Der Akku wird beim Registertrace nicht mehr korrekt angezeigt, was in den meisten Fällen unerheblich ist."

Heft 12 enthielt einen Vorschlag von Herrn Kneissel, wie der SYS-Befehl im AIM-BASIC implementiert werden kann. Herr Michael Krägeloh in Erlangen schreibt dazu: "Beim Nachdenken über das Programm kam mir eine Idee, wie der fehlende SYS-Befehl wesentlich einfacher ersetzt werden könnte. Dazu ist folgendes Programm nötig:

```
OFF6 20 JSR BEFE      Umwandlung des Floating Point-Akkumulators
                      in eine Integer-Zahl (L:AD;H:AC)
OFF9 A5 LDA AD        richtige Reihenfolge herstellen (L/H)
OFFB 85 STA AB        Ergebnis=Sprungziel
OFFD 6C JMP (00AB)

0004 .BYT $F6,$0F     Pointer auf USR-Funktion
```

Aufruf mit A=USR(Adresse)

Das Argument innerhalb der Klammern (in diesem Falle des Maschinenunterprogrammes) wird beim USR-Befehl in den Fließkomma-Akkumulator ab A9 bis AE gebracht. In der Routine ab $BEFE wird die dort jetzt als Fließkommazahl vorliegende Adresse in eine Integer-Zahl umgewandelt und wieder zurück in den Fließkomma-Akku gebracht (siehe auch BASIC-Manual, section Appendices, subject Assembly Language Subroutines). Dort steht jetzt die Adresse in AC/AD, aber in der Reihenfolge HIGH/LOW, statt LOW/HIGH. Nach Umspeichern von AD nach AB steht die Adresse richtig herum in AB/AC. Wie auch im BASIC-MANUAL angemerkt, gibt es in den verschiedenen Versionen des BASIC Unterschiede, was die Sprungadresse angeht, bei der die Umrechnungsroutine für den Fließkomma-Akku zu finden ist. Die richtige Adresse ist in $B006/B007 zu finden. Bei mir ist sie $BEFE."

Die Platine des AIM 65 hat sich in einigen Fällen als anfällig gegen Störstrahlungen erwiesen. Herr Bernhard Kokula in Ludwigshafen konnte diese Anfälligkeit für bisher zwei dort betriebene Systeme wie folgt beseitigen: Die CPU 6502 wurde von ihrem Sockel entfernt. An ihre Stelle trat eine kleine Aufsteckplatine, die die CPU und Treiber für Adreß- und Datenbus trägt. Bei Interesse wird diese Platine dort aufgelegt. Anschrift: Wredestr. 17, 6700 Ludwigshafen.

Dem Vernehmen nach hat Herr Gerold G. Wiese die Firma Rockwell International in München-Martinsried auf eigenen Wunsch verlassen.

Die Fa. GWK, Gesellschaft für technische Elektronik in Herzogenrath präsentierte auf der Hannover-Messe ihr breites Angebot für die Erweiterung des AIM 65/PC 100, die Gehäuseserie, Matrixdrucker, Video-Interface, BASIC-Erweiterung sowie insbesondere Ihre Floppy Disk. Das Betriebssystem (in EPROM) arbeitet voll mit Monitor, Editor, Assembler und BASIC zusammen. Bis zu vier Laufwerke mit je 98 kByte können an den AIM angeschlossen werden.

Floppy Disk für AIM nun auch vom Micro-Shop-Bodensee, M & R Nedela, in Markdorf. Es gelangt das Controller-Board von COMPAS (USA) zusammen mit denfür den deutschen Markt ausgewählten Laufwerken DAM zu Einsatz.

Die bewährte Commodore-Floppy CBM 3040 mit ihrer eigenen Intelligenz (enthält 2 Mikroprozessoren) wird demnächst auch am AIM 65/PC 100 betrieben werden können. Eine einfache Interfacekarte und das Programm für ihre Bedienung über IEE-Bus sind in Vorbereitung. Anfragen zunächst an den Herausgeber.

****

Berichtigung zu Heft 12: BASIC-Keywords to Shifted Keys

1256 DATA 00,00,00,33,39,3F,44,4B
1258 DATA 00,00,53,58,5F,66,6C,75

****

Wegen der Sommerpause erscheint Heft 14 dieser Zeitschrift erst Ende August 1980. Redaktions- und Anzeigenschluß: Sonnabend, der 16.8.1980.

WEGNER-IMPULSTECHNIK

FARBGRAPHIKTERMINAL

Das von mir entwickelte preisgünstige Farbgraphikterminal besteht aus einem Basisgerät und Einschubkassetten in verschiedenen Leistungsstufen und ist an jedes Heimfernsehgerät anschließbar. Als Basisgerät dient die von verschiedenen Firmen für Telespiele angebotene F8-Platine.

Bei allen Ausführungen sind 8192 Bildpunkte adressierbar. Hintergrund- und Vordergrundfarben sind für Zeile und Bildpunkt getrennt wählbar. Eine 15-polige Buchsenleiste verbindet das Farbterminal mit dem anzuschliessenden Prozessorsystem über eine parallele Schnittstelle. Die Ansteuerung für alle bekannten Standardterminalfunktionen im ASCII-Code. Farb- und Sonderfunktionen werden durch Kontrollzeichen generiert.

Die Serie beginnt mit dem Gerät STEWE I:

5x7 Bit-Matrix zur Darstellung von Zahlen, Buchstaben und Sonderzeichen (64). Volle Cursor-Kontrolle (zeichenweise, bzw. bitweise) Punkt malen/nicht malen oder löschen. Verschiedene Strichstärken. Frei wählbare Farben für Vorder- und Hintergrund, Positionierung wie beim Cursor, auch für Sonderzeichen und Gruppen. Display löschen, Zeile löschen, pulsierender Cursor ja/nein.

Dieses preisgünstige Gerät der Serie wird nur komplett geliefert, es ist voll ausbau- und erweiterungsfähig. Preis ab DM 748,-inkl. MWSt.

Die Terminalcassette STEWE II bietet darüber hinaus:

Wahlweise Umschaltung auf 7x9 und 5x5 Bit-Matrizen (96 Zeichen bei 7x9). Groß- und Kleinschreibung, Definition von Zeilen- und Zeichenabstand. Vollständige Anwenderprogramme können dauerhaft gespeichert werden (EEPROM).

Ausführung STEWE III:

Zusätzlich mit Tabulator, unabhängigem Laufschriftgenerator. Möglichkeit der Ein- und Ausgabe von selbstdefinierten Zeichen.

Sonderausführungen sind in vielen Variationen erhältlich. Die Cassetten enthalten Mikrocomputersysteme der 6502-Familie. Spannungsversorgung über das Basisgerät.

Das Basisgerät ist ferner ausbaubar als

DIGITAL-MULTIMETER mit farbiger Anzeige der Meßwerte auf dem Bildschirm und als

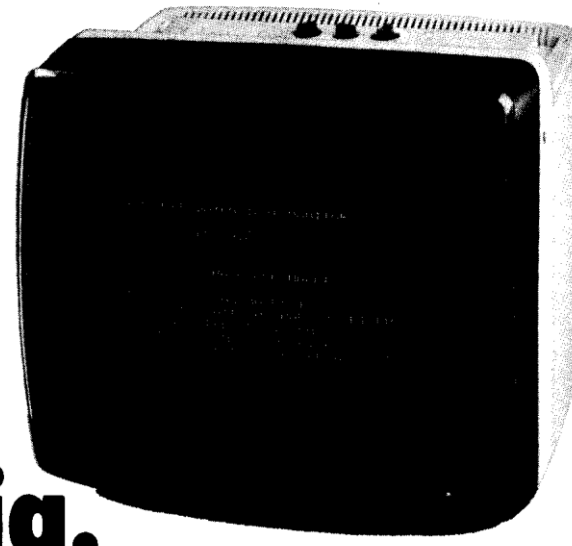
LOGIKANALYSATOR für 8-48 Kanäle mit binärer, hexadezimaler oder auch gemischter Darstellung. Es können bis zu 9 Logikzustände gespeichert werden. Er ist zudem extern triggerbar.

BITTE VERLANGEN SIE MEINEN SONDERDRUCK!

WEGNER-IMPULSTECHNIK 6395 WEILROD 11

PFINGSTBERGWEG 10

Wer beruflich »in die Röhre schaut« sieht bei uns grün. Und dies nicht nur scharf, sondern vielmehr auch ruhig.



Für alle, die mit Bildschirmdaten leben, entwickelte Sanyo den augenfreundlichen Typ DM5912CX. Denn EDV-Informationen lesen, kann weitaus deutlicher und erholsamer sein. So schrieb uns ein anwendungserfahrener Experte: „Der von Ihnen gelieferte Monitor ist ein wirklich angenehmes Display für den Mikroprozessor. Das grüne Bild steht scharf und ruhig und beansprucht mein leider krankes Augenlicht in keiner Weise.".
Das alles wurde von uns in ein formschönes, elfenbeinfarbiges Kunststoffgehäuse mit Rauchglas-Frontblende eingebaut.

Technische Hauptmerkmale:

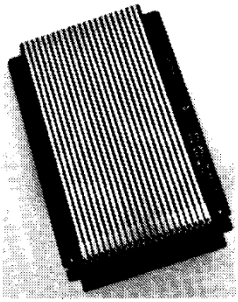
Bildschirmdiagonale:	12" (31 cm)
Videobandbreite:	18 MHz
Geometriefehler:	kleiner als 1%
Displayformat:	80 Charakter, 24 Zeilen
Eingangssignal:	Standard Video 1 V_{SS}

SANYO
VIDEO-SYSTEME

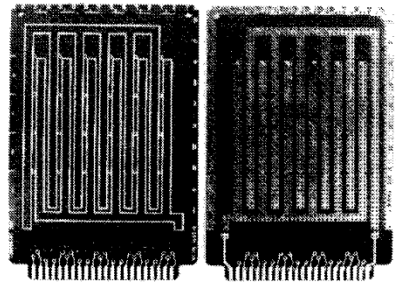
Ausführliche Informationen mit Fachhändler-Nachweis:
SANYO Video Vertrieb GmbH. & Co., Lange Reihe 29, 2000 Hamburg 1, Telefon (040) 24 62 66

Verkauf von Leiterplatten, Steckadapter, Relaisplatinen

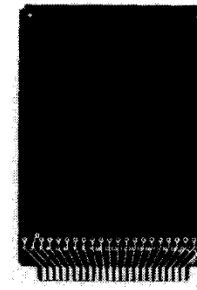
1

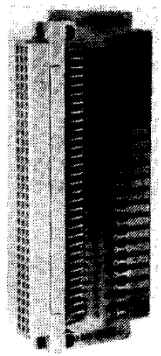


2

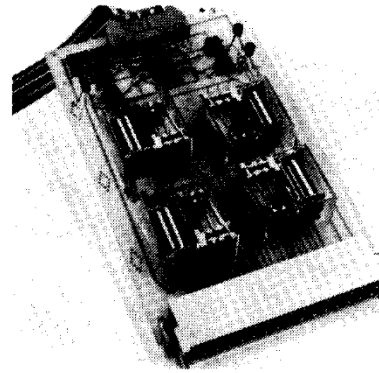


3





4



5

	p.St.	bei Abnahme von 3-5	von 6-10
1. 3690 Card Extender	DM 67,95	DM 57,75	DM 50,95
2. 3682-2 DIP Plugboard	DM 34,05	DM 28,95	DM 25,55
3. 3662 DIP Plugboard	DM 31,10	DM 26,45	DM 23,35
4. C 991 Steckadapter	DM 59,50	DM 50,55	DM 44,65
5. E 941 Relaisplatine	DM 210,60	DM 179,05	DM 157,95

Alle Preise zuzüglich MWSt und Versandspesen. Lieferung ab Lager Köln, Zwischenverkauf vorbehalten. Weitere techn. Beschreibung auf Anfrage.

STECKER

5000 KÖLN 60 (Niehl)
Postfach 60 07 66
Delmenhorster Straße 20
Telefon (02 21) 74 51 12

GWK
GESELLSCHAFT
FÜR TECHNISCHE ELEKTRONIK mbH.

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

NEU! NEU! NEU! NEU! NEU!

Floppy-Disk

für

AIM 65/PC 100

WIR HABEN ES !!!

DAS FLOPPY DISC SYSTEM,DAS WIRKLICH MIT ALLEN

EINGABE UND AUSGABE ROUTINEN VOLL ZUSAMMENARBEITET.

DIES GILT FÜR:A S S E M B L E R

B A S I C

E D I T O R

M O N I T O R

PREISE:

DOPPEL - FLOPPY - STATION,KOMPLETT	3.485,-- DM ZZGL M
CONTROLLER BOARD MIT RES.SOFTWARE	1.248,-- DM "

BITTE FORDERN SIE UNTERLAGEN AN!

GWK

FÜR TECHNISC… …LLSCHAFT
…KTRONIK mbH.

NEU! NEU! NEU! NEU! NEU!

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

Basic Expansion
für
AIM 65/PC 100
jetzt erweitert auf
4 KByte

Gegenüber der schon seit einiger Zeit erhältlichen GWK BASIC ERWEITERUNG (2K) sind in der BASIC EXPANSION (4K) die folgenden Funktionen verbessert oder neu aufgenommen worden:

MEMORY SIZE ändert untere und obere Grenze des Speicherbereichs sowie Line Width.
RENUMBER wesentlich verbessert.Von - bis.Schritteite.
RELOCATE von BASIC Programmen in anderen Speicherbereich.
START von BASIC Programmen,die in anderen Speicherbereichen z.B.in einem EPROM abgelegt sind.
CALL von Programmteilen,die von FLOPPY geladen werden
GET A$ liest Zeichen von allen Input Devices.
DATA INPUT / OUTPUT jetzt unter Programmsteuerung.
Definition von FILENAME und DEVICE durch Programmstatement.
SAVE ist spezifizierbar,z.B.SAVE 50 - 200
PRINT USING. Formatierte Zahlenausgabe.Rechtsbündig,Festkomma
USR Funktionen.Umwandlung Zahl in String und umgekehrt.

Erhältlich auf:Kassette,Diskette,2 EPROM's a 2K,EPROM 4K
Preis: 245,-- DM zuzüglich MWST und Datenträger.

Besitzer der BASIC ERWEITERUNG (2K) erhlten die EXPANSION zum Differenzpreis (Software,nicht Datenträger)

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 4,-/St., Nos. 7-12 DM 7,80 St. Die Hefte 1-6 sind auch als Sammelband beziehbar (s. Anzeige).

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

Das Buch 1-6 des 65xx MICRO MAG erscheint am 15.7.1980 mit ca. 230 Seiten Umfang. Ein thematisch geordneter Sammelband der Hefte 1-6 dieser Zeitschrift, gebunden, ohne Anzeigen. Ein Nachschlagewerk für den Arbeitsplatz Endpreis inkl. Versandkosten DM 26,-

Microprocessor Systems Engineering von Camp, Smay, Triska, 641 S. Lehrbuch für 65xx und speziell für AIM 65, s. Heft 11. Endpreis inkl. Versandkostenanteil DM 66,-

PROGRAMMIERWORKSHOPS 65XX

Einführendes Programmierseminar am 12./13. Sept. 1980 in Ahrensburg bei Hamburg, Fortgeschrittenen-Workshop am 26./27. Sept.80 ebenfalls in Ahrensburg. Kleine Teilnehmerkreise. Schulungen in Firmen auf Anfrage. - Anmeldung beim Herausgeber.