65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 7.80 Nr. 14 August 1980

Vor erst vier Jahren

trat der KIM-1 der Firma MOS Technology in den USA und auch hier in Erscheinung. Inzwischen kann man schon von Nachfolgegenerationen dieses Prozessors sprechen. In immer kürzeren Abständen kommen weiterentwickelte Computer auf der Basis des 6502 auf den Markt. Man kann es eigentlich nur begrüßen, wenn das Angebot für den Anwender immer vielfältiger und leistungsfähiger wird. Wer sich entsprechend orientiert, wird für seine gedachte Anwendung heute kaum noch Kompromisse hinsichtlich Ausrüstung und Erweiterungsfähigkeit eingehen müssen, Kompromisse, die vor wenigen Jahren noch häufig waren.

Die Maschinensprache unserer Computer hat sich seit der KIM-Zeit zum Glück nicht verändert. Veröffentlichte Lösungsansätze bleiben damit übertragbar, wenn man nur die Besonderheiten der Betriebssysteme und der Geräteausrüstung berücksichtigt. Diese Zeitschrift knüpft damit ein Band für alle Systembetreiber. Ihnen fällt nach den Beobachtungen der Umgang mit den Computern und ihrer Software immer leichter. Nicht zuletzt das hier gebotene Software-Forum hat zu dieser erfreulichen Entwicklung beigetragen. Die Leser seien ermuntert, den fruchtbaren Gedankenaustausch weiter zu pflegen.

Inhaltsverzeichnis

Ein- und Ausgabe am AIM 65	3
BASXT - BASIC-Erweiterung (AIM)	17
Generelle Dumpprogramme für breite Drucker	23
Automatische Zeilennumerierung PET/CBM	33
Das Auffinden einer BASIC-Variablen (CBM)	35
Dekadischer Logarithmus per USR (CBM)	36
Garbage Collection-Routine im CBM	38
Rechtsbündige Zahlenausgabe	41
Sichern und Laden dimensionierter Variablen	42
Branchen-Nachrichten	44
Buchbesprechung	45

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

CPU-Karte NICO 65

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

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

CPU-Karte NICO 69

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

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

Combo-Karte

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

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

Video-Karte

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

Preis:	DM 660,-

Buskarte

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

Matrixdrucker

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

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

Interfacekarte dafür

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

Preis:	DM 73,-

Alle Preise zuzüglich 13 % MWSt.

Ein- und Ausgabe am AIM 65

1. Beschreibung des E/A-Systems

1.1. Interfaces

Die Tätigkeiten der Datenverarbeitung lassen sich blockmäßig wie folgt gliedern:

Eingabe - Verarbeitung - Ausgabe.

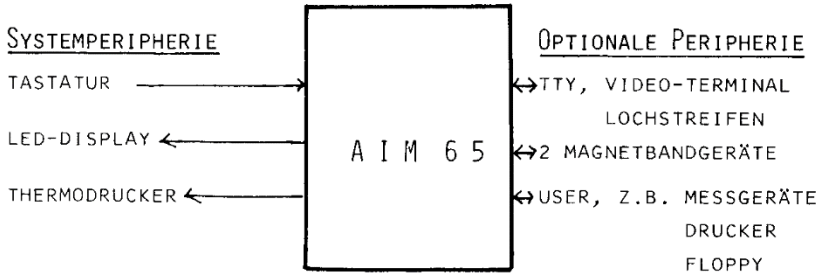
Bei der Entwicklung eines Anwenderprogrammes tritt neben der Programmierung der Verarbeitung regelmäßig die Frage auf, wie man Daten in den Computer hineinbekommt und wie man Mitteilungen und Ergebnisse von dort ausgibt. Die Ein- und Ausgabe (E/A oder englisch I/O - Input/Output) bildet oft ein schwieriges Kapitel innerhalb einer Problemlösung. Dieser Aufsatz möchte daher für den gut dokumentierten AIM 65/PC 100 eine Übersicht und möglichst viele Anregungen geben, wie man für die E/A programmiert. Dabei sollen die zahlreichen im Monitorprogramm enthaltenen Dienstprogramme geordnet und für die mögliche Anwendung erläutert werden.

Das Blockschaltbild Eingabe - Verarbeitung - Ausgabe läßt sich hardwaremäßig auch so beschreiben:

Eingabeeinheiten-Interfaces-Zentraleinheit-Interfaces-Ausgabeeinheiten.

Für den Aufbau eines Computers ist also typisch, daß die verarbeitende Zentraleinheit nur über besondere den Signal- und Zeitbedürfnissen angepaßte elektrische Schaltungen (Schnittstellen) mit Ein- und Ausgabeeinheiten zusammenarbeiten kann. Die Peripherie mag dabei auf den Menschen als Bediener zugeschnitten sein (Tastatur, Anzeigen, Drucker etc.) oder auf andere Maschinen (Meßeingänge, Stellglieder).

Die am AIM vorhandenen Schnittstellen lassen sich in die mit Peripherie fest beschalteten Systeminterfaces für Tastatur, LED-Display und Thermodrucker gliedern und solche, an die in Anwenderoption Geräte angeschlossen werden können, nämlich Fernschreib- oder Videoterminal (TTY), Lochstreifen, 2 Magnetbandgeräte und Anwender-VIA.



DIE SCHNITTSTELLEN DES AIM 65

Der Betrieb aller Schnittstellen - bis auf die Anwender-VIA - ist vom Monitorprogramm her unterstützt.

Begrifflich sollte bereits an dieser Stelle die anwenderdefinierte Ein- und Ausgabe (USER) abgegrenzt werden. Es sind reine 'Softwareschnittstellen' im Programmablauf, die der Anwender aktivieren kann und die es ihm dann erlauben, auf vorhandenen Interfaces mit anderen Codes (z.B. Kleinschreibung von der Tastatur her oder auf dem Printer) zu arbeiten oder zusätzliche Interfaces für E/A heranzuziehen. User-I/O hat also nur gelegentlich mit der Anwender-VIA zu tun.

1.2 Aufgaben des Monitorprogrammes

Das Monitorprogramm ist der Hauptdienstleister für das System. Seine zahlreichen E/A-Programme werden ebenso vom Text-Editor, vom Assembler wie auch vom BASIC benutzt. Beim E/A geht es aber keineswegs nur um den **Transport** von Information, sondern in erheblichem Umfang auch um die **Code-Erzeugung** und um **Code-Umwandlung**. Die vorherrschende interne Informationsdarstellung ist der ASCII-Code.

Nehmen wir die Tastatur: Sie wird vom Monitor decodiert. Es handelt sich also nicht um eine Encoder-Tastatur mit fertigen ASCII-Zeichen beim Strobe-Signal. In der Routine **GETKEY** wird 6-Bit ASCII in Abhängigkeit davon erzeugt, welche Kreuzungspunkte einer Matrix geschlossen sind. Je nach aufrufendem Programm kann das ASCII-Byte direkt abgelegt werden (im Display oder im Textspeicher) oder es muß zu einem Halbbyte (nibble) reduziert werden, wenn es sich um die hexadezimale Ziffer einer Adresse handeln soll (Code-Wandlung).

Umgekehrt müssen aus einem einzigen hexadezimalen Adressbyte zwei ASCII-Bytes für die lesbare Ausgabe gewandelt werden. Für den Printer muß das Zeichen mit Hilfe einer Zeichengeneratormatrix noch anders ungewandelt werden. Im TTY- und Tonbandbetrieb geht es um Erzeugung und Empfang serieller Daten, um weitere Arten der Umformung.

Neben den Funktionen Transport und Codewandlung hat das Monitorprogramm **verwaltende Einrichtungen**: Es muß wissen, von welcher Stelle das nächste Zeichen zu holen ist und an welche Stelle es zu bringen ist. Für die Verwaltungsaufgaben wird das **System-RAM** auf dem Combobaustein 6532 benutzt, und zwar im Adreßbereich $A400 bis $A4FF. Neben vielen Zellen, die der Zwischenablage dienen, finden sich dort zahlreiche **Flags** (Merker von 1 Byte) für die E/A-Steuerung (z.B. ob der Printer eingeschaltet ist) und **Cursor-Positionen**, die auf die nächste anzusprechende Zelle in den Ein- und Ausgabepuffern weisen.

1.3 Die Pufferung der Ein- und Ausgabe

Mit den Zeichenpuffern kommen wir auf das sehr sinnvoll gestaltete E/A-System des AIM zu sprechen. - Die Verarbeitung im Prozessor erfolgt an jeweils 1 Byte. Aber nicht alle Peripherie arbeitet byteweise. Der Thermodrucker soll nur dann ausgeben, wenn das 21. Zeichen ankommt oder wenn vorher Carriage Return 'CR' ausgelöst wurde. Für die Ansammlung der Zeichen wird daher in 20 Zellen der Printer-Buffer ab $A460 vorgehalten.

Da die Speicherung auf Magnetband blockweise erfolgt, sind hierfür ebenfalls 2 E/A-Puffer vorgesehen, einer für jeden Tonbandkanal.

Ein Ausgabepuffer ab $A438 nimmt die vom LED-Display anzuzeigenden Zeichen auf. - Andere Peripherie arbeitet normal ohne E/A-Puffer, die Tastatur und das TTY-Interface. Eine Besonderheit stellt auch das **KIM-Bandformat** dar: Ein Speicherbereich wird von Anfang bis Schluß ohne Pufferung entweder geladen oder gedumpt.

Es sollte hier bereits erwähnt werden, daß weitere Firmware zusätzliche Zeichenpuffer hat. Der **Editor** unterhält einen Eingabepuffer für die offene Zeile ab $A438, der mit dem Display-Ausgabepuffer identisch ist. Daneben gibt es in der Zeropage einen Buffer für die Ablage eines Suchstrings zum Vergleich (F- bzw. C-Command).

BASIC unterhält einen Tastatur-Eingabepuffer in der Zeropage. Eine Programmzeile/Anweisung wird erst mit 'CR' von dort übernommen/ausgeführt. Der **Assembler** schließlich verwaltet einen 'Kartenpuffer' ICRD von 60 Zeichen ab $46 und einen Code-Buffer für die Ausgabe von 20 Zeichen ab $170.

Die Abarbeitung des Prozessors in der Breite von jeweils 1 Zeichen verlangt, wie bereits erwähnt, eine Verwaltung der nächst zu bearbeitenden Pufferstellen. Im Zusammenhang damit ist eine sinnvolle Automatik implementiert: Sobald ein Ausgabepuffer gefüllt ist, wird sein Inhalt auf die zugeordnete Peripherieeinheit ausgegeben, z.B. auf den Drucker oder das Magnetband. Wenn ein Eingabepuffer erschöpft ist, wird er entweder gesperrt (61. Zeichen in einer Editorzeile) oder es werden automatisch neue Zeichen nachgezogen (1 Block vom Tonband).

1.4 E/A-Initialisierung in den Kopfverteilern WHERE1 und WHERE0

Wie erfährt ein Programm nun, mit welcher Peripherie der Anwender zusammenarbeiten möchte? Nehmen wir das Vorbild des Monitors: Mit dem power on-Reset sind die Systemeinheiten Tastatur, Display und Printer aktiviert. Der Printer kann abgeschaltet werden (Tasten CTRL + PRINT). Der Monitor wartet auf ein Kommando. Es werden durch entsprechenden Tastendruck folgende Routinen angesprungen:

L allgemeine Laderoutine in $E2E6 mit Folgeaufruf
JSR WHERE1 in $E848 und

D allgemeine Dumproutine in $E43B mit Folgeaufrufen
FROM= und TO= sowie
JSR WHERE0 in $ E871.

Für die E/A-Steuerung sind die allgemeinen Ansprungspunkte für Laden und Dump nicht entscheidend, sondern die nachfolgenden Kopfverteiler WHERE1 und WHERE0, die als Unterprogramm aufgerufen werden. Man analysiere diese Routinen im Monitor Program Listing einmal genauer. Sie schreiben 'IN=' bzw. 'OUT=' auf das Display und erwarten einen Tastendruck, zeigen hernach die betätigte Taste an und Speichern das Zeichen als ASCII im **INFLG** $A412 für WHERE1 bzw. im **OUTFLG** $A413 für WHERE0. Es folgt durch eine Reihe von CMP-Befehlen die stufenweise Abfrage, welche Taste gedrückt wurde. Bei Gleichheit werden Initialisierungsroutinen wie folgt angesprungen:

Taste	WHERE1-Aussprung	WHERE0-Aussprung
T	JSR FNAM, $E8A2 mit X=0 und JMP LOADTA	JSR FNAM, $E8A2 mit X=1 und JMP DUMPTA
K	JSR FNAM, mit X=0 und JMP LOADK1	JMP FNAM, mit X=1 -

Taste	WHERE1-Aussprung	WHERE0-Aussprung
U	JMP (UIN), $108 mit Carry Clear	JMP (UOUT), $10A mit Carry Clear
P	-	Ausdruck der letzten Zeile mit LDA // 'CR' und JMP OUTPR1, $F000
sonst	RTS, ohne	JMP CRLOW, $EA13 Leerzeile und Ausgabe auf Systemeinheiten.

Man beachte, daß die angesprungenen Routinen z.T. zu weiteren Programmabschnitten fortschreiten. Bei Magnetbandroutinen wird z.B. noch nach Filename und nach der Bandeinheit gefragt, das Band läuft an, das benannte File wird aufgesucht, ein erster Block geladen.

Die Unterprogramme WHERE1 und WHERE0 dienen der Zuordnung per INFLG bzw. OUTFLG und der Initialisierung der Peripherie. Dementsprechend sind auch die anwenderbestimmte Ein- und Ausgabe mit 'U' zu behandeln, z.B. um ein File aufzusuchen und um einen ersten Block in einen Buffer zu laden oder um bei der Ausgabe z.B. auf einen Matrixdrucker den Vorschub auf neue Seite mit Ausdruck einer Kopfzeile zu veranlassen.

Die Kopfverteiler WHERE1 und WHERE0 dienen also dem Anlaufen der Peripherie und dem ersten Bereitstellen von Daten in Buffern, nicht jedoch dem später folgenden byteweisen Datentransport, der über INALL und OUTALL gemanagt wird. In dieser Sicht gleichen die beiden Kopfverteiler einem OPEN FILE (by Name).

Der AIM kennt, auch in seinem BASIC, allerdings keine OPEN- und CLOSE-Befehle, die daher in der vorgenannten Art implementiert werden müssen. Er kennt auch keine logischen File-Nummern.

Dem Leser seien gleichwohl einige Anregungen für eine Erweiterung gegeben: Die Kopfverteiler lassen nur wenige E/A-Möglichkeiten zu, ihr Mechanismus ist aber nützlich. Was hindert uns, einen entsprechenden Vorspann zu schreiben und dann in den Rest der Abfrage hineinzuspringen? Es seien z.B. die Ausgabeeinheiten (devices) 'A' und 'B' zusätzlich zugelassen. Wir programmieren:

```
WHEROX  LDY #M9-M1      ADRESSIERE MESSAGE
        JSR KEPR        PRINT "IN="
        STA OUTFLG      SAVE DEVICE NAME
WHERO3  CMP #"A"        A-DEVICE?
        BNE WHERO5      NO
        JMP STARTA      INITIALIZE DEVICE A
WHERO5  CMP #"B"        B-DEVICE?
        BNE WHERO6      NO
        JMP STARTB      INITIALIZE DEVICE B
WHERO6  JMP WHERE0+8    $E850, COMPARE THE REST IN WHERE0
```

In obigem Programm sind zur Abfrage der Ausgabeeinheit die ersten drei Befehle mit denen bei WHERE0 identisch. Daran schließt sich die anwenderbestimmte Abfrage nach 'A' und 'B' an. Nach ergebnislosem Durchlauf im Vorspann erfolgt der Sprung nach WHERE0+8 zum Rest der Abfragen. WHEROX ist wie WHERE0 als Unterprogramm aufzurufen.

Es liegt der Gedanke nahe, eine Erweiterung in umgekehrter Reihenfolge aufzubauen. In WHEROX ruft man zunächst WHEREO als Unterprogramm auf. Nach einem Aussprung zu einer Initialisierung z.B. für den Printer würde man an einer Stelle entsprechend WHERO3 fortfahren. Zu diesem Zeitpunkt dürfte aber der eingangs mit dem OUTFLG geladene Akku zerstört sein und damit für die weiteren Abfragen nicht mehr taugen.

Der Ansatz von WHEROX bietet noch ungenutzte Möglichkeiten, jeweils mehrere Devices/Files für das E/A zu initialisieren, wenn z.B. die Taste 'A' (OUTFLAG) gedrückt wurde:

```
WHERO3  CMP #'A'        DEVICE A, C AND D?
        BNE WHERO5      NO
        JSR STARTA      INITIALIZE DEVICE A
        JSR STARTC      AND C
        JMP STARTD      AND FINALLY D
WHERO5  CMP #'B'
        ...
```

Eine **Auffächerung der Initialisierung** auf mehrere Geräte, wie im letzten Beispiel, zieht notwendig Maßnahmen zur Ansprache ggfs. mehrerer Filenames und sorgfältige Verwaltung sonstiger Steuerinformationen nach sich.

Die vorstehenden Beispiele bedeuten eine Überlagerung zu WHEREI und WHEREO. Bei der Konstruktion dieser Kopfverteiler scheint es auf den ersten Blick systemgerechter, eine Auffächerung auf mehrere Files in dem Moment vorzunehmen, da man sich per OUT=U auf der anwenderdefinierten Ebene befindet. Man müßte dort Unterabfragen und entsprechende initialisierende Aussprünge (JMP) machen und man müßte anwenderdefinierte IN- und OUT-Flags setzen, denn man darf für die späteren Datentransporte das mit 'U' vorbesetzte Flag nicht verändern.

1.5 Die Transportverteiler INALL und OUTALL

Im Abschnitt 1.3 sprachen wir bereits die den einzelnen Peripheriegeräten zugeordneten Zeichenpuffer für den blockweisen Transport an. Aufgabe von INALL und OUTALL ist es nun, ein einzelnes Zeichen (character) von dort zu holen bzw. dorthin zu schreiben. Die Übergabe des Zeichens erfolgt im Akku. Wenn der Zeichenspeicher abgearbeitet ist, wird automatisch ein Block nachgeladen (Eingabe) oder eine Zeile /ein Block ausgegeben (Ausgabe). Die Verwaltung liegt im Monitorprogramm mit seinen Flags und Pointeradressen. **INALL** residiert ab $E993, **OUTALL** ab $E9BC. Der Programmablauf mit Abfrage des Flags für die aktive Einheit erfolgt wiederum mit CMP-Befehlen.

Flag	INALL	OUTALL
	LDA INFLG	PHA LDA OUTFLG
T	JMP TIBYTE $ED3B	JMP TOBYTE $F18B
K	JMP GETTAP, $EE29	JMP OUTTAP, $F2A4
M	JMP MREAD, $FAD0	-
P	-	Turn printer on JMP OUTPRI, $F000
U	JMP (UIN), $108 mit Carry Set	JMP (UOUT), $10A mit Carry Set und dem Zeichen noch auf dem Stack

Flag	INALL	OUTALL
L	JMP GETTTY, $EBDB	nur dann JMP OUTTTY, $EEA8, wenn Schalter=TTY
X	Tastatur	PLA und RTS, do nothing
-	JMP RDRUB, $E95F Tastatur lesen	JMP OUT1, $E97B mit Ausgabe auf die aktiven Systeminterfaces Printer, Display oder ggfs. TTY.

Die jeweils angesprungenen Programmteile sind Transportroutinen mit nachgeschalteter Automatik, sofern Puffer angesprochen werden. Das gilt nicht für OUT=X (dummy device), bei dem nichts ausgegeben wird, und es gilt nicht für IN=U und OUT=U, für die der Anwender auf geeignete Art Transporte organisieren muß.

Dem Leser fällt auf, daß es per INALL natürlich kein Einlesen vom Printer geben kann. Auch für das Memory Read (M) mit der Pointeradressierung (NOWLN),Y per $DF/E0 für Editor und Assembler gibt es keinen Counterpart auf der Ausgabeseite. Dieser müßte per User (U) implementiert werden. Hierfür einer der vielen möglichen Vorschläge, in U1 erfolgt die Initialisierung eines Buffers, in STORE die Abspeicherung und Erhöhung des Pointers.

```
USER    BCS STORE        WENN MAN VON OUTALL KOMMT
U1      LDA #<BUF        INITIALISIERE ADRESSE IM POINTER
        STA NOWLN        $DF
        LDA #>BUF
        STA NOWLN+1
        RTS              RETURN TO OUTALL'S CALLER
STORE   LDY #0
        PLA              GET CHARACTER FROM STACK
        STA (NOWLN),Y    STORE AT INDIRECT ADDRESS
        JSR AD1          $F928 INCREMENT POINTER BY 1
        RTS              RETURN TO INALL'S CALLER
```

Die Routine OUTALL enthält für User eine gefährliche Unsymmetrie, die man nach der Ablauflogik aber nicht als Programmierfehler ansehen sollte. Eingangs OUTALL wird das zu speichernde Zeichen mit PHA auf den Stack geschoben und für alle übrigen Abspeicherungen von dort zurückgeholt. Nur für den User liegt es noch dort. Wie im vorstehenden Beispiel USER muß es vor der Abspeicherung erst von dort zurückgeholt werden.

Am Ende des Abschnittes hatten wir für die Bedienung weiterer Peripherieeinheiten die Routine WHEROX vor WHEREO+8 geschaltet. Gleiches können wir mit INALL und OUTALL programmieren:

```
INALLX  LDA INFLG        FROM WHICH DEVICE?
        CMP #'A'         FROM A?
        BNE IN1          NO
        JMP INA          GET CHAR. FROM A
IN1     CMP 'B'          FROM B?
        BNE IN2          NO
        JMP INB          GET CHAR. FROM B
IN2     JMP INALL+3      CHECK FOR THE OTHER DEVICES ($E996)
```

Eine **Auffächerung der Ausgabe** auf mehrere Geräte per Betriebssystem erfolgt normal nur für Printer und Display, wenn der Printer eingeschaltet ist.

Eine Auffächerung auf zusätzliche Einheiten könnte entsprechend den Vorschlägen in 1.4 mit OUTALY wie folgt bewirkt werden:

```
OUTALY  PHA             SAVE CHARACTER ON STACK
        LDA OUTFLG      TO WHICH DEVICE?
        CMP #'A'        THE A-GROUP?
        BNE OUTY1       NO
        PLA             CHAR. BACK AGAIN
        PHA             AND RETURNED TO STACK
        JSR STOREA      WRITE TO A-DEVICE
        PLA             TO AND FRO
        PHA
        JSR STOREC      WRITE TO C-DEVICE
        PLA
        JMP STORED      AND A FINAL JUMP TO D-DEVICE
OUTY1   ...
        JMP OUTALL+4    $E9C0 OR WRITE TO THE OTHER DEVICES
```

Die Verwendung anwenderbestimmter Auffächerung oder der Funktionen U (UIN/UOUT) macht eine Programmierdisziplin erforderlich. Bei Durchsicht der normalen E/A-Routinen des Monitors finden wir regelmäßig einen Schutz der Register X oder Y oder beider, bei der Ausgabe einen zusätzlichen Schutz des Akkumulators, in dem das anfangs übergebene Zeichen unversehrt zurückgegeben wird. Die Programmsequenzen lauten dazu

```
TXA - PHA und am Ende PLA - TAX
TYA - PHA und am Ende PLA - TAY
 JSR PHXY und am Ende JSR PLXY
           oder
PHA - JSR PHXY und am Ende JSR PLXY - PLA
```

Im vorstehenden Beispiel OUTALY haben wir den Schutz des Akkumulators bereits besorgt. Das könnte nach dem Vorbild der im Monitor enthaltenen E/A-Routinen natürlich auch innerhalb der Routinen STOREA usw. erfolgen, zugleich auch mit dem Schutz von X und Y. Sie würden damit bequemer einsetzbar und systemgerecht sein, eine beliebige Verkettung der Ausgabemodule wird möglich.

Die Register X und Y enthalten vor dem Aussprung auf den Durchlaufverteiler regelmäßig Laufvariablen zur indizierten Adressierung von Puffern. Diese dürfen nicht durch die E/A zerstört werden, sonst verliert sich das System im Nirgendwo.

Für den Fall der Ausgabeauffächerung auf der Anwenderebene OUT=U wäre wiederum eine Abfragesequenz an die vom Benutzer per OUTALL gesetzten Flags zu richten. Wir werden später darauf eingehen.

Es gibt eine weitere Möglichkeit zur Auffächerung der Ausgabe. Sie liegt in der Benutzung des DILINK, eines Adreßvektors in den Zellen $A406/A407. Er wird von der Programmstelle OUTDP1, $EF02, aus angesprungen und weist normal auf $EF05, die Ausgabe auf das LED-Display. Mit Veränderung des Adreßvektors ist es üblich, eine Ausgabe auf ein Video-Interface vorzunehmen.

Ein Hinweis: Die Routine OUTALL eröffnet für die bequeme Ausgabe eines Zeichens auf den Printerbuffer eine bequeme Möglichkeit, egalweg ob der Printer eingeschaltet ist oder nicht, wobei der Einschaltzustand anschließend restauriert wird:

```
PHA       Save character on stack
JSR OUTA2+4 , $E9D4
```

Die Wirkung entspricht einem PRINT! in BASIC.

In der Zusammenfassung für die E/A-Steuerung und für den ersten Abschnitt können wir feststellen:

E/A-Steuerung durch

WHEREI	WHEREO
INALL	OUTALL
INFLG $A412	OUTFLG $A413.

Der AIM 65/PC 100 bedient per Hard- und Firmware die residente Systemperipherie Tastatur, LED-Display und Thermodrucker sowie als Optionen Magnetband und TTY-Terminals. Daneben gibt es anwenderdefinierte Ein- und Ausgabe (Softwareinterface), die sich auf beliebige Peripherieeinheiten und Informationsdarstellung beziehen kann.

Mit kleinen Überlagerungsprogrammen ist es problemlos möglich, anwenderdefinierte Einheiten zuzuschalten und in der Initialisierung sowie im Datendurchlauf zu bedienen. Es gibt keine logischen Files und keine Anweisungen wie OPEN oder CLOSE (device/file). Die Kopfverteiler WHEREI und WHEREO üben aber eine gleichartige Funktion für OPEN aus, und zwar für jeweils eine device/file auf der Ein- und Ausgabeseite. Eine Erweiterung auf mehrere aktive Peripherieeinheiten muß vom Anwender programmiert werden.

Der eigentliche Datentransport erfolgt über die Durchlaufverteiler INALL und OUTALL. Einige E/A-Einheiten übergeben/übernehmen jeweils 1 Zeichen, im allgemeinen ist die E/A des AIM jedoch gepuffert, eine sinnvolle Automatik sorgt für Auffüllung der Eingabepuffer oder Ausschreibung der Ausgabepuffer. Das E/A-System kann damit als vielseitig und gut dokumentiert bezeichnet werden.

Nach dieser Beschreibung können wir zu den Bedienungsprogrammen für die einzelnen Peripherieeinheiten übergehen.

2. Die Bedienung der Systeminterfaces

Als Systeminterfaces haben wir Tastatur, LED-Display und Thermodrukker als residente Peripherie sowie Tonband und TTY als Optionen bezeichnet. Dieser Abschnitt stellt nun die jeweils zugehörigen Betriebsprogramme dar. Zum besseren Verständnis der Software nehme der Leser das AIM Monitor Program Listing zur Hand. Die Programmierung von Peripherie ist immer von der elektrischen Beschaltung abhängig. Man ziehe daher ggfs. auch das Anwenderhandbuch nebst Schaltplänen und das Hardwarehandbuch mit der Beschreibung der Interfacebausteine heran.

2.1. Die Tastatur

Die Tastatur bildet eine Matrix von 8x8 Leiterbahnen, in deren Kreuzungspunkten die Tasten sitzen. Betätigte Tasten werden per Betriebsprogramm durch eine Abrasterung erkannt (ein Sendeport zieht die Ausgabepins nacheinander auf LOW, der Empfangsport untersucht, an welchem seiner Pins das LOW ankommt).

Die zentrale Tastaturroutine ist **GETKEY** in $EC40 mit den Unterprogrammen ROONEK (prüfe, ob die zuletzt betätigte Taste noch gedrückt ist) und DEBKEY zur Tastaturentprellung um 5 msek..

Die GETKEY-Routine übergibt komplette ASCII-Zeichen im Akku. Es handelt sich dabei um 6-Bit ASCII, also nur Großbuchstaben, Ziffern, Sonderzeichen. Erzeugt werden allerdings zunächst auch Kleinbuchstaben, die jedoch im weiteren Verlauf auf Großbuchstaben abgeändert werden. Mit Zusatzprogrammen ist es möglich, gleichwohl auch Kleinbuchstaben mit dieser Tastatur zu erzeugen (s. Heft 8, S. 39 und Heft 12, S. 43, eine verbesserte Version werden wir im Abschnitt für User-Eingabe abdrucken). An den x- und Y-Registern kann man allerdings nach GETKEY sehen, welche Tasten oder Kombinationen betätigt wurden:

X=00 Zeichentaste wurde allein betätigt
X=10 CTRL-Taste wurde zusätzlich betätigt
X=20 Linke Shift-Taste wurde betätigt
X=40 Rechte Shift-Taste wurde betätigt.

Das Y-Register enthält nach GETKEY noch die Indizierung auf die ASCII-Codetabelle ROW1 in $F421.

Beispiel 2.1.1: GETKEY kann vom Anwender beliebig aufgerufen werden. Wir legen eine Zeichenkette mit bis zu 256 Zeichen im Speicher ab $300 ab. Mit Taste ESCAPE (ASCII $1B) brechen wir ab.

```
          GETKEY=$EC40    SYMBOLERKLÄRUNG
          MEMY=$E7
          BUF=$300

          *=$200          ANFANGSADRESSE
          LDY #0          INDEX ALS LÄUFER
LOOP      STY MEMY        ERREICHTEN WERT RETTEN
          JSR GETKEY      HOLE ZEICHEN VON DER TASTATUR
          CMP #$1B        ESCAPE?
          BEQ OUT         JA
          LDY MEMY
          STA BUF,Y       SPEICHERE AN DER INDIZIERTEN STELLE
          INY             INDEX+1
          BNE LOOP        DO IT AGAIN
OUT       BRK             RÜCKKEHR ZUM MONITOR
```

Mit dem M-Command können wir hernach die ab $300 abgelegten Bytes analysieren. - Das vorstehende kleine Programm belegt die Einsatzmöglichkeit von GETKEY, es hat allerdings den Schönheitsfehler, daß es nichts auf dem Display anzeigt.

Beispiel 2.1.2: Es soll zu einem Anwenderprogramm nach $400 verzweigt werden, wenn Taste 4 betätigt wird, nach $500 bei Taste 5. Alle anderen Tasten außer ESCAPE werden ignoriert. Das LED-Display soll die jeweils betätigte Taste anzeigen. Wir greifen dazu auf die Routine OUTDIS vor, die die LED-Anzeige besorgt.

```
          OUTDIS=$EF05    SYMBOLERKLÄRUNG
          GETKEY=$EC40

          *=$200          PROGRAMMANFANG AB $200
START     JSR GETKEY      HOLE ZEICHEN VON DER TASTATUR
          CMP #$1B        ESCAPE?
          BEQ OUT         JA
          JSR OUTDIS      ANZEIGE
          CMP #'4'        EINE 4?
          BNE FUENF       NEIN      G
          JMP $400
```

```
FUENF   CMP #'5'        EINE 5?
        BNE START       KEIN ZULÄSSIGES KOMMANDO
        JMP $500        AUSSPRUNG
        *=$400
        BRK             ABLAGE EINES BRK-BEFEHLES
        *=$500
        BRK
```

Die 'Anwenderprogramme' in $400 und $500 bestehen jeweils nur aus dem BRK-Befehl, sie geben also nur an das Monitorprogramm des AIM zurück. Der disassemblierte Folgebefehl zeigt aber wo wir gelandet sind.

Mit GETKEY und den folgenden Abfragen haben wir eine eigene Kommandoebene aufgebaut. Ehe wir diesen Gedanken weiter verfolgen ein Hinweis: Das Unterprogramm OUTDIS gibt am Ende den von GETKEY übernommenen Akku-Inhalt unversehrt zurück.

Der Aufbau einer **eigenen Kommando-Ebene** wurde bereits in Heft 10, S. 34 dieser Zeitschrift beschrieben: Die von GETKEY geholten Zeichen werden durch indizierte Adressierung mit einer Tabelle zulässiger Befehlstasten (ab KEY1) verglichen. Bei zutreffendem Vergleich wird das Indexregister zur Beschaffung eines (indirekten) Sprungvektors aus einer zweiten Tabelle JMPTAB benutzt (s.u.).

Beispiel 2.1.3: In Abhängigkeit von den zugelassenen 5 Befehlstasten K, L, M, N, und Z soll der Aussprung zu verschiedenen Anwenderprogrammen erfolgen:

```
        GETKEY=$EC40
        JUMP=$A47D      ZUR VEKTORHINTERLEGUNG
        KPRO=$400       ADRESSEN DER ANWENDERPROGRAMME
        LPRO=$410
        MPRO=$420
        NPRO=$430
        ZPRO=$440

        *=KPRO          DIE ANWENDERPROGRAMME BESTEHEN NUR AUS
        BRK             DEM BRK-BEFEHL
        *=LPRO
        BRK
        *=MPRO
        BRK
        *=NPRO
        BRK
        *=ZPRO
        BRK
        *=$200          PROGRAMMSTART
KEY     JSR GETKEY      HOLE ZEICHEN VON DER TASTATUR
        LDX $4          ADDRESSER UND ZAEHLER
KEY1    CMP BEF,X       GEHE DIE ERLAUBTEN BEFEHLE DURCH
        BEQ JUMPE       BEI ERLAUBTEM BEFEHL
        DEX
        BPL KEY1        PRUEFE NAECHSTE MOEGLICHKEIT
        BMI KEY         JUMP ALWAYS, GET A NEW KEY
JUMPE   TXA             BEFEHL GEFUNDEN, INDIZIERUNG VORBEREITEN
        ASL A           MULITPLIKATION MIT 2
        TAX             ZURUECK ZUR INDIZIERUNG
        LDY #1          FANGE VON HINTEN AN
J1      LDA JMPTAB,X    HOLE ADRESSE
```

```
        STA JUMP,Y      HINRERLEGE VEKTORBYTE
        INX
        DEY
        BPL J1          ZWEITER DURCHLAUF
        JMP (JUMP)      INDIREKTER SPRUNG PER VEKTOR
BEF     .BYT 'KLMNZ'    ERLAUBTE TASTEN
JMPTAB  .DBY KPRO,LPRO,MPRO,NPRO,ZPRO
                        ADRESSEN DER ANWENDERPROGRAMME
```

Rein zu Testzwecken bestehen die 'Anwenderprogramme' KPRO usw. nur aus dem BRK-Befehl, damit wir das Ziel des Aussprunges überprüfen können. An der Programmstelle JUMPE wird X durch TXA, ASL A und TAX mit 2 multipliziert, um mit X auf einem geraden Tabellenplatz aufzusetzen und nach INX auf dem folgenden ungeraden. Das Y-Register wird benutzt, um die Abspeicherung in der Schleife zu besorgen. Aber: X steigt auf, Y dekrementiert. Damit die Reihenfolge der Vektorablage richtig bleibt, muß die Tabelle JMPTAB mit der .DBY-Anweisung angelegt werden, statt der üblichen .WOR-Anweisung. In JMPTAB sind die Adreßbytes also in der geschriebenen Reihenfolge HIGH/LOW abgelegt, in JUMP steht der Vektor in der Folge LOW/HIGH, weil die Indices überkreuzt arbeiten.

Dieses Prinzip ist ähnlich im Monitorprogramm ab Stelle $E194 verwirklicht.

Beispiel 2.1.4: Die Tastatur soll für die Zeit einer Abwesenheit vom Arbeitsplatz gegen unbefugte Betätigung verriegelt werden. Zur Wiedereröffnung der Tastatur sind die Tasten A und Z in der richtigen Reiheinfolge zu drücken. In diesem Falle kehrt der Computer in das Monitorprogramm zurück. Das nachstehende kleine Programm könnte man natürlich auch in einer verfeinerten Form auf eine Funktionstaste legen:

```
*=START
JSR GETKEY      HOLE ZEICHEN VON DER TASTATUR
CMP #'A'        EIN A?
BNE START       VERWORFEN
JSR GETKEY      HOLE ZEICHEN NACH 'A'
CMP #'Z'        EIN Z?
BNE START       NEIN, FANGE VON VORNE AN
JMP E182        JA, RICHTIG, SPRINGE IN DEN MONITOR
```

Zu GETKEY ist ein wichtiger Hinweis nachzutragen. Der Programmablauf steckt in dieser Routine fest, solange keine Taste gedrückt wurde. Solange GETKEY läuft, kann man also keine andere Abfrage am Prozessor durchführen. Ggfs. muß man GETKEY durch ein Interruptprogramm aufbrechen. - Dieses Festsitzen und der Weg über Interrupt sind in der Programmierung ggfs. etwas hinderlich. Wie kann man nun in einem unbehinderten Durchlauf feststellen, ob überhaupt eine Keyboard-Taste betätigt wurde? Zunächst ein Versuch:

Beispiel 2.1.5: Erkennen der Tastenfreigabe. Wir starten das nachfolgende Programm in $200 und halten die auslösende G-Taste (G=GO) etwas länger fest. In dieser Zeit bleibt das Display dunkel, weil die zuletzt betätigte Taste noch gedrückt ist. Erst mit deren Freigabe verläßt man ROONEK und gelangt über BRK in den Monitor.

```
ROONEK=$ECEF
*=$200
JSR ROONEK
BRK
```

Ehe man eine neue Taste dekodieren will, muß man also per ROONEK erst abwarten, daß die zuletzt betätigte freigegeben worden ist. Sofern keine Taste im Anschlag ist, wird ROONEK gradlinig und ohne Verzögerung durchlaufen, so daß man weitere Verarbeitungen am Prozessor durchführen oder prüfen kann, ob evtl. andere Peripherie bedient werden muß. Das Y-Register ist dann 00, in der Zelle STBKEY $A42B findet man zugleich $FF.

Wenn aber doch eine Taste betätigt ist, dann steckt man bis zu deren Freigabe in ROONEK. Man halte sich das hinsichtlich des Zeitbedarfes vor Augen. Die menschliche Tastaturbedienung ist angesichts der Prozessorgeschwindigkeit recht langsam. Während der Dauer eines Tastendruckes kann an schnell zu bedienender Peripherie die entscheidende Reaktionsgeschwindigkeit unterschritten werden. In solchen Fällen ist trotz der nachfolgend aufgezeigten Möglichkeit die oben bei GETKEY vorgeschlagene Interruptlösung zu wählen.

Wenn man per ROONEK auf die betätigte Tastatur eingegangen ist, so möchte man im Anschluß daran ja ganz sicher eine Auswertung des Tastendruckes wie in GETKEY haben. Der Einsprungspunkt ist dann GETKY=GETKEY+3 in $EC43.

Beispiel 2.1.6: Abfrage, ob eine Taste betätigt ist. Wenn ja, Verarbeitung bei VERARB. Wenn nein Durchlauf zu WEITER. Die Programmfortsetzung erfolgt auch dann bei VERARB, wenn nur CTRL oder SHIFT betätigt wurde.

```
          ROONEK=$ECEF
          GETKY=$EC43
          *=$200
          JSR ROONEK      EINE TASTE GEDRUECKT?
          DEY
          BMI WEITER      NEIN, Y WAR 00 NACH ROONEK
VERARB    LDX $0          JA
          JSR GETKY       DECODIERE
          BRK
WEITER    BRK             ANDERE VERARBEITUNG
```

Nach dem Programmstart bei $200 halten wir die startende G-Taste noch etwas fest. Wenn wir sie dann freigeben, landen wir auf dem Breakpoint bei WEITER. Wenn wir aber noch eine andere Taste länger betätigen, während G noch gedrückt ist und wenn wir dann G freigeben, landen wir auf dem Breakpoint in $20B, der auf GETKY folgt. An dieser Stelle können wir uns mit dem R-Command (Registeranzeige) wieder den ASCII-Wert des Tastendruckes und des interessierenden X-Registers anzeigen.

Unter dynamischen Programmbedingungen braucht man die Vortaste nicht festzuhalten. Dieser Versuch zeigt: ROONEK kann als Durchlaufroutine benutzt werden, man verläßt sie mit DEY und BMI .. bei unbetätigten Tasten. Bei betätigter Taste besorgt man die ASCII-Decodierung mit JSR GETKY.

Im Zusammenhang mit der Tastatureingabe sind weitere Abschnitte des Monitorprogrammes anzusprechen. Die Routine **RCHEK** ab $E907 ist im Gegensatz zu GETKEY 'offen'. Sie prüft, ob der Benutzer die ESC-Taste betätigt. Wenn ja, erfolgt der Rücksprung zum Monitorprogramm. RCHEK kann von überall her aufgerufen werden, wie folgender Versuch zeigt:

Beispiel 2.1.7: Programmierung der Escape-Taste.

```
RCHEK=$E907
*=$200
JSR RCHEK
JMP $200
```

Die Schleife kann nur mit der ESC-Taste verlassen werden. RCHEK entspricht unserem Beispiel 2.1.6.

Die Aktivierung einer entfernten **Terminaltastatur** erfolgt mit dem Schiebeschalter KB/TTY. Nicht nur im Monitor sondern auch bei Anwenderprogrammen will man vor Eingaben prüfen, welche Tastatur aktiv ist. Dazu dient die Routine TTTYTST in $E842, die die Schalterstellung abfragt. Nach ihrem Durchlauf kann wie folgt verzweigt werden:

BEQ nach TTY-Bedienung
BNE nach AIM-Keyboard.

Auch die RCHEK-Routine macht davon Gebrauch und verzweigt bei eingeschaltetem TTY zu RCHTTY in $E926, um ein evtl. von dort kommendes 'ESC' abzufragen.

Man halte sich vor Augen, daß der Schalter KB/TTY nur Pin PB3 der System-VIA mit dem Pegel LOW oder HIGH belegt. Er hat keinerlei elektrisch umschaltende Funktion. Nichts hindert daher den Anwender, auf TTY auszusenden oder von dort zu empfangen, wenn er das in seiner Software berücksichtigt. Er kann ggfs. sogar softwaremäßige Umschalter zur wechselseitigen Betätigung von Tastaturen einrichten. Unter solchen Bedingungen ist eine Verbindung von zwei oder mehreren AIM per TTY zu verwirklichen, die im stand by-Betrieb abwechselnd die eigene Tastatur per ROONEK auf betätigte Taste abfragen und hernach prüfen, ob die Gegenstation das Startbit sendet. Wir gehen beim TTY-Interface darauf ein.

Zur Tastaturbedienung gehören weitere interessante Teile des Betriebssystems. **READ** in $E93C ist eine Zusammenfassung bisher besprochener Unterprogramme. Die Routine liest die aktive Tastatur (KB oder TTY), kehrt auf Escape zum Monitor zurück, normal wird der Tastenwert als ASCII im Akku retourniert. READ schützt per PHXY und PLXY die Register.

Andere Tastaturroutinen stehen im Zusammenhang mit dem Editor und insbesondere mit der Anzeige auf dem LED-Display. Sie werden dort angesprochen. - Es wurde bereits darauf hingewiesen, daß die E/A am Prozessor vorwiegend in ASCII erfolgt und daß das Display und auch der Drucker 6-Bit ASCII akzeptieren, also 64 Zeichen darstellen können. Nun gibt es ja auch die nichtdruckbaren Zeichen wie $0D, das 'CR' oder $0A, das Linefeed. Im Bereich von hex 00 bis 1F gibt es 32 Kombinationen (nicht druckbarer) Zeichen. Sie werden in der Datenübertragung zu Steuerzwecken verwendet oder zur Steuerung der Funktionen eines Bildschirmterminals, z.B. für die Cursorbewegungen, Löschen einer Zeile oder des Bildschirmes. Fast alle diese Kombinationen werden per GETKEY ganz normal von der AIM-Tastatur geholt, und und zwar beim Betätigen der Taste CTRL zusammen mit einer Zeichentaste. Die auf der folgenden Seite abgedruckte Konbinationstabelle mag für verschiedene Anwendungen nützlich sein.

Erzeugung von Kontroll-Codes mit der AIM-Tastatur

Code $	CTRL + Taste	Code $	CTRL + Taste
00		10	P
01	A	11	Q
02	B	12	R
03	C	13	S
04	D	14	T
05	E	15	U
06	F	16	V
07	G	17	W
08	H	18	X
09	I	19	Y
0A	J	1A	Z
0B	K, auch SHIFT+ESC	1B	F1, auch ESC allein
0C	L	1C	
0D	M, auch CR	1D	F2
0E	N	1E	F3
0F	O	1F	

Die Frage, wie man Zeichen per Tastatur in den Prozessor hineinbekommt, ist mit einem recht ausführlichen Abschnitt zu einem vorläufigen Ende gelangt. Die übrigen Interfaces des AIM wollen nicht weniger aufmerksam bedacht sein. Der Leser möge den Eindruck gewonnen haben, daß der Monitor reiche Möglichkeiten zur Weiterbenutzung in Anwenderprogrammen bietet. Viele interessierende Möglichkeiten der Tastaturabfrage sind noch nicht abgehandelt: Mit CTRL und SHIFT können Kontrollzeichen und Groß- und Kleinschreibung gewonnen werden. Noch nicht dokumentiert wurde das SHIFT-LOCK, die Einrastung der Großschreibung in der anwenderdefinierten Eingabe (siehe dort).

Zu Anfang dieses Abschnittes 2.1 stellten wir heraus, daß das X-Register nach GETKEY mit $20 bzw. $40 gefüllt ist, wenn die linke bzw. die rechte SHIFT-Taste betätigt wurde. Es gibt da also feine Unterschiede, die es uns erlauben, z.B. mit der linken Taste die normale Buchstabenumschaltung zu bewirken und mit der rechten den Übergang auf einen grafischen Modus für ein Videoterminal oder für den Printer mit entsprechendem Grafik-Programm.

Das Experimentierfeld ist noch groß genug. Wer hindert uns, Kontrollzeichen (mit CTRL) über das DILINK zu schicken und damit ein Bank-Select für verschiedene Batterien von Festwertspeichern zu bewirken? Oder auch für RAM-Speicher? Für den Assembler kann man sich auch ohne große Überarbeitung des im ROM befindlichen Übersetzungsprogrammes vorstellen, daß man in den Quelltext Steuerzeichen aufnimmt. Der Assembler liest den Quelltext zweimal durch und pumpt die Zeichen mit dem in $D107 enthaltenen Befehl JSR OUTDP1 ($EF02) über das DILINK. Dort könnte man in Abhängigkeit von den Steuerzeichen z.B. auf eine LIBRARY umschalten und damit die Funktionen .LIB oder auch .MACRO implementieren. Hierzu sei erklärt, daß der Editor die in vorstehender Tabelle aufgeführten Steuerzeichen akzeptiert und auch abspeichert, so daß sie über das DILINK gepumpt werden.

Es sind hier Möglichkeiten gegeben, die die Designer der Firmware sicher noch nicht in Erwägung gezogen hatten. Ihre Öffnung der Ausgabe über das DILINK ist geschickt. Auf der Eingabeseite (Tastatur

etc.) haben sie kein entprechendes Loch gelassen, daher muß man Leistungserweiterungen der Eingabe im allgemeinen durch eine Programmüberlagerung vollziehen.

Den Anwender interessiert immer sehr, wie er auf eine kompromißlose und saubere interaktive Art Anzeigen, Ausgaben erzeugt, nämlich die Benutzerführung, das Echo der Tastaturbetätigung und die Ausgabe von Ergebnissen auf dem LED-Display, dem Printer oder auf einem optionalen Bildschirmmonitor. Davon wird in den nächsten Abschnitten die Rede sein.

R. L.

##########

Ing. grad. Horst Steder, Frankfurt

BASXT - BASIC-Erweiterung (AIM)

Die von Herrn C. Streicher in den Heften 12 und 13 vorgestellte Idee einer BASIC-Erweiterung über die CHAR-Get-Routine durch das Überschreiben mit dem "REM"-Code begeisterte den Verfasser durch ihre Eleganz. Beim Studium des Programms erschienen folgende Punkte verbesserungswürdig und wurden experimentell untersucht:

1) Aussprung in die Abfrage-Routine über den ATN-Vektor
2) Maskierung des USER-Befehls durch "REM"
3) Zeitbedarf der Abfrage-Routine.

Zu 1): Es besteht keine Notwendigkeit, über den ATN-Vektor in $BC/BD in die Abfrageroutine zu springen. Hier bieten sich die Speicherplätze $100 bis $107 (nur für Breakpoints verwendet) an, da sie im Bereich der Branch-Sprungweite liegen. Hier wurden auch die nötigen SAVE-Register und die Flag angesiedelt. Damit ist der direkte Weg des Aufrufs für ATN frei.

Zu 2): Zur Freude des Verfassers stellte sich heraus, daß der "REM"-Code nicht in den BASIC-Buffer eingeschrieben werden muß. Da der Interpreter im nachfolgend vorgestellten Programm mit dem ersten Zeichen nach einem ":" quasi 'stehenbleibt', muß nur dafür gesorgt werden, daß nach der Abarbeitung der entsprechenden USER-Routine der Rücksprung in den Interpreter mit #$8E für "REM" im Akku erfolgt. Das wird im vorliegenden Programm durch die Definition der EXECUTE-Routine als Unterprogramm sichergestellt. Das doch recht aufwendige Abfragen nach dem "REM" und das Überkopieren des USER-Codes kann dadurch entfallen.

Zu 3): Ein wesentlicher Punkt ist die Verlangsamung des Interpreters, da ja im Originalprogramm bei jedem Alphazeichen (und den BASIC-Funktionscodes) die Abfrageroutine durchlaufen wird mit einem Zeitbedarf von min. 60 µsek. Hier wurde vom Verfasser folgender Weg eingeschlagen:

Durch das Setzen einer Flag beim Auffinden eines ":" wird nur das Zeichen unmittelbar hinter dem ":" näher untersucht. Bei allen anderen Zeichen wird nur die Flag getestet. Danach konnten folgende Verzögerungen ermittelt werden:

a) bei Zeichen (nur Flagtest) 20 µsek (3x kürzer)

b) bei Zeichen nach ":" - keine Übereinstimmung mit dem ersten Zeichen des USER-Codes 119 µsek

c) wie b), jedoch bei Übereinstimmung und Test der 2 nächsten Zeichen 93-148 µsek.

Voraussetzung ist allerdings, daß die USER-Codes mit unterschiedlichen Zeichen beginnen. Dadurch kann die Abfrage in 2 Stufen erfolgen (1. Zeichen, dann 2./3. Zeichen), und die Routine dafür wird sehr einfach und kurz.

Jeder USER-Befehl muß aus mindestens 3 Zeichen bestehen, um eventuelle Verwechslungen mit Variablen auszuschließen. Er kann aber darüberhinaus beliebig lang sein, was durch die Art der Zeichensuche mit FNDCHR ermöglicht wird. Durch sie werden alle Zeichen bis zum Auffinden einer definierten Marke (im vorliegenden Fall "(" und "$") einfach überlesen.

Da die Sache großen Spaß machte, wurde auch gleich eine "SYS"-Routine eingebaut. Die Adresse muß allerdings mit "$..." in HEX eingegeben werden. Durch die Routine GETHEX kann die Adresse mit variabler Stellenzahl angegeben werden. Folgende Schreibweisen seien hier angedeutet

:SYS$E9F0 oder :SYS ($100) oder :SYSTEM $EF (Klammern optional).

In die Erweiterung wurde auch gleich der "ATN"-Befehl aufgenommen (direkt aufrufbar). Sie wurde im Listing aus Platzgründen unterdrückt, da sie exakt der im BASIC-Handbuch abgedruckten Befehlsfolge entspricht.

OPEN und CLOSE wurden versuchsweise eingebaut, wobei hier mit dem Device-Code 0 die Monitor-Routinen WHEREI und WHEREO angespungen werden. Hier ist eine Erweiterung um weitere Routinen (z.B. die interessante Tape-Routine von Herrn Streicher) durch Abfrage des Device-Codes und entsprechender Verzweigung leicht möglich und geplant. "CLOSE" wurde übrigens ebenfalls mit Parametern versehen, um das selektive Schließen von Files zu ermöglichen.

Ein kleines Beispielprogramm zeigt die Möglichkeiten einer benutzerfreundlichen Befehlsstruktur durch das USCOM-Verfahren. Die Initialisierungsroutine setzt alle erforderlichen Pointer für ATN und die BASIC-Erweiterung und springt dann direkt über $B003 ins BASIC, was dem Aufruf mit <6> entspricht. Das Programm ist daher in seiner Struktur EPROM-fähig.

Zum Start wird wie üblich BASIC mit <5> initialisiert, mit ESCAPE und dann mit *5000, /G in die Erweiterung eingesprungen. Danach kann wahlweise mit <6> oder über die Startadresse $5000 ins BASIC zurückgesprungen werden. Man kann den Vektor natürlich auch wieder mit POKE auf die Standard-Routine setzen:

```
POKE 203,10  =Setzen des Vektors auf $D6,
POKE 203,52  = Setzen auf USER-Vektor.
```

Der Verfasser hofft, einen kleinen Beitrag und einige Anregungen zum weiteren Esperimentieren gegeben zu haben.

```
0000  0000          ;     ********************************************
0001  0000          ;     *                < B A S X T >             *
0002  0000          ;     *            BASIC - EXTENSION             *
0003  0000          ;     * V 2.4A      COPYRIGHT BY H.STEDER  8/80 *
0004  0000          ;     ********************************************
0005  0000
0006  0000                 *=$103
0007  0103 SAVA           *=*+1
0008  0104 SAVX           *=*+1
0009  0105 SAVY           *=*+1
0010  0106 FLAG           *=*+1
0011  0107
0012  0107                 *=$5000          ;+++ START OF INIT ROUTINE
0013  5000
0014  5000 START  A9DB     LDA #<ATN        ;POINTER TO ATN ROUTINE
0015  5002        85BC     STA $BC
0016  5004        A950     LDA #>ATN
0017  5006        85BD     STA $BD
0018  5008 STRT1  A94C     LDA #$4C         ;JMP INSTRUCTION...
0019  500A        8D0001   STA $100
0020  500D        A923     LDA #<BASXT      ;TO BASIC EXTENSION
0021  500F        8D0101   STA $101
0022  5012        A950     LDA #>BASXT
0023  5014        8D0201   STA $102
0024  5017        A934     LDA #$34         ;CHANGE BRANCH WIDTH IN $CB...
0025  5019        85CB     STA $CB          ;TO BASXT VECTOR IN $100
0026  501B        A900     LDA #0
0027  501D        8D0601   STA FLAG         ;CLEAR USER FLAG
0028  5020        4C03B0   JMP $B003        ;RESTART BASIC AS WITH <6>
0029  5023
0030  5023 BASXT  D006     BNE BASXT1       ;IF IT WAS NOT ":"
0031  5025        08       PHP              ;IT WAS ":", SAVE STATUS
0032  5026        CE0601   DEC FLAG         ;SET FLAG (=FF)
0033  5029        28       PLP
0034  502A        60       RTS
0035  502B
0036  502B BASXT1 08       PHP
0037  502C        2C0601   BIT FLAG
0038  502F        F039     BEQ RTB2         ;FLAG NOT SET, RETURN
0039  5031        8D0301   STA SAVA         ;SAVE REGISTERS
0040  5034        8E0401   STX SAVX
0041  5037        8C0501   STY SAVY
0042  503A        EE0601   INC FLAG         ;CLEAR FLAG
0043  503D TEST1  A005     LDY #5           ;FOR 6 COMMANDS
0044  503F TST1   D97E50   CMP USTBL1,Y
0045  5042        F005     BEQ TEST2        ;MATCH, TEST NEXT 2 CHARS
0046  5044        88       DEY
0047  5045        10F8     BPL TST1         ;GET NEXT CMD
0048  5047        3018     BMI RTB1         ;NO CMD FOUND, RETURN
0049  5049 TEST2  98       TYA              ;MATCH FOUND, CHECK...
0050  504A        0A       ASL A            ;NEXT 2 CHARS
0051  504B        AA       TAX              ;POINTER TO SECOND TABLE
0052  504C        A001     LDY #1
0053  504E        B1C6     LDA ($C6),Y      ;GET 2.CHAR FROM BUFFER
0054  5050        DD8450   CMP USTBL2,X
0055  5053        D00C     BNE RTB1         ;WAS NO REAL CMD
0056  5055        C8       INY
0057  5056        E8       INX
```

```
0058  5057          B1C6   LDA ($C6),Y      ;GET 3.CHAR FROM BUFFER
0059  5059          DD8450 CMP USTBL2,X
0060  505C          D003   BNE RTB1         ;NO REAL CMD
0061  505E          206C50 JSR EXCUTE       ;FOUND A CMD, EXECUTE IT
0062  5061 RTB1    AE0401 LDX SAVX         ;RESTORE REGISTERS
0063  5064          AC0501 LDY SAVY
0064  5067          AD0301 LDA SAVA
0065  506A RTB2    28     PLP
0066  506B RTX     60     RTS
0067  506C
0068  506C EXCUTE  BD8F50 LDA VCTBL-1,X    ;GET VECTOR INTO POINTER
0069  506F          855B   STA $5B
0070  5071          BD9050 LDA VCTBL,X
0071  5074          855C   STA $5C
0072  5076          A98E   LDA ##8E         ;TO RETURN WITH "REM"
0073  5078          8D0301 STA SAVA
0074  507B          6C5B00 JMP ($005B)      ;JUMP IND. TO ROUTINE
0075  507E
0076  507E USTBL1 525A   .BYT 'RZOCSB'
0077  5084 USTBL2 454E   .BYT 'ENLEPELOYSUS'
0078  5090 VCTBL  8151   .WOR RENUM,EDIT,OPNFIL
0078  5092          8251
0078  5094          4851
0079  5096          6051   .WOR CLSFIL,FSUBR,BUSCMD
0079  5098          9C50
0079  509A          8351
0080  509C
0081  509C          ;  =====  ENTRY POINT "SYS"  =====
0082  509C FSUBR   A924   LDA #'$'         ;"$"= START OF ADDR STRING
0083  509E          20CF50 JSR FNDCHR       ;FIND "$" IN STRING
0084  50A1          20A750 JSR GETHEX       ;GET ADDR OF SUBROUTINE
0085  50A4          6C5B00 JMP ($005B)      ;JMP TO SUBROUTINE
0086  50A7
0087  50A7 GETHEX  A900   LDA #0
0088  50A9          855B   STA $5B          ;CLEAR ADDRESS VECTOR
0089  50AB          855C   STA $5C
0090  50AD HEXLP   C8     INY
0091  50AE          B1C6   LDA ($C6),Y      ;GET NEXT CHAR FROM BUFFER
0092  50B0          C930   CMP ##30
0093  50B2          901A   BCC RTG          ;END OF STATEMENT
0094  50B4          207DEA JSR $EA7D        ;CONVERT ASCII TO HEX
0095  50B7          B012   BCS ERR2         ;IF NO HEX CHAR
0096  50B9          A204   LDX #4           ;FOR SHIFT OF 4 BITS
0097  50BB HXLP    065B   ASL $5B          ;SHIFT HEX DIGITS FROM RIGHT..
0098  50BD          265C   ROL $5C          ;TO LEFT INTO $5B & $5C
0099  50BF          B00A   BCS ERR2         ;IF MORE THAN 4 DIGITS
0100  50C1          CA     DEX
0101  50C2          D0F7   BNE HXLP
0102  50C4          055B   ORA $5B          ;MERGE LAST DIGIT...
0103  50C6          855B   STA $5B          ;INTO LOW BYTE
0104  50C8          4CAD50 JMP HEXLP        ;GET NEXT DIGIT
0105  50CB ERR2    4CD0BC JMP $BCD0        ;"SN ERROR"
0106  50CE RTG     60     RTS
0107  50CF
0108  50CF FNDCHR  855A   STA $5A          ;SAVE CHAR TO SEARCH FOR
0109  50D1 FNDC    C8     INY
0110  50D2          B1C6   LDA ($C6),Y      ;SEARCH IN STRING UNTIL..
0111  50D4          F0F5   BEQ ERR2         ;END OF LINE (=ERROR), OR..
```

```
0112  50D6          C55A    CMP $5A          ;DESIRED CHAR IS FOUND
0113  50D8          D0F7    BNE FNDC
0114  50DA          60      RTS
0115  50DB
0116  50DB          ;  =====  ENTRY POINT "ATN"  =====
0117  50DB ATN      A5AE    LDA $AE
0149  5148                  .OPT LIS
0150  5148
0151  5148          ;  =====  ENTRY POINT "OPEN"  =====
0152  5148 OPNFIL 207851 JSR FNDDEV       ;GET DEVICE CODE
0153  514B          C930    CMP #'0'         ;IS IT TO BE DEFINED?
0154  514D          F003    BEQ *+5          ;YES, ASK FOR DEVICE
0155  514F          4CD0BC  JMP $BCD0        ;"SN ERROR"
0156  5152          C8      INY
0157  5153          C8      INY
0158  5154          B1C6    LDA ($C6),Y      ;CODE FOR IN- OR OUTPUT
0159  5156          C94F    CMP #'O'         ;IS IT OUTPUT?
0160  5158          D003    BNE *+5          ;NO, MUST BE INPUT
0161  515A          4C71E8  JMP $E871        ;GET DEVICE IN "WHEREO"
0162  515D          4C48E8  JMP $E848        ;GET DEVICE IN "WHEREI"
0163  5160
0164  5160          ;  =====  ENTRY POINT "CLOSE"  =====
0165  5160 CLSFIL 207851 JSR FNDDEV       ;WHICH DEVICE?
0166  5163          C930    CMP #'0'         ;DEVICE TO BE DEFINED?
0167  5165          F003    BEQ *+5
0168  5167        ' 4CD0BC  JMP $BCD0        ;"SN ERROR"
0169  516A          C8      INY
0170  516B          C8      INY
0171  516C          B1C6    LDA ($C6),Y      ;IN- OR OUTPUT?
0172  516E          C94F    CMP #'O'         ;IS IT OUTPUT?
0173  5170          D003    BNE *+5          ;NO, MUST BE INPUT
0174  5172          4C0AE5  JMP $E50A        ;CLOSE OUTPUT ("DU11")
0175  5175          4C20E5  JMP $E520        ;CLOSE INPUT ("DU13")
0176  5178
0177  5178 FNDDEV   A928    LDA #'('         ;SEARCH FOR THE "("
0178  517A          20CF50  JSR FNDCHR
0179  517D          C8      INY
0180  517E          B1C6    LDA ($C6),Y      ;NEXT CHAR MUST BE DEV CODE
0181  5180          60      RTS
0182  5181
0183  5181          ;  ROUTINES NOT YET IMPLEMENTED
0184  5181 RENUM    60      RTS              ;FOR RENUMBER ROUTINE
0185  5182 EDIT     60      RTS              ;FOR LINE EDITING
0186  5183 BUSCMD   60      RTS              ;FOR IEEE-488 BUS
0187  5184
0188  5184
0189  5184                  .END
```

** SYMBOL TABLE **

ATN	50DB	ATNTB	510B	BASXT	5023	BASXT1	502B
BUSCMD	5183	CLSFIL	5160	EDIT	5182	ERR2	50CB
EXCUTE	506C	FLAG	0106	FNDC	50D1	FNDCHR	50CF
FNDDEV	5178	FSUBR	509C	GETHEX	50A7	HEXLP	50AD
HXLP	50BB	OPNFIL	5148	RENUM	5181	RT1	5104
RTB1	5061	RTB2	506A	RTG	50CE	RTX	506B
SAVA	0103	SAVX	0104	SAVY	0105	START	5000
STRT1	5008	TBPTR	50F1	TEST1	503D	TEST2	5049
TST1	503F	USTBL1	507E	USTBL2	5084	VCTBL	5090

```
6)
LIST

 10 REM  ***  VERSUCH INPUT/OUTPUT VIA TAPE  ***
 20 REM                              H.STEDER  02/80
 30 REM  ======================================
 40 REM
 50 REM  *** OPEN OUTPUT FILE  (WHEREO=$E871)  ***
 60 REM  ---------------------------------------
→70 POKE 4,113 : POKE 5,232 : X=USR(0)
 80 FOR N=1 TO 15
 90 PRINT SQR(N+0.5)
 100 NEXT
 110 REM  *** CLOSE FILE  (DU11=$E50A)  ***
 120 REM  ---------------------------------
→130 POKE 4,10 : POKE 5,229 : X=USR(0)
 140 REM
 150 DIM A(15)
 160 REM  ***  OPEN INPUT FILE  (WHEREI=$E848)  ***
 170 REM  ----------------------------------------
→180 POKE 4,72 : POKE 5,232 :X=USR(0)
 190 FOR I=15 TO 1 STEP-1
 200 INPUT A(I)
 210 NEXT
 220 REM  ***  CLOSE FILE  (DU13=$E520)  ***
 230 REM  ---------------------------------
→240 POKE 4,32 : POKE 5,229 : X=USR(0)
 250 REM
 260 REM  ++++  ARRAY TEST  ++++
 270 REM  ----------------------
 280 FOR N=1 TO 15
 290 PRINT INT(1E4*A(N)+0.5)/1E4,
 320 NEXT


RUN
OUT=T F=TEST T=1
000102
IN=T F=TEST  T=1
00 SRCH F=TEST  BLK=(00) LOAD?  1.22474487
?  1.58113883
?  1.87082869
?  2.12132034
?  2.34520788
?  2.5(01)4950976
?  2.73861279
?  2.91547595
?  3.082207
?  3.24037035
?  3.39116499
?  (02)3.53553391
?  3.67423462
?  3.80788655
?  3.93700394

 3.937     3.8079    3.6742    3.5355    3.3912    3.2404
 3.0822    2.9155    2.7386    2.5495    2.3452    2.1213
 1.8708    1.5811    1.2247
```

(xx) = BLOCKNUMMERN

```
<6>
LIST

 10 PRINT "ATN(1.7)="; ATN(1.7)
 15 PRINT""
 20 :SYS $F704
 30 :SYSTEM-ROUTINE $F704
 40 :SYS ($F704) SHOW "END"
 50 :SYS (CR/LF)= ($E9F0)
 60 :SYS-AIM $E7CA ,DISPLAY "*"
 70 PRINT"":PRINT""
 80 :OPEN DEVICE FILE (0,OUT)
 90 FOR N=1TO50
100 PRINT N
110 NEXT
120 :CLOSE (0,OUT)
130 :OPEN (0,INPUT FILE)
140 FORN=1TO50 :INPUT A :NEXT
150 :CLOSE FILE (0,IN)


RUN
ATN(1.7)= 1.03907226

ENDENDEND
*
```

```
OUT=T F=TEST  T=1
00010203
IN=T F=TEST  T=1
00 SRCH F=TEST  BLK= 00 LOAD?  1
?  2
?  3
?  4
?  5
?  6
?  7
?  8
?  9
?  10
?  11
?  12
```

Michael Zimmermann, Pfungstadt

Generelle Dumpprogramme für breite Printer

1. Einleitung

Für jeden Betreiber des AIM, der von einem kleinen System herkommt und der nur Anzeige oder Bildschirm als einziges Ausgabemedium gewöhnt ist, erscheint der kleine 20-stellige Thermodrucker schon ein großer Fortschritt. Aber beim Arbeiten mit dem AIM ergeben sich in der täglichen Praxis doch Probleme, die dem Betreiber, einmal auf den Geschmack gekommen, den Wunsch nach einem breiteren Normalpapierdrucker nahelegen.

Hier werden Geräte für eine direkte Ansteuerung durch den AIM für unter DM 1000,- angeboten, bei eigener Intelligenz und serieller oder paralleler Schnittstelle für unter DM 2000,-, beides bei einer Druckbreite von 80 Zeichen. Eine Schwachstelle beim Betrieb eines derartigen Druckers ist aber, daß die Druckbreite der Monitorausgaben nach wie vor auf 20 Zeichen beschränkt ist. Dies führt insbesondere bei Speicherausdrucken mit Hilfe des M-Befehles zu einem erhöhten Papierverbrauch und geringer Übersichtlichkeit.

2. Memory-Dump-Programm

Um hier eine Abhilfe zu schaffen, wurden vom Verfasser nachstehende Programme entwickelt, die jeweils 16 Bytes in einer Zeile in hexadezimaler Darstellung als auch in ASCII-Zeichen ausgeben, soweit es sich um druckbare Codes handelt. - In jeder Zeile ist zuerst die Adresse der ersten Speicherzelle, sodann - durch jeweils eine Leerstelle getrennt - der Speicherinhalt in hexadezimaler Form ausgegeben.

Hieran schließt sich der Ausdruck in ASCII-Form an, undruckbare Zeichen werden dabei durch einen Punkt ersetzt. Beginn und Abschluß des Ausdruckes bildet eine Zeile, in der Adressen relativ zum Zeilenanfang gelistet sind.

3. Programmbedienung

Nach Starten des Programms werden FROM- und TO-Adresse über das Display abgefragt, der Benutzer hat hierbei Anfangs- und Endadresse einzugeben, von dem ein Dump gewünscht wird. Zu beachten ist, daß immer eine Normierung an 16-Byte-Grenzen vorgenommen wird.

4. Programmlogik

Wegen des Programmablaufes sei auf das kommentierte Listing verwiesen. Mit dessen Hilfe sollte es auch möglich sein, Anpassungen an andere Drucksysteme, z.B. TTY vorzunehmen.

5. Listprogramm für Dateien

Neben dem Ausdruck von Speicherbereichen ist es in vielen Fällen interessant sich anzusehen, was auf Dateien abgespeichert ist. Bei den AIM-Systemdateien ist das weiter kein Problem. Schwieriger wird es jedoch bei Files, die man als Benutzer selber anlegt und die evtl. noch ASCII-, gepackte und binäre Daten im gleichen Satz enthalten. Derartige Daten entziehen sich z.B. dem Zugriff des Editors total. Es ist also notwendig, hier ein Hilfsmittel zu schaffen, das entsprechende Files anlisten kann. Ein Hilfsmittel soll das hier beschriebene TDUMP sein.

Vom allgemeinen Speicherausdruck (s.o.) unterscheidet es sich nur dadurch, daß nach Einlesen eines Datensatzes immer der gleiche Speicherbereich angelistet werden kann, d.h. der Speicherbereich, in den der Satz eingelesen wurde. Als Format für den Ausdruck wurde hier die schon vom Memory-Dump her bekannte Form gewählt. - Für die 80 Zeichen eines Blocks werden 5 Zeilen benötigt, einen Hinweis auf die Adressen gibt eine Kopfzeile, auf eine weitergehende Anschrift der Adressen kann verzichtet werden. Zusätzlich wird lediglich eine Blocknummer angelistet.

6. Programmbedienung

Für eine Dateianlistung sind weitergehende Funktionen erforderlich, als bei einem Speicherausdruck, bei dem VON- und BIS-Adresse ausreichen. Hier wurden als mögliche Funktionen gewählt:

P Print records
S Skip records
F Find record and print
E End, Return to monitor.

Dem Benutzer wird nach Starten des Programms und nach Ausführung jeweils einer Funktion ein Prompt mit den Möglichkeiten im Display angezeigt. Nach Eingabe des Code-Buchstabens für die gewünschte Funktion wird der Benutzer mit dem Schrägstrich gepromptet. In diesem Falle hat er einzugeben:

Bei P die Anzahl der Sätze, die ausgedruckt werden sollen,
Bei S die Anzahl der Sätze, die übersprungen werden sollen,
Bei F die Nummer des Blockes, der gesucht und gedruckt werden soll.

Als Beispiel sei ein kleiner Dialog abgedruckt, in dessen Folge zuerst 2 Sätzes übersprungen, dann ein Satz gedruckt werden und nach Suchen und Drucken des Blocks 01 auf der nächsten Datei zum Monitor zurück verzweigt wird. Zu beachten ist, daß TDUMP über Dateigrenzen hinweg arbeitet. Sollte es einmal vorkommen, daß keine Datensätze mehr in einer Datei vorhanden sind, um gesucht, gedruckt oder übersprungen zu werden, so hilft nur die RESET-Taste, aber TDUMP ist jederzeit restartbar.

7. Programmlogik

Ebenso wie im vorstehenden MDUMP-Programm sei auch hier auf das kommentierte Programmlisting verwiesen, aus dem die erforderlichen Einzelheiten hervorgehen.

8. Benutzermodifikationen

Die vorstehenden Programme wurden für einen Drucker geschrieben mit Druckprogramm an bestimmten Adressen. Aus diesem Grunde wird jeder Benutzer zumindest die Adresse der PRT-Routine zu ändern haben. Weitere denkbare Modifikationen können sich z.B. aus der Charakteristik des Wagenrücklaufes insbesondere bei seriellen Druckern ergeben. Hier ist die Routine CRLF zu ändern, evtl. Linefeed und Leerzeichen sind zusätzlich mit auszugeben. Verfügt das Druckwerk nicht über die erforderliche Breite, so sind Änderungen denkbar, die durch Variation von LINLEN einmal die Zeilenbreite beschneiden oder wahlweise nur einen hexadezimalen oder einen ASCII-Ausdruck vorsehen.

```
0000
0000          ---------------------
0000          MEMORY-DUMP FOR ADVANCED PRINTERS
0000
0000
0000          ;ID               MDUMP
0000          ;VERSION          1
0000          ;RELEASE          1
0000          ;COPYRIGHT        MICHAEL ZIMMERMANN
0000          ---------------------
0000
0000
0000          ---------------------
0000          MONITOR-ADRESSES
0000 FROM           =$E7A3
0000 TO             =$E7A7
0000 BLANK          =$E83E
0000 RSET           =$E0BF
0000 ADDR           =$A41C
0000          ---------------------
0000          ADRESS OF PRINT-ROUTINE
0000
0000 PRT            =$9E0F   9E93 9A9F
0000          ---------------------
0000          LENGTH OF LINE
0000
0000 LINLEN         =$10
0000          ---------------------
0000          PAGE-ZERO-ADRESSES
0000 PTRF           =$D0            ;POINTER FROM
0000
```

```
0000          ;--------------------
0000          ;MAINLINE-PROGRAMM
0000          ;
0000          ;       *=$200
0200 MA00     ;       *=*
0200 MA01     ;       *=*
0200 20A3E7   ;       JSR FROM        ;GET FROM-ADRESS
0203 B0FB     ;       BCS MA01        ;RETURN IF ERROR
0205 AD1CA4   ;       LDA ADDR        ;TRANSFER ADDRESS TO POINTER
0208 29F0     ;       AND #$F0
020A 85D0     ;       STA PTRF
020C AD1DA4   ;       LDA ADDR+1
020F 85D1     ;       STA PTRF+1
0211 MA02     ;       *=*
0211 203EE8   ;       JSR BLANK       ;DISPLAY BLANK
0214 20A7E7   ;       JSR TO          ;GET TO-ADRESS
0217 B0F8     ;       BCS MA02        ;RETURN IF ERROR
0219          ;--------------------
0219          ;PROCESS AND PRINT ONE LINE
0219          ;                 Bildschirm leeren
0219 209D02   ;       JSR LINE
021C MA10     ;       *=*
021C 208002   ;       JSR CRLF        ;FIRST A CARRIAGE-RETURN'
021F A5D1     ;       LDA PTRF+1      ;PROCESS AND PRINT ADRESS'
0221 208602   ;       JSR NUMA
0224 A5D0     ;       LDA PTRF
0226 208602   ;       JSR NUMA
0229 207702   ;       JSR SPACE2
022C A000     ;       LDY #0          ;CLEAR POINTER
022E MA20     ;       *=*
022E B1D0     ;       LDA (PTRF),Y    ;LOAD CHARACTER
0230 208602   ;       JSR NUMA        ;AND PRINT IT IN HEXA
0233 207A02   ;       JSR SPACE       ;ONE SPACE
0236 C8       ;       INY             ;INCREMENT LINE-POINTER
0237 C010     ;       CPY #LINLEN     ;AND CHECK IF WE ARE THRU WITH
0239 D0F3     ;       BNE MA20        ;WITH ONE LINE
023B 207702   ;       JSR SPACE2
023E A000     ;       LDY #0          ;AND NOW THE SAME IN CHARACTERS
0240 MA25     ;       *=*
0240 B1D0     ;       LDA (PTRF),Y
0242 C91F     ;       CMP #$1F        ;CHECK FOR CONTROLL-CHARCTER
0244 9004     ;       BCC MA27
0246 C960     ;       CMP #$60
0248 9002     ;       BCC MA28
024A MA27     ;       *=*
024A A92E     ;       LDA #'.'        ;MAKE NONPRINTABLE CHARACTER '.'
024C MA28     ;       *=*
024C          ;*200F9E JSR PRT
024F          ;C8     INY
0250          ;C010   CPY #LINLEN
0252          ;D0EC   BNE MA25
0254          ;18     CLC             ;NOW ADD LINELENGTH
0255          ;A5D0   LDA PTRF
0257          ;6910   ADC #LINLEN
0259          ;85D0   STA PTRF
025B          ;9002   BCC MA30
025D          ;E6D1   INC PTRF+1
```

```
025F MA30           *=*
025F        A5D0    LDA PTRF        ;NOW COMPARE WITH TO-ADRESS
0261        CD1CA4  CMP ADDR
0264        A5D1    LDA PTRF+1
0266        ED1DA4  SBC ADDR+1
0269        90B1    BCC MA10
026B        208002  JSR CRLF
026E        209D02  JSR LINE
0271        208002  JSR CRLF
0274        20BFE0  JSR RSET        ;BRANCH BACK TO MONITOR
0277        ********************
0277        ROUTINE TO PRINT SPACES
0277
0277 SPACE2         *=*
0277        207A02  JSR SPACE
027A SPACE          *=*
027A        A920    LDA #' '
027C PRINT          *=*
027C      > 200F9E  JSR PRT
027F        60      RTS
0280        ********************
0280        ROUTINE TO PERFORM A CARRIAGE-RETURN
0280
0280 CRLF           *=*
0280        A90D    LDA #$0D  - 0A
0282      < 200F9E  JSR PRT
0285        60      RTS
0286        ********************
0286        CONVERT BYTE TO 2 HEX-CHARACTERS AND PRINT
0286
0286 NUMA           *=*
0286        48      PHA             ;SAVE ACCU
0287        4A      LSR A           ;NOW SEPARATE HIGH-ORDER-NIBBLE
0288        4A      LSR A
0289        4A      LSR A
028A        4A      LSR A
028B        209102  JSR NOUT        ;AND PROCESS IT
028E        68      PLA             ;GET ACCU AGAIN
028F        290F    AND #$0F        ;AND SEPARATE LOW-ORDER NIBBLE
0291 NOUT           *=*
0291        0930    ORA #$30        ;NOW MAKE NIBBLE A CHARACTER
0293        C93A    CMP #$3A        ;TEST IF NUMBER
0295        9002    BCC NO10
0297        6906    ADC #6          ;ADD 6 FOR CHARACTERS
0299 NO10           *=*
0299      > 200F9E  JSR PRT
029C        60      RTS
029D        ********************
029D        PRINT FIRST AND LAST LINE
029D
029D LINE           *=*
029D        A000    LDY #0
029F L10            *=*
029F        B9AB02  LDA LBUF,Y
02A2      > 200F9E  JSR PRT
02A5        C8      INY
02A6        C04F    CPY #LLBUF
```

```
02A8            D0F5    BNE L10
02AA            60      RTS
02AB LBUF       4144    .BYT 'ADDR  0 1 2 3 4 5 6 7'
02C8            2020    .BYT '  8 9 A B C D E F'
02E0            2020    .BYT '  0123456789ABCDEF      '
02FE LLBUF              =79
02FE                    .END
```

PROBEAUSDRUCK MIT MDUMP

```
ADDR  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
0270  02 20 80 02 20 BF E0 20 7A 02 A9 20 20 CC 9B 60   . . . . . .
0280  A9 0D 20 CC 9B 60 48 4A 4A 4A 4A 20 91 02 68 29   . . . . HJJJJ . . )
0290  0F 09 30 C9 3A 90 02 69 06 20 CC 9B 60 A0 00 B9   . 0 . : . . i . . . . .
02A0  AB 02 20 CC 9B C8 C0 4F D0 F5 60 41 44 44 52 20   . . . . . O . . ADDR
02B0  20 20 30 20 20 31 20 20 32 20 20 33 20 20 34 20     0  1  2  3  4
02C0  20 35 20 20 36 20 20 37 20 20 38 20 20 39 20 20    5  6  7  8  9
02D0  41 20 20 42 20 20 43 20 20 44 20 20 45 20 20 46   A  B  C  D  E  F
02E0  20 20 20 30 31 32 33 34 35 36 37 38 39 41 42 43     0123456789ABC
02F0  44 45 46 20 20 20 20 20 20 20 20 20 20 20 E7 AA   DEF . . . . . .
ADDR  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
```

```
                ;--------------------
0000            ;TAPE-DUMP FOR ADVANCED PRINTERS
0000            ;
0000            ;
0000            ;ID             TDUMP
0000            ;VERSION        1
0000            ;RELEASE        1
0000            ;COPYRIGHT      MICHAEL ZIMMERMANN
0000            ;--------------------
0000            ;
0000            ;
0000            ;--------------------
0000            ;MONITOR-ADRESSES
0000 CLR                =$EB44
0000 OUTDIS             =$EF05
0000 REDOUT             =$E973
0000 BLANK              =$E83E
0000 RSET               =$E0BF
0000 TIBY1              =$ED53
0000 GCNT               =$E785
0000 PSL1               =$E837
0000 DONE               =$E790
0000 BCNT               =$0115
0000 TBUF               =$0116
0000 COUNT              =$A419
0000            ;--------------------
0000            ;ADRESS OF PRINT-ROUTINE
0000            ;
0000 PRT                =$9E0F
0000            ;--------------------
0000            ;LENGTH OF LINE
0000            ;
0000 LINLEN             =$10
0000            ;--------------------
0000            ;PAGE-ZERO-ADRESSES
0000 PTRF               =$D0            ;POINTER FROM
0000            ;
```

```
0000          ;--------------------
0000          ;MAINLINE-PROGRAMM
0000          ;
0000                *=$200
0200 MA00           *=*
0200  20CA02  JSR FUNC        ;GET FUNCTION
0203  C945    CMP #'E'        ;COMPARE END
0205  D003    BNE MA03
0207  4CBFE0  JMP RSET
020A MA03           *=*
020A  C950    CMP #'P'        ;COMPARE PRINT
020C  F003    BEQ MA05
020E  4C7902  JMP MA70        ;SKIP OR FIND
0211 MA05           *=*
0211  209B02  JSR READ        ;READ BLOCK
0214          ;--------------------
0214          ;PROCESS AND PRINT ONE LINE
0214          ;
0214 MA08           *=*
0214  A204    LDX #4          ;LOAD LINE-COUNTER
0216  A916    LDA #<TBUF      ;LOAD TAPE-BUFFER-ADRESS TO POINTER
0218  85D0    STA PTRF
021A  A901    LDA #>TBUF
021C  85D1    STA PTRF+1
021E  200803  JSR LINE        ;PRINT HEADLINE
0221  20AD02  JSR CRLF
0224  AD1601  LDA TBUF        ;PROCESS BLOCK-NUMBER
0227  20B302  JSR NUMA
022A  4C3002  JMP MA11
022D MA10           *=*
022D  20A402  JSR SPACE2
0230 MA11           *=*
0230  20A402  JSR SPACE2
0233  20A402  JSR SPACE2
0236  A000    LDY #0          ;CLEAR POINTER
0238 MA20           *=*
0238  B1D0    LDA (PTRF),Y    ;LOAD CHARACTER
023A  20B302  JSR NUMA        ;AND PRINT IT IN HEXA
023D  20A702  JSR SPACE       ;ONE SPACE
0240  C8      INY             ;INCREMENT LINE-POINTER
0241  C010    CPY #LINLEN     ;AND CHECK IF WE ARE THRU WITH
0243  D0F3    BNE MA20        ;WITH ONE LINE
0245  20A402  JSR SPACE2
0248  A000    LDY #0          ;AND NOW THE SAME IN CHARACTERS
024A MA25           *=*
024A  B1D0    LDA (PTRF),Y
024C  C91F    CMP #$1F        ;CHECK FOR CONTROLL-CHARCTER
024E  9004    BCC MA27
0250  C960    CMP #$60
0252  9002    BCC MA28
0254 MA27           *=*
0254  A92E    LDA #'.'        ;MAKE NONPRINTABLE CHARACTER '.'
0256 MA28           *=*
0256  200F9E  JSR PRT
0259  C8      INY
025A  C010    CPY #LINLEN
025C  D0EC    BNE MA25
```

```
025E          18      CLC             ;NOW ADD LINELENGTH
025F          A5D0    LDA PTRF
0261          6928    ADC #LINLEN
0263          85D0    STA PTRF
0265          9002    BCC MA30
0267          E6D1    INC PTRF+1
0269 MA30             *=*
0269          20AD02  JSR CRLF
026C          CA      DEX             ;DECREMENT LINE-COUNTER
026D          10BE    BPL MA10
026F          20AD02  JSR CRLF
0272          2090E7  JSR DONE
0275          D09A    BNE MA05
0277          F087    BEQ MA00
0279          ------------------
0279          PERFORM FIND OR SKIP
0279
0279 MA70             *=*
0279          C953    CMP #'S'        ;IF SKIP
027B          F013    BEQ MA80        ;GO TO MA80
027D MA72             *=*
027D          209B02  JSR READ        ;READ BLOCK
0280          AD1601  LDA TBUF        ;GET BLOCK-NUMBER
0283          CD19A4  CMP COUNT       ;AND COMPARE IT WITH NUMBER TO FIND
0286          D0F5    BNE MA72        ;NOT FOUND, REPEAT
0288          A901    LDA #1          ;SET COUNTER FOR PRINT
028A          8D19A4  STA COUNT       ;OF 1 RECORD
028D          4C1402  JMP MA08        ;FOUND, PRINT
0290          ------------------
0290          PERFORM SKIP
0290
0290 MA80             *=*
0290          209B02  JSR READ
0293          2090E7  JSR DONE        ;DECREMENT COUNTER
0296          D0F8    BNE MA80        ;NOT DONE, REPEAT
0298          4C0002  JMP MA00        ;DONE, GET NEXT FUNCTION
029B          ------------------
029B          READ ONE BLOCK
029B
029B READ             *=*
029B          A900    LDA #0          ;SET FOR READ ALWAYS
029D          8D1501  STA BCNT        ;BY CLEARING BLOCK-COUNT
02A0          2053ED  JSR TIBY1       ;FILL BUFFER
02A3          60      RTS
02A4          ------------------
02A4          ROUTINE TO PRINT SPACES
02A4
02A4 SPACE2           *=*
02A4          20A702  JSR SPACE
02A7 SPACE            *=*
02A7          A920    LDA #' '
02A9 PRINT            *=*
02A9          200F9E  JSR PRT
02AC          60      RTS
02AD
```

```
                ;--------------------
                ; ROUTINE TO PERFORM A CARRIAGE-RETURN
                ;
02AD            CRLF    *=*
02AD A90D               LDA #$0D
02AF 200F9E             JSR PRT
02B2 60                 RTS
                ;--------------------
                ; CONVERT BYTE TO 2 HEX-CHARACTERS AND PRINT
                ;
02B3            NUMA    *=*
02B3 48                 PHA             ;SAVE ACCU
02B4 4A                 LSR A           ;NOW SEPARATE HIGH-ORDER-NIBBLE
02B5 4A                 LSR A
02B6 4A                 LSR A
02B7 4A                 LSR A
02B8 20BE02             JSR NOUT        ;AND PROCESS IT
02BB 68                 PLA             ;GET ACCU AGAIN
02BC 290F               AND #$0F        ;AND SEPARATE LOW-ORDER NIBBLE
02BE            NOUT    *=*
02BE 0930               ORA #$30        ;NOW MAKE NIBBLE A CHARACTER
02C0 C93A               CMP #$3A        ;TEST IF NUMBER
02C2 9002               BCC NO10
02C4 6906               ADC #6          ;ADD 6 FOR CHARACTERS
02C6            NO10    *=*
02C6 200F9E             JSR PRT
02C9 60                 RTS
                ;--------------------
                ; GET FUNCTION AND COUNT
                ;
02CA            FUNC    *=*
02CA 2044EB             JSR CLR         ;CLEAR DISPLAY
02CD A000               LDY #0          ;OUTPUT PROMPT-TEXT
02CF            F10     *=*
02CF B9F802             LDA FTXT,Y
02D2 2005EF             JSR OUTDIS
02D5 C8                 INY
02D6 C010               CPY #LFTXT
02D8 D0F5               BNE F10
02DA 2073E9             JSR REDOUT      ;GET ANSWER / FUNCTION
02DD C945               CMP #'E'        ;AND CHECK VALID ANSWERS
02DF F016               BEQ F99
02E1 C953               CMP #'S'
02E3 F00A               BEQ F20
02E5 C950               CMP #'P'
02E7 F006               BEQ F20
02E9 C946               CMP #'F'
02EB F002               BEQ F20
02ED D0DB               BNE FUNC        ;NO VALID ANSWER, REDO
02EF            F20     *=*
02EF 48                 PHA
02F0 2037E8             JSR PSL1        ;DISPLAY '/' FOR PROMPT
02F3 2085E7             JSR GCNT        ;AND GET COUNT OR BLOCK-NUMBER
02F6 68                 PLA
02F7            F99     *=*
02F7 60                 RTS
```

```
02F8 FTXT    5444    .BYT 'TDUMP  P/S/F/E ='
0308 LFTXT           =*-FTXT
0308         ---------------------
0308         PRINT FIRST AND LAST LINE
0308
0308 LINE            *=*
0308         A000    LDY #0
030A L10             *=*
030A         B91603  LDA LBUF,Y
030D         200F9E  JSR PRT
0310         C8      INY
0311         C04F    CPY #LLBUF
0313         D0F5    BNE L10
0315         60      RTS
0316 LBUF    424C    .BYT 'BLK=   0  1  2  3  4  5  6  7'
0333         2020    .BYT '  8  9  A  B  C  D  E  F'
034B         2020    .BYT '   0123456789ABCDEF           '
0369 LLBUF           =79
0369                 .END
```

* PROBEAUSDRUCK MIT TDUMP *

```
TDUMP  P/S/F/E =S/02

TDUMP  P/S/F/E =P/01

BLK=   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
02     02 0A 59 0D 3B 18 0E 78 0E A5 D3 A6 D5 F0 0E 46   . . Y . ; . . . . . . . . . F
       D5 C9 2E D0 C2 20 B5 0E A9 3B 4C 93 0E C9 7D D0   . . . . .  . . . ; L . . . . .
       0D 0B 0D 3B 18 0E 90 02 A9 20 20 99 0E A6 D4 60   . . . ; . . . . .  . . . . . `
       20 0F 9E E6 D0 60 A9 0D 20 99 0E A9 00 85 D0 0A     . . . . . `. . . . . . . . .
       80 0D 3B 18 0E A8 85 D2 60 20 91 0E A6 D0 E0 0D   . . ; . . . . . `  . . . . . .

TDUMP  P/S/F/E =F/01

BLK=   0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F   0123456789ABCDEF
01     01 31 0D 53 4D 46 4C 47 20 2A 3D 2A 2B 31 0D 49   . 1 . S M F L G   * = * + 1 . I
       43 4E 54 20 2A 3D 2A 2B 31 0D 50 52 54 20 3D 24   C N T   * = * + 1 . P R T   = $
       39 45 30 46 0D 2A 3D 24 45 30 30 0D 42 43 53 20   9 E 0 F . * = $ E 0 0 . B C S
       54 44 53 50 0D 52 54 53 0D 54 44 53 50 20 2A 3D   T D S P . R T S . T D S P   * =
       2A 0D 43 4C 44 0D 53 54 58 20 53 41 56 58 0D 4C   * . C L D . S T X   S A V X . L

TDUMP  P/S/F/E =E
```

****

Rockwell International hat zur Lieferung ab etwa Ende 1980 das **Miroflex 65-Ausbausystem für den AIM 65** angekündigt, in Eckverbinder- und in Europakartenversion. Es umfaßt Piggy Pack Motherboard für 4 Karten (Kartenkäfig), Adapter/Buffer Modul, Extender-Modul, 1-Karten-Adapter, 8 k statische und 16 k dynamische RAM-Karte, PROM-Programmer, ACIA-Kommunikationskarte.
Die höhere Sprache FORTH, speziell für steuernde Anwendungen geeignet, ist jetzt von Rockwell, CA. für den AIM 65 verfügbar, und zwar als 2 ROMs mit zus. 8 kB. Das interaktive FORTH umfaßt nach hier vorliegender Mitteilung einen Compiler, Assembler, Text-Editor und run time Interpreter.

Dr. Johannes Grabsch

Automatische Zeilennumerierung PET/CBM

Im 65xx MICRO MAG wurde schon eine Reihe von nützlichen Hilfsprogrammen für PET/CBM vorgestellt. Es fehlte noch ein Maschinenprogramm zur automatischen Zeilennumerierung bei der Eingabe von BASIC-Programmen. An anderer Stelle wurden schon Numerierungsprogramme dargestellt, deren umständlicher Aufbau aber nicht befriedigt. Das hier beschriebene Programm 'AUTO' residiert im 2. Kassettenpuffer hex 033A-03D8. Es wird aktiviert durch: SYS (826)N. N ist eine Variable oder Zahl, deren Wert als Start-Zeilenunmmer dienen soll. Wenn nun bei der Eingabe eine Zeile abgeschlossen wurde und die 'RETURN'-Taste bedient wird, erscheint am Anfang der nächsten verfügbaren Zeile die Zeilen'Nr. N+10. Das Inkrement ist durch POKE 906,X (0 <X<256) variierbar. Zum Abschalten wird mehrmals 'CR', dann SYS(826)0 gegeben. Während der Programmeingabe kann durch SYS(826)M eine neue Startnummer M gewählt werden. Vor der Benutzung des Recorders muß AUTO abgeschaltet werden, da der IRQ-Vektor verändert wird.

Zur Funktion: Nach dem Aufruf wird durch die Systemprozeduren PARAM und UMWAND das Argument in SYS(826)N ausgewertet und ins Integerformat umgewandelt. Ist das Argument gleich Null, so wird der IRQ-Vektor in jedem Fall umgeschaltet. Im anderen Fall wird die neue Startnummer gespeichert und im Unterprogramm REENTRY geprüft, ob der IRQ-Vektor bereits umgeändert war. Der geänderte IRQ-Vektor zeigt auf EINSCHUB. Nach einer Stackmanipulation wird geprüft, ob eine neue Zeilennummer ansteht. Ist dies der Fall, so wird die Zeilenummer zunächst durch die Systemroutinen 'SGN' und 'ASCII' ins ASCII-Format umgewandelt und ab $0100 abgelegt. Dieser String wird dann in den Keyboard-Buffer übertragen. Nach der Beendigung des Interrupts wird die Zeilennummer so auf dem Bildschirm ausgegeben. Während der Übertragung in den interruptgesteuerten Puffer müssen die Interrupts gesperrt werden.

Bei $0375 wird der Tastenwert geladen. Wenn eine Taste gedrückt war, wird sie mit dem Tastenwert aus dem letzten Zyklus verglichen. Bei Identität wird der normale Interruptablauf fortgesetzt. Bei Ungleichheit wird der neue Tastenwert für später gerettet und mit dem Tastenwert für 'CR' (hex 1B) verglichen. Liegt 'CR' an, dann wird eine neue Zeilen-Nr. berechnet und ein Flag gesetzt.

Alternativ zum direkten Aufruf mit SYS kann 'AUTO' natürlich auch in einer Erweiterung des Betriebssystems untergebracht werden. Dieser Weg wurde schon in 65xx MICRO MAG beschrieben.

```
##################################################
#                                                #
#       A U T O M A T I S C H E                  #
#       Z E I L E N N U M M E R I E R U N G      #
#       F U E R   P E T - 2 0 0 1                #
#                                                #
#       DR. JOHANNES GRABSCH                     #
#       AM ANGER 14                              #
#       2105 SEEVETAL 2                          #
#                                                #
##################################################
```

```
000                                 [*] = $033A
001  033A  20 A4 CC   AUTO    JSR PARAM       ßPARAMETER LADEN
002  033D  20 D0 D6           JSR UMWAND      ßIN INTEGER UMWANDELN
003  0340  A9 00              LDA #$00
004  0342  85 06              STA FLAG        ßFLAG LOESCHEN
005  0344  8D 0D 02           STA BUFCOUNT    ßKEYBOARDBUFFER LOESCHEN
006  0347  C5 09              CMP NUMH        ßPARAMETER = 0 ?
007  0349  D0 04              BNE LOAD
008  034B  C5 08              CMP NUML        ßLSB TESTEN
009  034D  F0 0B              BEQ BIEGEN      ßWENN 0 VEKTOR IN NORMALSTELLUNG
010  034F  A6 08      LOAD    LDX NUML        ßSTARTNUMMER LADEN
011  0351  A5 09              LDA NUMH        ßUND RETTEN
012  0353  86 00              STX KL
013  0355  85 01              STA KH
014  0357  20 CD 03           JSR REENTRY     ßTEST, OB ERSTER AUFRUF
015  035A  78          BIEGEN  SEI           ßINTERRUPTVEKTOR VERBIEGEN
016  035B  AD 19 02           LDA $0219       ßAUF EINSCHUB
017  035E  49 E8              EOR #$E8        ßBEI $036D
018  0360  8D 19 02           STA $0219
019  0363  AD 1A 02           LDA $021A
020  0366  49 E5              EOR #$E5
021  0368  8D 1A 02           STA $021A
022  036B  58                  CLI           ßINTERRUPTS WIEDER ZULASSEN
023  036C  60                  RTS
024  036D  20 A4 03   EINSCHUB JSR POP        ßSTACK BEHANDELN
025  0370  EA                  NOP
026  0371  A6 06              LDX FLAG        ßFLAG TEST: ZULEZT EIN [CR] ?
027  0373  D0 21              BNE EIN3        ßWENN JA, ZEILENNR. AUSGEBEN
028  0375  AD 03 02           LDA TASTE       ßTASTENCODE HOLEN
029  0378  C9 FF              CMP #$FF        ßKEINE TASTE GEDRUECKT ?
030  037A  F0 17              BEQ EIN2        ßWENN JA, WEITER IN INT.ROUTINE
031  037C  C5 07              CMP ALTE        ßIDENTISCH MIT ALTER TASTE ?
032  037E  F0 13              BEQ EIN2        ßWENN JA, WEITER IN INT.ROUTINE
033  0380  85 07              STA ALTE        ßTASTENWERT RETTEN
034  0382  C9 1B              CMP #$1B        ßIST ES EIN [CR] ?
035  0384  D0 0D              BNE EIN2        ßNEIN: WEITER IN INT.ROUTINE
036  0386  18                  CLC           ßES WAR EIN [CR]
037  0387  A5 00              LDA KL          ßNEUE ZEILENR.
038  0389  69 0A              ADC #10         ßINKREMENT 10 ADDIEREN
039  038B  85 00              STA KL
040  038D  90 02              BCC EIN1        ßPAGE CROSSING ?
041  038F  E6 01              INC KH
042  0391  E6 06      EIN1    INC FLAG        ßFLAG SETZEN. NEUE
043                                           ßZEILENNR. STEHT AN.
044  0393  4C 7E E6   EIN2    JMP $E67E       ßIN INT.ROUTINE FORTFAHREN
045  0396  A2 00      EIN3    LDX #$00        ßFLAG LOESCHEN
046  0398  86 06              STX FLAG
047  039A  A6 00              LDX KL          ßZEILENNR. LADEN
048  039C  A5 01              LDA KH          ßUND IN DEN
049  039E  20 AD 03           JSR INBUF       ßKEYBOARDBUFFER BRINGEN
050  03A1  4C 7E E6           JMP $E67E       ßWEITER IN INT.ROUTINE
051  03A4  A9 00      POP     LDA #$00        ßSTACK VORBEREITEN
052  03A6  48                  PHA
053  03A7  48                  PHA
054  03A8  48                  PHA
055  03A9  48                  PHA
056  03AA  4C 85 E6           JMP $E685       ßANFANG DER INT.ROUTINE
```

```
057  03AD  85 B1     INBUF   STA $B1          ßZEILENNR. IN ASCII
058  03AF  86 B2             STX $B2          ßUMWANDELN
059  03B1  A2 90             LDX #$90
060  03B3  38                SEC
061  03B4  20 1B DB          JSR SIGN
062  03B7  20 AF DC          JSR ASCII        ßASCIIDARSTELLUNG NACH $0100 FF.
063  03BA  A2 00             LDX #$00         ßZEICHENZAEHLER AUF 0
064  03BB  78                SEI              ßINTERRUPTS SPERREN
065  03BD  BD 01 01  READ    LDA $0101,X      ßCHAR. LADEN
066  03C0  F0 06             BEQ EXIT         ßENDE DES STRINGS IN $00
067  03C2  9D 0F 02          STA BUFFER,X     ßIN DEN KEYBOARDBUFFER SCHREIBEN
068  03C5  E8                INX
069  03C6  D0 F5             BNE READ
070  03C8  8E 0D 02  EXIT    STX BUFCOUNT     ßLAENGE IN PUFFERZAEHLER
071  03CB  58                CLI              ßINTERRUPTS ZULASSEN
072  03CC  60                RTS
073  03CD  AD 1A 02  REENTRY LDA $021A        ßMSB DES INT.VEKTORS LADEN
074  03D0  CP 03             CMP #$03         ßSTEHT ER AUF 'EINSCHUB' ?
075  03D2  F0 04             BEQ CLEAN
076  03D4  C9 E6             CMP #$E6         ßODER IN DER NORMALSTELLUNG ?
077  03D6  F0 02             BEQ RETU
078  03D8  68        CLEAN   PLA              ßSCHON VORBEREITET !
079  03D9  68                PLA
080  03DA  60        RETU    RTS
```

SYMBOLTABELLE

ALTE	0007	ASCII	DCAF	AUTO	033A
BIEGEN	035A	BUFCOUNT	020D	BUFFER	020F
CLEAN	03D8	EINSCHUB	036D	EIN1	0391
EIN2	0393	EIN3	0396	EXIT	03C8
FLAG	0006	INBUF	03AD	KH	0001
KL	0000	LOAD	034F	NUMH	0009
NUML	0008	PARAM	CCA4	POP	03A4
READ	03BD	REENTRY	03CD	RETU	03DA
SIGN	DB1B	TASTE	0203	UMWAND	D6D0

****

Dipl.-Math. A. Quindt, Wiesbaden

Auffinden einer BASIC-Variablen

Bei vielen Programmen, die in Assembler geschrieben wurden und die auf dem CBM 3001 laufen, findet man die Festlegung, daß eine bestimmte Variable als erste Variable definiert werden muß. Im folgenden Artikel wird gezeigt, wie man diese Einschränkung fallen lassen kann.

BASIC-Variablen benötigen im CBM 3001 immer 7 Bytes, unabhängig davon, ob sie vom Typ Gleitkomma, Integer oder String sind. Die beiden ersten Bytes werden benötigt, um den Namen der Variablen und den Typ zu verschlüsseln. Dies geschieht folgendermaßen:

Gleitkomma — Die Variable A wird zu $4100,
die Variable B5 zu $4235
($41=A, $42=B, $35=5).

Integer — Zum 1. und 2. Byte werden $80 addiert:
A% wird zu $C180, B5% zu $C2B5.

Strings	Zum zweiten Byte werden $80 addiert: A$ wird zu $4180, B5$ zu $42B5.

Die Bytes 3 bis 7 enthalten dann folgende Informationen:

Gleitkomma	Alle 5 Bytes werden benötigt. Dabei enthält Byte 3 den Exponenten.
Integer	Im 3. und 4. Byte wird die Zahl dargestellt (HI,LO). Die übrigen Bytes sind Null.
String	Das 3. Byte enthält die Länge des Strings, das 4. und 5. Byte die Startadresse (LO,HI). Die übrigen Bytes sind Null.

Will man nun eine Variable suchen, so müßte man mit einer Schrittweite von 7 Bytes den Bereich der Variablen durchsuchen. Dieser Bereich wird begrenzt durch die Pointer in den Zellen $002A und $002B (=dezimal 42 und 43, Beginn der Variablen) und $002C und $002D (=dezimal 44 und 45, Beginn der Felder).

Eine Suchroutine muß nicht von jedem Anwender in Assembler programmiert werden, da der BASIC-Interpreter ebenfalls Variable suchen muß. Diese Suchroutine benutzt folgende Speicherplätze: Name der Variablen in dez. 66 und 67 =$0042/$0043, Adresse der Variablen in dez. 92 und 93 =$005C/$005D. Die Routine beginnt mit dez. 53197 = $CFCD. Ein Programm zum Suchen der Variablen A könnte damit so aussehen:

```
LDA #$C1
STA $42
LDA #$00
STA $43
JSR $CFCD
```

Der Zugriff auf die gespeicherte Information kann damit erfolgen über

```
LDA ($5C),Y
```

wobei im Y-Register Werte von 2 bis 6 stehen können.

Mit dieser kleinen Routine ist die Beschränkung, eine Variable als erste definieren zu müssen, aufgehoben worden. Diese Routine findet eine Variable nur dann, wenn diese vorher benutzt worden ist.

****

Dipl.-Ing. Uwe Kornnagel, Raunheim

Dekadischer Logarithmus per USR

Das nachfolgende Maschinenprogramm gestattet es, den im CBM-BASIC nicht implementierten Logarithmus zur Zahlenbasis 10 zu berechnen, und zwar durch Übergabe des Argumentes in der USR-Funktion in der Form Y=USR(X). Diese neue Funktion kann sowohl im direkten als auch im Programm-Modus aufgerufen werden, nachdem man den Vektor des Programms LGT durch SYS826 für die USR-Routine initialisiert hat. Das Programm stützt sich weitgehend auf die im Interpreter enthaltenen ausführenden Routinen.

Beispiel zu USR ------ Y = LGT(X) Y = USR(X)

Adr.	Code	LBL	MEM	Coment

```
                                !USR - lg (x)
                                .BA 826
                                .LS
                                .OS
                        USR     .DE 1
                        FAC1    .DE $5E
                        FAC2    .DE $66
                        LOG     .DE $D8F6
                        DIV     .DE $DA20
                        AKKUEX  .DE $DB18
                        LDFAC1  .DE $DAAE
033a  a9  43            init    lda #l,lgt      INITIALISIEREN USR
033c  85  01                    sta USR
033e  a9  03                    lda #h,lgt
0340  85  02                    sta USR + 1
0342  60                        rts
0343  20  f6  d8        lgt     jsr LOG         A = ln(x)
0346  a2  06                    ldx #6
0348  b5  5e            trlop   lda FAC1,x
034a  9d  80  03                sta fach,x
034d  ca                        dex
034e  10  f8                    bpl trlop
0350  a9  00                    lda #0
0352  a2  06                    ldx #6
0354  95  5e            zr1     sta FAC1,x
0356  ca                        dex
0357  10  fb                    bpl zr1
0359  a9  84                    lda # $84       konstante 10 in akku 1
035b  85  5e                    sta FAC1
035d  a9  a0                    lda #$a0
035f  85  5f                    sta FAC1 + 1
0361  20  f6  d8                jsr LOG         B = ln(10)
0364  20  18  db                jsr AKKUEX
0367  a9  80                    lda #l,fach
0369  a0  03                    ldy #h,fach
036b  20  ae  da                jsr LDFAC1
036e  a2  06                    ldx #6
0370  b5  5e            exlo    lda FAC1,x
0372  a8                        tay
0373  b5  66                    lda FAC2,x
0375  95  5e                    sta FAC1,x
0377  98                        tya
0378  95  66                    sta FAC2,x
037a  ca                        dex
037b  10  f3                    bpl exlo
037d  4c  20  da                jmp DIV         C = A/B
0380  00  00  00        fach    .by 00 00 00
0383  00  00  00                .by 00 00 00
0386                            .en
```

Dipl.-Math. A. Quindt, Wiesbaden

GARBAGE COLLECTION-ROUTINE IM CBM

Bei einem Buchhaltungsprgramm trat folgender Effekt auf: Der Computer war für etliche Minuten völlig tot. Er reagierte auch nicht auf die RUN/STOP-Taste. Sobald er sich wieder meldete, arbeitete er wie gewohnt weiter. Im Programm wurden 600 Strings über ein Feld definiert. In diesem Feld standen sortiert die Kontonummern der vorhandenen Konten. Außerdem wurde eine größere Zahl einzelner Strings definiert. Einer der Strings diente zur Druckaufbereitung. Ihm wurde sehr oft ein neuer Wert zugewiesen, dabei wurde er durch Blanks und durch RIGHT$ auf eine vorgegebene Länge gebracht, damit der Ausdruck die gedachte Form bekam. - Obwohl es in einem solchen Fall sinnvoll ist, eine andere Dateiverwaltungsform zu wählen, soll hier dargestellt werden, wie man doch noch auf akzeptable Rechenzeiten kommen kann.

Im CBM 3001 werden Strings bei Veränderungen nicht überspeichert, sondern neu angelegt. Es muß also Pointer geben, die den Bereich für die Neuanlage eines Strings angeben ($2E ff.). Außerdem muß eine Routine vorhanden sein, die die "Leichen" entfernt und die den Stringbereich zusammenschiebt (garbage collection routine, beginnt in $D400). Das folgende Programm zeigt den zeitlichen Aufwand dieser Routine:

```
10 N=0:T1=0
20 INPUT"ANZAHL STRINGS";N
30 DIM A$(N-1)
40 FOR I=1 TO N-1
50 A$(I)="1234567890"+""
60 NEXT I
70 T1=T1:PRINT FRE(0),T1-T1
```

Die Anweisung +"" in der Zeile 50 bewirkt, daß jeder String im Speicher des CBM getrennt angelegt wird. - Schreibt man die Zeile 50 in der Form

```
50 A$(I)="1234567890",
```

so enthält jedes Element von A$(.) einen Pointer in das Programm, dh. jeder String von A$(.) wird dargestellt durch die selben 10 Bytes, die außerdem auch noch im Programm stehen.

Das Testprogramm liefert die Zeit für die Abfrage des noch freien Speicherplatzes. Diese Frage bewirkt einen Aufruf der garbage collection-Routine.

Mit N=320 werden 520 Einheiten benötigt (60 Einheiten=1 Sek.)
N=640 werden 2000 Einheiten und mit
N=1280 werden 8000 Einheiten (über 2 Min.) benötigt.

Aus den Werten erkennt man, daß diese Routine quadratisch ist, dh. bei doppelter Anzahl zu untersuchender Strings benötigt sie etwa die vierfache Zeit.

Da bei dem Buchhaltungsprogramm über 600 Strings definiert waren, mußte ein Ausweg gefunden werden. Der Ausweg liegt in der Verwendung einer Puffervariablen. Diese wird im folgenden Programm BB$ genannt. Die Probleme wurden verursacht durch die Variable AA$, deren häufige Manipulation den Speicherbereich verkleinerte und zum Aufruf der garbage collection-Routine führte. Diese Variable soll jetzt in den Stringbereich der Variablen BB$ kopiert werden.

Die Variable BB$ muß also eine Länge haben, die immer größer ist als die größtmögliche Länge von AA$. Damit könnte man die Länge von BB$ auf 255 setzen, ein kleinerer Wert erwies sich aber wegen der Papierbreite als ausreichend.

Man muß jetzt vor den Manipulationen an AA$ die entsprechenden Pointer retten. Nach den Manipulationen muß man prüfen, ob diese eine garbage collection-Routine erzwungen haben. Falls dieses geschehen ist, unternimmt man nichts. War dies nicht der Fall, so kopiert man den String AA$ in den String BB$ und stellt die Pointer auf den Beginn der Strings auf den alten Wert zurück. Die folgenden Programme zeigen einen Weg dazu:

```
00 REM********************************
10 REM**                            **
20 REM**  PGM: GARBAGE COLLECTION   **
30 REM**                            **
40 REM**  PROGRAMM VON              **
50 REM**  A. C. QUINT               **
60 REM**                            **
70 REM**  DIESES PROGRAMM ZEIGT,    **
80 REM**  WIE MAN UNTER VERWEN-     **
90 REM**  DUNG EINER PUFFERVARI-    **
00 REM**  ABLEN RECHENZEIT EIN-     **
10 REM**  SPAREN KANN.              **
20 REM**  ENTFERNT MAN DIE ZEILEN   **
30 REM**  1650 UND 1800,            **
40 REM**  SO VERLAENGERT SICH DIE   **
50 REM**  RECHENZEIT ETWA UM        **
60 REM**  EINEN FAKTOR 10.          **
70 REM**                            **
80 REM********************************
000 REM FELD ERZEUGEN
050 DIMA$(1000)
100 FORI=0TO1000
150 A$(I)="0123456789"+""
200 REM +"" BEWIRKT DIE GETRENNTE ANLAGE JEDES EINZELNEN ELEMENTS DES FELDES
250 NEXTI:PRINT"BEGINN DER ARBEIT"
300 REM PUFFERVARIABLE DEFINIEREN
350 BB$=""
400 REM ZEIT STOPPEN
450 T1=TI
500 REM OPERATIONEN DURCHFUEHREN
550 M=35:FORI=1TO500
600 AA$="123456"
650 SYS634:REM POINTER RETTEN
700 AA$="        "+AA$
750 IFLEN(AA$)<MGOTO1700
800 SYS645:REM VARIABLE UMSPEICHERN IN PUFFERBEREICH
850 NEXTI
900 PRINT"BENOETIGTE ZEIT =";TI-T1
EADY.
```

Hängt man an die Zeile 1550 die Anweisung

:PRINT I

an, so kann man auf dem Bildschirm erkennen, bei welchem Durchlauf das Programm ist. Mit dem Assemblerprogramm läuft es durch, ohne Assemblerprogramm (also ohne die Zeilen 1650 und 1800) stoppte es 3mal, um in die garbage collection-Routine zu verzweigen. Die Unterschiede in der Rechenzeit lagen bei 1580 Einheiten zu 16000 Einheiten.

Es erscheint nicht sinnvoll, ein allgemeines Programm zu schreiben, das Strings im CBM 3001 immer überspeichert. - In einem Programm führt die Anweisung

(Zeilen-Nr) A$=B$

dazu, daß die entsprechenden Pointer von A$ auf den Beginn des Strings B$ gestellt werden, ohne daß der String neu angelegt oder umgespeichert wird. Damit wird Rechenzeit und Speicherplatz gespart. Fügt man ein Programm ein, daß veränderte Strings zurückspeichert (unter Beachtung der Länge), so würde durch eine Veränderung von A$ auch ungewollt B$ mitverändert. Diesen Effekt kann man durch folgendes Programm erzeugen:

```
10 B$="1234"
20 A$=B$
```

Startet man das Programm mit RUN und ändert danach über den TIM-Monitor den Speicherplatz $0409 in $30 ab (bzw. entprechend durch POKE 1033,48), so erzielt man den beschriebenen Effekt.

```
027A  A2 03     LDX  #03
027C  B5 30     LDA   30,X          POINTER RETTEN, ABLEGEN
027E  9D E6 02  STA   02E6,X        IN $02E6 FF.
0281  CA        DEX
0282  10 F8     BPL   027C
0284  60        RTS
0285  AD E7 02  LDA   02E7
0288  C5 31     CMP   31            PRÜFEN AUF DURCHGEFÜHRTE
028A  B0 01     BCS   028D          GARBAGE-COLLECTION
028C  60        RTS
028D  A9 42     LDA  #42            SUCHEN DER VARIABLEN BB$
028F  85 42     STA   42
0291  A9 C2     LDA  #C2
0293  85 43     STA   43
0295  20 C9 CF  JSR   CFC9
0298  A5 5C     LDA   5C            ADRESSE VON BB$ IN $01,02
029A  85 01     STA   01            SPEICHERN
029C  A5 5D     LDA   5D
029E  85 02     STA   02
02A0  A0 01     LDY  #01            BEGINN DES STRINGS VON BB$
02A2  B1 44     LDA  (44),Y         SPEICHERN IN $16, $17
02A4  85 16     STA   16
02A6  C8        INY
02A7  B1 44     LDA  (44),Y
02A9  85 17     STA   17
02AB  A9 41     LDA  #41            SUCHEN DER VARIABLEN AA$
02AD  85 42     STA   42
02AF  A9 C1     LDA  #C1
02B1  85 43     STA   43            ADRESSE VON AA$ BLEIBT IN
02B3  20 C9 CF  JSR   CFC9          $5C, $5D
```

```
02B6  A0 00     LDY  #00
02B8  B1 44     LDA  (44),Y        BEGINN DES STRINGS AA$
02BA  85 4F     STA   4F           SPEICHERN IN $4D, $4E,
02BC  C8        INY                LAANGE IN $4F
02BD  B1 44     LDA  (44),Y
02BF  85 4D     STA   4D
02C1  C8        INY
02C2  B1 44     LDA  (44),Y
02C4  85 4E     STA   4E
02C6  C8        INY
02C7  B1 01     LDA  (01),Y        POINTER AUF BEGINN DES STRINGS
02C9  91 5C     STA  (5C),Y        BB$ UNTER AA$ ABSPEICHERN
02CB  C8        INY
02CC  B1 01     LDA  (01),Y
02CE  91 5C     STA  (5C),Y
02D0  A0 00     LDY  #00
02D2  B1 4D     LDA  (4D),Y        STRING KOPIEREN
02D4  91 16     STA  (16),Y
02D6  C8        INY
02D7  C4 4F     CPY   4F
02D9  D0 F7     BNE   02D2         LAENGE  PRÜFEN
02DB  A2 03     LDX  #03
02DD  BD E6 02  LDA   02E6,X       POINTER RÜCKSPEICHERN
02E0  95 30     STA   30,X
02E2  CA        DEX
02E3  10 F8     BPL   02DD
02E5  60        RTS
```

****

Wolfgang E. Müller, Berlin 37

Rechtsbündige Zahlenausgabe

Für eine Zahl n, die unter BASIC im Bereich 0.01 < ABS(n) < 999999999 im Festkommaformat ausgegeben wird, kann mit Hilfe von Stringfunktionen eine tabellengerechte Anordnung erreicht werden, bei der das Komma stets in der gleichen Spalte steht. In nachstehendem Beispiel wird außerdem eine Rundung auf 2 Stellen vorgenommen. Es mag als Unterprogramm für BASIC nützlich sein:

```
10 REM RECHTSBUENDIGE ZAHLENAUSGABE, GERUNDET
20 INPUT X
30 X=INT(X*100+0.5)/100
40 Z=LEN(STR$(INT(ABS(X))))
50 IF ABS(X)<1 THEN Z=Z-1:IF X=0 THEN Z=2
60 PRINT SPC(12-Z):X
```

BEISPIELE:

```
    7777
     777.7
      77.77
       7.78
         .78
         .08
         .01
        -.01
        -.08
        -.78
       -7.78
```

Peter W. Arps, Hamburg 73

Sichern und Laden dimensionierter Variablen

Dieses Programm sichert auf dem PET dimensionierte BASIC-Variable (VARSAV) und stellt es per VARLD einem anderen Programm oder dem ursprünglichen in einem späteren Durchlauf wieder zur Verfügung. Da die gesicherten Variablen immer an ein Programm angehängt werden, können die Programme unterschiedliche Länge haben. Diese Leistung gilt auch für die etwas schwierigeren Stringvariablen, deren Inhalt während eines Programmdurchlaufes ja verstreut im Speicher abgelegt wird. Durch den Aufruf JSR KOFFER ($D404) ist das möglich, er packt die Strings zu einem Block. Das Vorbild dieses Programmes könnte auch auf einfache Variable angewandt werden.

Man beachte, daß das Nachfolgeprogramm nicht noch einmal eine Dimensionierung (DIM) enthält. Die Syntax des Aufrufes SYS (xxx)prm entspricht hinsichtlich der Parameterübergabe prm dem normalen SAVE oder LOAD mit File usw. Namen. Das Programm eignet z.B. zur Ablage von Bearbeitungssummen/Zwischenergebnissen in kaufmännischen Anwendungen mit späterer Weiterverarbeitung oder zur späteren Überarbeitung von Adreßdateien usw..

```
832               ;
832               ;  V A R S A V L D
832               ;
832               ;
832               ;  SICHERN UND LADEN DIMENSIONIERTER VARIABLEN
832               ;
832               ;  SAVE:  SYS(832)prm
832               ;
832               ;  LOAD:  SYS(873)prm
832               ;
832               ;  prm = PARAMETER FUER SAVE/LOAD
832               ;
832               ;
832               FNLEN  =    $EE
832               SEC    =    $F0
832               VON    =    $F7
832               BIS    =    $E5
832               TOS    =    $84
832               ARRA   =    $7E
832               ARRE   =    $80
832               VERCK  =    $020B
832               TAPE   =    $027A
832               ST     =    $020C
832               FETCH  =    $F433
832               KOFFER =    $D404
832               SAVE   =    $F6B1
832               SETTB  =    $F667
832               PRMSG1 =    $F83B
832               PRMSG2 =    $F3FF
832               PRMSG3 =    $F422
832               SEARCH =    $F495
832               NFOUND =    $F579
832               NEXTFL =    $F5AE
832               COPY   =    $F64D
832               READ   =    $F88A
```

```
B32                    TWAIT  =    $F913
B32                    ERR    =    $F3DB
B32                    ;
B32                    ;
B32  20 04 D4  VARSAV  JSR KOFFER     ;STRINGS ZUSAMMENPACKEN
B35  20 33 F4          JSR FETCH      ;PARAMETER HOLEN GGF. DEFAULT-WERTE
B38  8A                TXA            ;3.PARAMETER SICHERN
B39  48                PHA
B40  A9 00             LDA #0         ;GGF. EOT UNTERDRUECKEN
B42  85 F0             STA SEC
B44  A2 7E             LDX #ARRA
B46  20 56 03          JSR LOOP1      ;SAVE VON ARRA BIS ARRE
B49  68                PLA
B50  85 F0             STA SEC        ;RESTORE SEC.ADR
B52  A2 84             LDX #TOS       ;SAVE TOP OF STRING BIS TOP OF MEMORY
B54  B5 00     LOOP1   LDA 0,X
B56  85 F7             STA VON
B58  B5 01             LDA 1,X
B60  85 F8             STA VON+1
B62  B5 02             LDA 2,X
B64  85 E5             STA BIS
B66  B5 03             LDA 3,X
B68  85 E6             STA BIS+1
B70  4C B1 F6          JMP SAVE       ;SAVE IT
B73                    ;
B73                    ;
B73  20 33 F4  VARLD   JSR FETCH      ;PARAMETER FUER LOAD
B76  20 AF 03          JSR UP1        ;FILE SUCHEN
B79  AD 7D 02          LDA TAPE+3     ;LAENGE DES FILES ERRECHNEN
B82  38                SEC
B83  ED 7B 02          SBC TAPE+1
B86  AA                TAX
B87  AD 7E 02          LDA TAPE+4
B90  ED 7C 02          SBC TAPE+2
B93  A8                TAY
B94  A5 7E             LDA ARRA       ;ALTE VON-LADEADR MODIFIZIEREN,
B96  8D 7B 02          STA TAPE+1     ;DAMIT FILE AN DAS PGM ANGEHAENGT
B99  A5 7F             LDA ARRA+1     ;WIRD
B01  8D 7C 02          STA TAPE+2
B04  18                CLC
B05  8A                TXA
B06  65 7E             ADC ARRA       ;NEUE BIS-LADEADR ERRECHNEN
B08  85 80             STA ARRE       ;= NEUE ADR ARRAY-ENDE
B10  8D 7D 02          STA TAPE+3
B13  98                TYA
B14  65 7F             ADC ARRA+1
B16  85 81             STA ARRE+1
B18  8D 7E 02          STA TAPE+4
B21  20 CA 03          JSR UP2        ;LADE FILE
B24  20 AF 03          JSR UP1        ;FILE MIT STRINGS SUCHEN
B27  AD 7B 02          LDA TAPE+1     ;ADR TOP OF STRING VERAENDERN
B30  85 84             STA TOS
B32  AD 7C 02          LDA TAPE+2
B35  85 85             STA TOS+1
B37  20 CA 03          JSR UP2        ;STRINS LADEN
B40  4C 04 D4          JMP KOFFER
B43                    ;
```

```
943  20 67 F6   UP1    JSR SETTB      ;SET BUFFER-POINTER
946  20 3B F8          JSR PRMSG1     ;PRINT 'PRESS PLAY ....
949  20 FF F3          JSR PRMSG2     ;PRINT FILE-NAME
952  A5 EE             LDA FNLEN      ;LAENGE FILE-NAME
954  F0 08             BEQ ANYFL      ;OHNE  -  NEXT FILE LADEN
956  20 95 F4          JSR SEARCH     ;SUCHE FILE
959  D0 08             BNE OK
961  4C 79 F5   NOTF   JMP NFOUND     ;FILE NOT FOUND ERROR
964  20 AE F5   ANYFL  JSR NEXTFL     ;SUCHE NEXT FILE
967  F0 F8             BEQ NOTF
969  60         OK     RTS
970             ;
970  20 4D F6   UP2    JSR COPY       ;COPY NEUE ADR FUER LOAD
973  A2 00             LDX #0
975  8E 0B 02          STX VERCK      ;IST LOAD
978  20 22 F4          JSR PRMSG3     ;PRINT 'LOADING'
981  20 8A F8          JSR READ       ;READ FILE
984  20 13 F9          JSR TWAIT
987  AD 0C 02          LDA ST         ;STATUS LADEN
990  F0 E9             BEQ OK
992  4C DB F3          JMP ERR        ;LOAD ERROR
995             ;
995                    .END
```

Branchen-Nachrichten

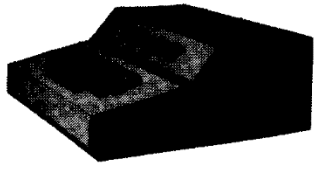
Die **GWK-Elektronik** in Herzogenrath ergänzt ihr Angebot für Hard- und Software weiter. Für OEM-Zwecke ist jetzt eine 6502-CPU-Karte lieferbar, die weitgehend zum AIM kompatibel ist. Für den AIM 65 wird neben der schon bekannten BASIC-Erweiterung eine Reihe von Dienstprogrammen offeriert: Assembler-Formatierer, interaktiv geführte Laderoutine mit Verschiebung usw.. Für Betrieb auf dem GWK Foppy Disk-System kann ein kaufmännisches Fakturierprogramm bezogen werden mit Informationsruf unter Kunden- und Artikelnummer, Zahlungsbedingungen. Nach Eingabe der Stückzahl und ggfs. Nachlässen wird der Warenwert je Zeile und der Rechnung sowie die MWSt berechnet. - Möglichst noch zur Electronica will man ein eigenes BASIC für den AIM vorstellen. Info: Tel.: 02406-62 394.

* * *

Das Haus Siemens hat die Freigabe des in Heft 13 besprochenen PC 1000 einstweilen zurückgestellt.

* * *

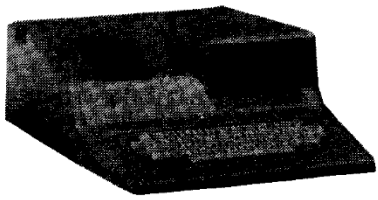
Beim Selbstbau und bei der Ergänzung von Computern tritt regelmäßig die **Gehäusefrage** auf. Das von der Firma **Bündoplast** beziehbare bopla-Gehäuse BP690 bietet mit seinen Abmessungen 460x525x170 mm reichlich Platz für Computer, Erweiterungen, Tastatur, Netzteil usw. sowie Standfläche für einen Monitor. Es besteht aus zwei Halbschalen in ABS-Vollkunststoff sowie aus festen Alu-Frontplatten. Info: Bündoplast, Postfach 1460 4980 Bünde, Tel. 05223-6622.



Die FUNKSCHAU (Franzis-Verlag) veranstaltet am 24.9.80 ein ganztätiges **Seminar in München "Anwendung von Personal-Computern"** In diesem Rahmen wird der Herausgeber über den Einsatz von Personal-Computern in automatisierten Meßabläufen referieren. Weitere Themen des Seminars: Marktsituation bei Tischcomputern, Auswahlkriterien, Anforderungen an Software für Small-Business-Aufgaben. Info: Tel. 089-5117-342, Herr Feichtinger.

* * *

Eine interessante **Palette von Sichtgeräten und Computern** bietet die KEB Computer GmbH & Co in Berlin seit einiger Zeit an: Bei den Sichtgeräten handelt es sich um ergonomisch gestaltete Monitore mit schwenkbarem grünen Bildschirm und beweglicher Sichtblende. Zusammen mit der frei beweglichen Video-Tastatur VT80 mit Bildaufbereitung 80x24 bilden sie das on-line Datensichtgerät DS 1012. Die Geräte befinden sich in stabilen tiefgezogenen Alu-Gehäusen. Das trifft ebenso auf die Tischcomputer TC 2001 mit einem Cassetenlaufwerk und Tastatur (siehe) Bild) und den TC 8002 mit 2 Cassettenlaufwerken, Video-Tastatur und Monitor zu. Es handelt sich damit um gebrauchsfertige Computer auf der Basis des AIM 65 mit Netzteil und Speichererweiterungsmöglichkeiten für den professionellen Einsatz und für die Verwendung im Ausbildungswesen. Info: KEB Computer, Herr Ritzerfeld, Tel.: 030-432 60 96.



* * *

Buchbesprechung

Lance A Leventhal: 6502 Assembly Language Programming, Osborne/Mc Graw-Hill Inc. 1979. Bei Durchsicht dieses Buches merkt man sofort, daß hier ein Fachmann mit gleichermaßen hohen didaktischen Fähigkeiten am Werk war. Nach einigen Einleitungen über Programmiersprachen im allgemeinen und Assembler im besonderen, werden Adressierungsformen und Befehle der 6502 vorgestellt. Neben einer verbalen beschreibung wird eine in sich konsistente graphische Form benutzt, die zum einen den Befehlsablauf verdeutlicht, zum anderen die Wirkung auf Register und Speicher aufzeigt.

Nach Beschreibung der Befehle folgen erste Beispiele an linearen Programmen, jeweils mit Problembeschreibung, Flußdiagramm, Quellencode und Objektcode. Den Programmbeispielen sind weitere Probleme nachgestellt, die den Leser zu weiteren Lösungen anregen sollen. Weitere Kapitel behandeln Programmschleifen, Daten in Zeichenform, Code-Wandlung, arithmetische Probleme, Tabellen und Listen sowie Unterprogramme.

Während diese Abschnitte jeweils einen Umfang von ca. 25 Seiten haben, ist die Beschreibung von Ein- und Ausgabe mit über 125 Seiten sehr ausführlich und detailiert. Beschrieben werden einmal die Interface-Bausteine, die Schnittstelle zur Umwelt und die Steuerung dieses Gebildes durch die Software. Mehrere praxisbezogene Bispiele runden dieses Kapitel ab und leiten zur folgenden Darstellung der Interrupts über. An diese prozessorbezogenen Darstellungen schließen sich Abschnitte über Problemdefinition und Programmaufbau, Fehlersuche und Test sowie Dokumentation und Programmwartung an. Das Buch klingt aus mit zwei Projektbeispielen.

Insgesamt macht diese Veröffentlichung einen sehr guten, nahezu wissenschaftlichen Eindruck (viel Literaturhinweise) und sollte jedem ernsthaften Betreiber eins 6502-Systems, der sich mit der Assembler-Programmierung beschäftigen will, empfohlen werden.

Michael Zimmermann

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

Sonderangebot (als Einschub) DM 348,-

BITTE VERLANGEN SIE MEINEN SONDERDRUCK!

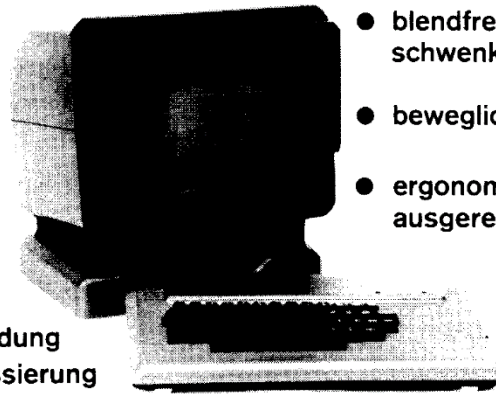
WEGNER-IMPULSTECHNIK
Bahnhofstr. 2 (alte Volksbank)
6390 Usingen
Tel. 06081-2394 oder 14435

ITT- und APPLE-Vertragshändler - Großes Computerstudio

KEB
COMPUTER

Video-Tastaturen • Monitore
Tischcomputer • Datensichtgeräte

DS 1012



- blendfreier und schwenkbarer Bildschirm.
- bewegliche Sichtblende
- ergonomisch und technisch ausgereifte Konzeption.

- Semi-Grafik
- inverse Abbildung
- Cursor-Adressierung

„Geräte der Datentechnik aus Berlin"

KEB
COMPUTER

KEB COMPUTER GMBH & CO
KONSTRUKTIONS-ELEMENTE-
BAU KG

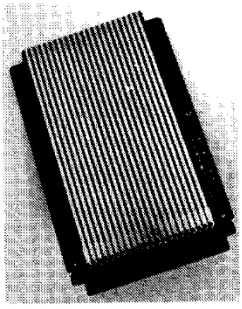
Triftstraße 25–35
D-1000 Berlin 27

Telefon (030) 432 60 96
Telex 181489 efe d

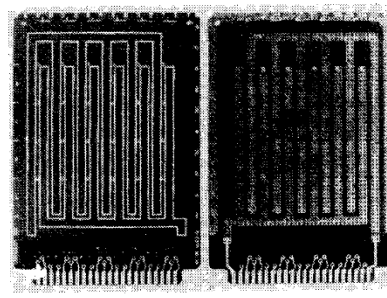
KA/UP

Verkauf von Leiterplatten, Steckadapter, Relaisplatinen

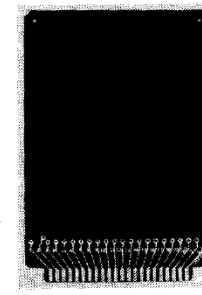
1



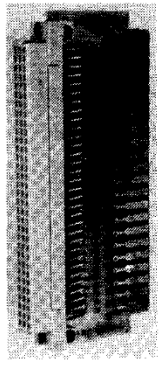
2



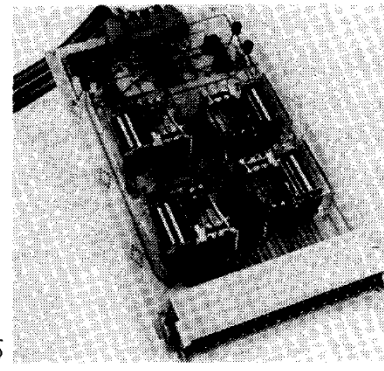
3



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 991 Relaisplatine	DM 210,60	DM 179,05	DM 157,95
6. E 941 dito m. Halter	DM 261,80	DM 221,85	DM 201,35

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

STECKER
INGENIEURBURO

5000 KÖLN 60 (Niehl)
Postfach 60 07 66
Delmenhorster Straße 29
Telefon (0221) 74 51 12

GWK

FÜR TECHNISC… …LLSCHAFT
…KTRONIK mbH.

NEU! NEU! NEU! NEU!

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

Floppy-Disk für AIM 65/PC 100

Wir haben es !!!

Das Floppy Disk System, das wirklich mit allen Eingabe- und Ausgaberoutinen voll zusammenarbeitet. Dies gilt für

ASSEMBLER - BASIC - EDITOR - MONITOR

Preise:

Doppel - Floppy - Station komplett mit Controller und Software	3.485,-- DM + MWSt
Controller Board mit res. Software	1.248,-- DM + MWSt

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, Schrittweite.

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 von Zahl in String und umgekehrt.

Erhältlich auf: Kassette, Diskette, 2 EPROM's á 2K, EPROM 4K.
Preis: DM 245,-- zuzügl. MWSt und Datenträger.

Besitzer der BASIC-Erweiterung (2K) erhalten 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

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

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 45,- (Inlandsendpreis). Ausland/Foreign via surface mail DM 50,-, USA air $30. Die früher erschienenen Ausgaben dieser Zeitschrift sind nachlieferbar. Einzelpreis für Nos. 1 - 3 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.

Leser-Service des Herausgebers

Thermopapier
für AIM 65/PC 100 in kontrastreicher Spitzenqualität
Packung mit 8 Großrollen à 65 m = 520 m, preiswert — DM 50,85

Printerplatte für AIM 65/PC 100 m. Einbauanleitung — DM 21,--

Disketten 5,25", softsektoriert, einseitig, single density (z.B. für CBM 3040), vom Hersteller 100%-ig geprüft. 10 St. in Plastik-Archiv-Aufstellbox — DM 108,--

Anwenderhandbuch für AIM 65, deutsch
Original Rockwell-Handbuch — DM 32,10

Das Buch 1-6 des 65xx MICRO MAG
ca. 230 Seiten, Sammelband der Hefte 1-6 dieser Zeitschr. gebunden, o. Anzeigen. Wertvolles Arbeitsbuch — DM 26,--

Microprocessor Systems Engineering
von Camp, Smay, Triska, englisch. Lehrbuch für 65xx/AIM 65 — DM 66,--

Vorstehende Preise sind Endpreise, einschl. Versand. NN. + DM 1,50

Programmierworkshops 65xx

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