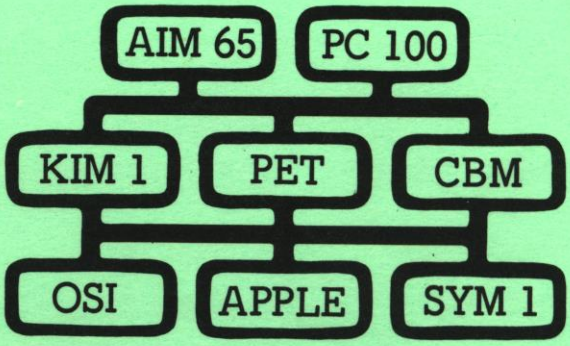
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 9,50 Nr. 26 August 1982



Inhaltsverzeichnis

FORTH!

FORTH-Befehle für 32-Bit-Zahlen

Dieser Beitrag des Herausgebers möchte einen weiteren Befehlssatz vorstellen, der das Speichern, Laden und Rechnen mit Zahlen doppelter Genauigkeit erlaubt (Darstellung in 32 Bit bzw. 4 Byte).

Es hat zwar gelegentlich Meinungsäußerungen gegeben, daß für die meisten Anwendungen von FORTH der übliche Zahlenbereich von Integerzahlen mit ca. ±32000 ausreiche. Für Anwendungen in der Steuerung von Interfaces trifft das wohl zu. In kaufmännischen und auch wissenschaftlichen Programmen kommt es aber mindestens auf Zahlen doppelter Genauigkeit an, die vorzeichenbehaftete Beträge bis ca. 2,14 Milliarden oder vorzeichenfreie Beträge bis 4,29 Mrd. zulassen. In vielen Fällen möchte man sogar einen noch größeren Zahlenbereich haben, vielleicht sogar ein Rechenpaket mit Gleitkommazahlen.

Einige Implementierungen des FORTH lassen von Haus aus nur den erstgenannten Zahlenbereich von ±32000 zu, andere haben von Anfang an den Bereich von ±2,14 Mrd., so das FORTH des AIM 65. Der Befehlsvorrat für doppelt genaue Zahlen ist hier aber gering. Er deckt die Addition mit D+, die Ausgabe einer Zahl mit D., die Vorzeichenumkehr mit DNEGATE und die Absolutwertbildung mit DABS ab. Wie die nachfolgenden Befehlslisten aber zeigen, läßt sich mit wenig Mühe ein doch recht umfassender Befehlssatz kompilieren. Wir haben dann je nach Bedarf doppelt genaue Konstante (DCONST), Variable (DVAR) und Arrays (DARRAY) sowie Befehle für das Abspeichern vom Datenstack und das Holen dorthin. Wichtig sind immer die dort auch enthaltenen Vergleichsbefehle und die Möglichkeit die kleinere oder größere von zwei Zahlen zu ermitteln (DMIN bzw. DMAX).

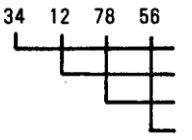
Der Aufbau einer Matrix für doppelt genaue Zahlen fällt nicht schwer. Dafür wurde das 'defining word' DARRAY geschrieben. Es wird aufgerufen: n DARRAY Name. Mit diesem Aufruf wird ein Vektor mit n+1 Elementen angelegt, es gibt damit auch ein Element mit der Ordungszahl 0. Nach dem Prinzip des Wortes DARRAY lassen sich bei Bedarf mühelos auch mehrdimensionale Anordnungen schaffen.

Aus Gründen der Didaktik wurden auch die dem DARRAY voraufgegangenen Versuche gelistet, nämlich DARRAYS, DAR@ und DAR!. Diese Befehlsworte funktionieren zwar auch, der Aufruf der beiden letzten ist aber nicht ganz FORTH-gemäß. Und dann störte den Verfasser, daß für das Holen oder Abspeichern besondere Befehlsworte gebraucht werden sollten, wenn eine Matrix angesprochen wird. Aus diesem Grunde wurde im endgültigen Wort DARRAY in jenem Definitionsteil, der das Verhalten während des Programmlaufes bestimmt (Befehle nach DOES>) die automatische Berechnung der effektiven Adresse für das anzusprechende Element 'n' aufgenommen. Damit sind auch für Matrizen die üblichen ladenden und speichernden Befehle D@ und D! weiterverwendbar.

Es gibt in DARRAY eine weitere Besonderheit: Zur Ausführungszeit wird geprüft, ob der Parameter 'n', der das n-te Element ansprechen soll, überhaupt zulässig ist. Wenn das Array z.B. für 5 Elemente angelegt ist, dann würde ein Schreiben in das 6. Element eine Nachbarzelle zerstören, was zu verhindern ist. Aus diesem Grunde wird in jedem Array zuvorderst in 2 Byte hinterlegt, für wieviele Elemente es erkärt worden ist. Man sollte auch in anderen Fällen an eine solche Sicherheitsautomatik denken.

Und es gibt eine weitere Besonderheit für alle hier vorgeführten neuen Befehlsworte: Eine doppelt genaue Zahl wird bekanntlich in der Weise in die Maschine eingegeben, daß in der Ziffernfolge an irgendeiner Stelle ein Punkt enthalten ist, z.B also: 12345.67 . FORTH unterhält nun im Systemspeicher eine Variable mit Namen DPL (Decimal Point Location), in der die Zahl der Nachkommastellen vorübergehend erfaßt wird. In vielen Fällen wäre es schade, wenn diese Information im weiteren Verlauf verloren ginge. Alle abspeichernden Befehlsworte übertragen daher auch den Inhalt von DPL in die Variablen/Arrays und alle ladenden bringen die Stellung des Dezimalpunktes nach DPL. Es ist dem Anwender überlassen, aus dieser Information ggfs. die stellengerechte Formatierung solcher Zahlen für den Bedarfsfall zu entwickeln.

Für das Verständnis der Programmierung ist kurz auf das Speicherformat doppelt genauer Zahlen einzugehen. Wenn wir hexadezimal z.B. eingeben 12345678. so finden wir auf dem Datenstack folgende Reihenfolge der Hexbytes:




```
( FORTH WORDS FOR DOUBLE LENGTH NUMBERS)
( COPYRIGHT JUNE 25, 1982 BY ROLAND LOEHR)
( REC. FOR40)


FORGET TASK : TASK ;
HEX

: DVAR
( DEFINING WORD, CREATES VARIABLE FOR DOUBLE PRECISION)
( NUMBERS, CONTAINS LOCATION OF DECIMAL POINT AND NUMERICS)
( WHEN CALLED PUTS ADDRESS TOS)
( USED: DVAR NAMEX)
<BUILDS 5 ALLOT DOES>
;

: D!
( D ADR -- ) ( STORES DOUBLE PRECISION NUMBER TO VARIABLE)
( SYNONYM TO 2!)
DUP ( ADDRESS)
DPL C@ SWAP C! ( DEC. POINTER TO VAR)
1+ DUP >R ( POINT TO NEXT LOCATION & SAVE)
! R> 2+ !    ( STORE LOW & HIGH PARTS OF NUMBER TO VAR)
;

: D@    ( D-FETCH)
( ADDR -- D) ( DPL FILLED)
( FETCHES CONTENTS OUT OF DVARIABLE)
( SYNONYM TO 2@)
DUP DUP >R >R    ( SAVE COPIES OF ADDR)
3 + @    ( GET LOW PORTION)
R> 1+ @    ( GET HIGH PORTION)
R> C@ DPL C!    ( GET LOC. OF DECIMAL POINT)
;

: DCONST
( D -- )
( DEFINING WORD FOR DOUBLE PRECISION CONSTANT)
( SYNONYM TO 2CONSTANT)
<BUILDS HERE D! 5 ALLOT    ( STORE D FROM STACK)
DOES> D@    ( PUT D ONTO STACK AT RUNTIME)
;
```

```
: 2SWAP
( D1 D2 -- D2 D1)
( SWAPS 2 DOUBLE PRECISION NUMBERS)
ROT >R    ( HIGH PART OF D1 REMOVED TO HW-STACK)
ROT R>    ( LO-PART TOS, HIGH PART BACK TOS)
;

: 2OVER
( D1 D2 -- D1 D2 D1)
( MAKES A COPY OF THE SECOND PAIR OF NUMBERS TO TOS)
4 PICK 4 PICK
;

: DO=
( D -- FLAG)
( SEE IF 2 TOS ITEMS ARE ZERO)
OR NOT
;

: D-
( D1 D2 -- D-DIFF)
( SUBRACT 2 DOUBLE PRECISION INTEGERS)
DNEGATE D+
;

: D=
( D1 D2 -- FLAG)
( COMPARE 2 DOUBLE PRECISION NUMBERS)
D- DO=    ( SUBTRACT AND SEE IF ZERO)
;

: DO<
( D1 -- FLAG)
( EXAMINES SIGN OF D1. FLAG=1 ON MINUS, ELSE =0)
SWAP DROP    ( REMOVE LOWER PART)
0<    ( EXAMINE HIGHER PART)
;

: 2DDUP
( D1 D2 -- D1 D2 D1 D2)
( DUPLICATE TOPMOST TWO 32-BIT NUMBERS)
2OVER 2OVER
;

: DU<
( UD1 UD2 -- FLAG)
( COMPARE TWO 32-BIT UNSIGNED NUMBERS)
( FLAG=1 ON UD1 < UD2)
ROT 2DUP    ( THE TWO HIGHER PARTS TOS & DUPLICATED)
= IF 2DROP U<    ( ARE EQUAL, ONLY CHECK LOWER PARTS)
  ELSE U<    ( COMPARE HIGHER PARTS)
        IF 2DROP 1    ( FLAG=1, UD1<UD2)
        ELSE 2DROP 0    ( UD1 MUST BE GREATER)
        THEN
  THEN
;
```

```
: D<
( D1 D2 -- FLAG)
( COMPARE TWO SIGNED 31-BIT NUMBERS. FLAG=1 ON D1 < D2)
( COMMENTS AS IN DU<)
ROT
2DUP
= IF 2DROP U<
  ELSE
   D- DROP 0<
  THEN
;

: DU>
( UD1 UD2 -- FLAG)
( COMPARES TWO 32-BIT UNSIGNED NUMBERS)
( FLAG=1 ON UD1 > UD2, ELSE =0)
2DDUP D= IF 2DROP 2DROP 0
         ELSE DU< NOT
         THEN
;

: D>
( D1 D2 -- FLAG)
( COMPARES TWO 31-BIT SIGNED NUMBERS)
( FLAG=1 ON D1 > D2, ELSE =0)
2DDUP D= IF 2DROP 2DROP 0
         ELSE D< NOT
         THEN
;

: DMIN
( D1 D2 -- DMIN)
( COMPARES TWO 31 BIT-SIGNED NUMBERS & LEAVES THE SMALLER )
2DDUP ( DUPLICATE BOTH)
D= IF 2DROP
   ELSE 2DDUP D<
                IF 2DROP
                ELSE 2SWAP 2 DROP
                THEN
   THEN
;

: DMAX
( D1 D2 -- DMAX)
( COMPARES TWO 31-BIT SIGNED NUMBERS & LEAVES DMAX TOS)
2DDUP    ( DUPLICATE D1 & D2)
D= IF 2DROP    ( EQUAL, ONLY D1 LEFT)
   ELSE 2DDUP    ( DUPLICATED ONCE MORE)
        D< NOT IF
                 2DROP
               ELSE 2SWAP 2DROP
               THEN
   THEN
;
```

```
: DARRAYS
( DEFINING WORD TO CREATE A LINEAR ARRAY OF N+1 ITEMS)
( OF 32-BIT NUMBERS AND THEIR DECIMAL POINT LOCATIONS)
( CALLED: N DARRAYS NAMEX, WHERE N=ITEMS-1)
( THIS WORD IS AN EXAMPLE ONLY. INSTEAD USE DARRAY)
<BUILDS 1+ 5 * ALLOT DOES>    ( N+1 * 5 BYTES)
;

: DAR!
( D ADDR ITEM-# -- )
( STORE 32-BIT NUMBER D AS ARRAY ITEM-#)
( WORD NOT RECOMMENDED, EXAMPLE ONLY FOR DARRAYS)
5 * + D!    ( ITEM-# TIMES 5 ADDED TO ADDRESS)
;

: DAR@
( ADDR ITEM-# -- D)
( FETCHES D FROM ITEM-# IN DARRAY)
( WORD NOT RECOMMENDED. EXAMPLE ONLY FOR DARRAYS)
5 * + D@    ( ADDRESS PLUS 5 TIMES ITEM-# = LOCATION)
;

: ARRAYERR
( DISPLAYS MESSAGE)
." ARRAY ERROR" ABORT
;


: DARRAY
( DEFINING WORD TO CREATE LINEAR ARRAY OF N+1 ITEMS)
( OF 32-BIT DOUBLE PRECISION NUMBERS)
( CALLED: N  DARRAY NAMEX)
( AT RUNTIME EXECUTES SECURITY AGAINST ARRAY ERROR)
<BUILDS DUP ,    ( # OF ITEMS STORED AT PFA ADDRESS)
         1+ 5 * ALLOT    ( RESERVE 5 BYTES PER ITEM)
        DOES>    ( AT RUNTIME)
   2DUP @ 1+ SWAP    ( COMPARE MAX. ITEMS TO ADDRESSED ITEM)
          > IF 2+ SWAP 5 * +    ( LOCATION OF ITEM IN ARRAY)
            ELSE ARRAYERR
            THEN
;
```

```
FINIS          123.4567 DCONST ZZ OK
               ZZ D. 1234567 OK
               DVAR XYZ OK
               9876.54 XYZ D! OK
               XYZ D@ D. 987654 OK
               DPL @ . 2 OK
               HERE . 5E7 OK
               3 DARRAY JOTA OK
               4 JOTA D@ ARRAY ERROR
                 AIM 65 FORTH V1.3
               HERE . 608 OK

               77. 0 JOTA D! OK
               0 JOTA D@ D. 77 OK
               0 JOTA D@ ZZ DMAX D. 1234567 OK
               0 JOTA D@ ZZ DMIN D. 77 OK
```

Nebenstehend einige kleine Anwendungsbeispiele für die Befehlsworte

R.L. #

FORTH (3)

Im letzten Abschnitt (3.6) lernten wir als erste Kontrollstruktur zur Beeinflussung des Programmablaufes die Zählschleife DO ... LOOP kennen. Ehe wir die weiteren Kontrollstrukturen besprechen, sollten wir die Stackbefehle und die immer leicht nachvollziehbaren arithmetischen Befehlsworte im Zusammenhang darstellen, danach die Vergleiche.

4. Stackbefehle

Die Reihenfolge und Menge der auf dem Datenstack befindlichen Operanden entspricht nicht immer den Erfordernissen der nächsten Abarbeitung. Man muß daher in der Lage sein, die Reihenfolge umzustellen, ggfs. (nach einem Vergleich) nicht mehr benötigte Operanden zu entfernen oder auch Duplikate bereitstellen zu lassen, denn die meisten Befehle verbrauchen oder verändern die auf dem Stack befindlichen Parameter. Die Stackbefehle dienen den genannten Aufgaben.

Bei den folgenden Beispielen, in denen vor allem die Wirkung der Befehle gezeigt wird, gehen wir gleichmäßig davon aus, daß sich vor der Befehlsausführung folgende markanten Zahlen auf dem Datenstack befunden haben:

```
DECIMAL  11  22  33  44
```

Stackbefehle für 16-Bit-Zahlen

Befehl	Beschreibung		Wirkung
DUP	Dupliziere oberstes Element (item)	(n -- n n)	11 22 33 44 44
DROP	Entferne oberstes item	(n --)	11 22 33
SWAP	Vertausche die beiden obersten items	(n1 n2 -- n2 n1)	11 22 44 33
OVER	Dupliziere das zweitoberste item	(n1 n2 -- n1 n2 n1)	11 22 33 44 33
n PICK	Dupliziere das n-te Element	(n -- n n-tes) z.B. 4 PICK	11 22 33 44 11
ROT	Ringtausch der drei obersten items, das dritte gelangt nach ganz oben	(n1 n2 n3 -- n2 n3 n1)	11 33 44 22
-DUP	Dupliziere oberstes item nur dann, wenn es ungleich Null ist	(n --- n ?)	11 22 33 44 44

Befehle im Zusammenhang mit dem Hardware-Stack

>R	Entnimm oberstes item vom Datenstack und speichere es auf den Hardware-Stack um	(n ---)	11 22 33 44
R>	Speichere oberstes Element vom Hardware-Stack auf den Datenstack zurück	(-- n)	11 22 33 44
R oder I	Dupliziere das oberste item des Hardware-Stacks auf den Datenstack	(-- n)	11 22 33 44 xx

Der Leser möge die vorgenannten einfachen Beispiele an der Tastatur nachvollziehen und dabei Aufzeichnungen darüber führen, was sich vorher und nachher auf dem Datenstack befunden hat. Es ist zu erinnern, daß der Befehl .S (s-dot) dem nicht zerstörenden Ausdruck des Datenstacks dient. Er ist hier - wie überall - zur Kontrolle von Modulen wertvoll. Der einfache Druckbefehl '.' (dot) druckt zwar die oberste Zahl aus, entfernt sie damit aber vom

Stack. Es ist dem Leser unbenommen, mehrere dot-Befehle (getrennt durch Zwischenräume) in eine Zeile zu schreiben, z.B. . . . . zur Ausgabe von 4 Zahlen.

Die Befehle >R und R> sind häufig beim Umsortieren von Operanden nützlich, auch im Zusammenhang mit dem Duplizieren vom Hardware-Stack (Befehl 'R'). Sie sollten in einer Befehlszeile oder in einem neuen Befehlswort paarweise gebraucht werden, weil die Maschine wegen der Veränderung des Hardware-Stacks sonst abstürzt.

Hinsichtlich der Bearbeitung des Datenstacks sei auch auf die Möglichkeit hingewiesen, Parameter vorübergehend in Variablen abzulegen, aus denen sie beliebig oft dupliziert werden können oder Parameter aus Konstanten zu beziehen. Auf Variable und Konstante gehen wir im nächsten Abschnitt ein.

Wie bereits erwähnt wurde, haben viele FORTH-Implementierungen auch doppelt genaue Zahlen (32 Bit, 4 Byte). Für diese gelten folgende Stackbefehle:

2DUP	Dupliziere die beiden obersten items	(n1 n2 -- n1 n2 n1 n2)	11 22 33 44 33 44
2DROP	Entferne die beiden obersten items	(n1 n2 --)	11 22

Da diese Befehle immer auf 4 Bytes wirken, ohne Ansehung des vorher auf den Stack geschickten Datentyps, lassen sie sich selbstverständlich auch auf Paare von 16-Bit-Zahlen anwenden. Wo diese und ähnliche Befehle nicht vorhanden sind, lassen sie sich bei Bedarf leicht als neue Befehlsworte nachimplementieren:

```
: 2DUP  DUP  DUP ;          ( n1 n2 -- n1 n2 n1 n2)
: 3DUP  DUP  DUP  DUP ;
: 2OVER  4 PICK 4 PICK ;    ( d1 d2 -- d1 d2 d1)
```

usw., wobei n bzw. n1 etc. Zahlen einfacher Genauigkeit symbolisieren und d bzw. d1 etc. Zahlen mit doppelter Genauigkeit.

5. Arithmetische Befehle

5.1. Konstante und Variable einfacher Genauigkeit

Für die Benutzung von Zahlen und beliebigen Operanden sowie für deren Abspeicherung und Weiterverwendung sind Konstante und Variable in allen Computersprachen unentbehrlich. FORTH fordert vom Benutzer, daß er sie dem System vor deren Benutzung erkärt. Dem dienen die sog. 'defining words' CONSTANT und VARIABLE. Sie schaffen einen namentlichen Eintrag in das Verzeichnis der Befehlsworte (dictionary) und versehen diesen Eintrag mit einem Initialwert. Im weiteren Verlauf werden Konstante und Variable mit ihrem Namen aufgerufen. Konstante legen dabei ihren Inhalt auf dem Datenstack ab, Variable ihre Adresse. Dieser Unterschied ist nötig, damit man in Variable auch hineinschreiben kann. Die reservierten Worte werden wie folgt benutzt:

```
-123 CONSTANT Name1        Anlage der Konstanten Name1, Wert -123
25   VARIABLE Name2        Anlage der Variablen Name2 mit Wert 25
```

Für das Holen des Wertes aus einer Variablen benutzt man das FORTH-Wort '@' (siehe auch Abschnitt 8), für das Schreiben vom Datenstack in eine Variable benutzt man den Befehl ' ! '. Englisch werden sie als 'fetch' und 'store' ausgesprochen. Also:

```
Name1 @          bringt -123 auf den Stack
Name2 @          bringt den Inhalt 25 aus der Variablen
44 Name2 !       speichert 44 in Variable Name2
```

5.2 Arithmetische Befehle für Zahlen einfacher Genauigkeit

Im Abschnitt 3.1. lernten wir bereits die Grundrechenarten für 32-Bit-Zahlen kennen:

- \+ Addition
- – Subtraktion
- * Multiplikation (ganzzahlig)
- / Division (ohne Rest oder Dezimalbruch)

Hinzu kommen die schnell erprobten Befehlsworte

MOD Division mit Ablage nur des Divisionsrestes
/MOD Division mit Ablage von Rest und Quotient (zuoberst)

Es gibt weitere multiplizierende und dividierende Befehle aus deren Symbolik wir ablesen: Ein '*' bedeutet immer Multiplikation, '/' Division, 'MOD' (modulo), daß auch ein Divisionsrest erzeugt wird. 'U' in diesen Kombinationen bedeutet, daß die Zahl als vorzeichenfrei behandelt wird (das höchstwertige Bit gilt als Binärziffer, nicht als Vorzeichen). Mit n, n1 etc. bezeichnen wir vorzeichenbehaftete Zahlen (16 Bit), mit d, d1 solche von 32 Bit inkl. Vorzeichen, mit u1, u2 vorzeichenlose (unsigned) 16-Bit Zahlen und mit ud1 etc. ebensolche mit 32 Bit.

Der Druckbefehl für 16-Bit-Zahlen ('.' dot) interpretiert Zahlen größer als 32767 als negativ. Folgende Befehlszeile ergibt daher das Ergebnis –25536 statt der erwarteten 40000:

```
200 200 * .                    -25536OK
```

Bitmäßig wird bei den genannten Grundrechenarten richtig gerechnet, nur der Druckbefehl ist nicht für alle Fälle geeignet. Wir sollten daher das ggfs. störende negative Ergebnis durch Nachschieben einer Null zu einer doppelt genauten Zahl machen und den Ausgabebefehl für doppelt genaue Zahlen (D. d-dot) anwenden:

```
D.                             ( d -- )      d-dot, Ausgabefehl für 32 Bit
200 200 * 0 D.                 40000OK
```

Wenn wir von den noch zu besprechenden Spielarten der Multiplikation und Division absehen, finden wir in FORTH keine höheren mathematischen Funktionen. Solche können ggfs. z.B. mit dem Math-ROM von Rockwell nachimplementiert werden.

Für 16-Bit-Zahlen oder -Elemente stehen weitere Befehle zur Verfügung, die auf jeweils 2 Operanden wirken ('binäre' Befehle):

Binäre Befehle

			Auf unser Beispiel
MAX	hinterläßt das Maximum zweier Zahlen	(n1 n2 -- max)	11 22 44
MIN	hinterläßt das Minimum zweier Zahlen	(n1 n2 -- min)	11 22 33
	z.B. MIN MIN MIN		11
AND	logisches AND	(n1 n2 -- n')	11 22 32
OR	logisches ODER	(n1 n2 -- n')	11 22 45
XOR	logisches exklusiv ODER	(n1 n2 -- n')	11 22 13
+–	XOR der Vorzeichenbits	(n1 n2 -- n1')	11 22 33

Bei den nachfolgenden Befehlsworten sollten wir uns deutlich vor Augen halten, daß sie 'unär' sind, sie wirken nur auf einen Operanden.

Unäre Befehle

ABS	Bildung des Absolutwertes	(n -- +n)	
	z.B. –123 ABS		123
NEGATE	Vorzeichenumkehr	(n -- –n)	11 22 33 –44
1+	Erhöhung/Erniedrigung		11 22 33 45
2+	des Operanden top of stack		11 22 33 46
1–	um 1 oder 2		11 22 33 43
2–			11 22 33 42

NOT Invertiert ein Flag=1 zu 0 oder 0 zu 1

Die vier Befehle ab 1+ werden häufig gebraucht und erhielten daher kompakte Befehlsworte. Sie sind nach folgendem Muster definiert:

```
: 1+  1  +  ;                    usw.
```

5.3 Artithmetische Befehle für gemischte und doppelte Genauigkeit

Auf 16 Bit wirkende Befehle führen schneller aus als vergleichbare für doppelte Genauigkeit. Es wird daher empfohlen (ggfs. auch aus Gründen des Speicherbedarfes), soweit als möglich mit einfacher Genauigkeit zu rechnen. Im Bedarfsfall kann man eine Zahl auf 32 Bit erweitern oder vorübergehend mit doppelter Genauigkeit rechnen und das Ergebnis in einfacher Genauigkeit zurückerhalten. Das trifft auf die meisten der nachfolgenden Befehle zu. Man sollte sie nicht unbedingt auswendig lernen, sie stattdessen einmal ausprobieren und dann aufschlagbar halten.

*/ 'times divide' (n1 n2 n3 -- quot)
Multiplikation n1*n2 mit doppelt genauem
Zwischenergebnis, anschl. Teilung durch n3
mit Quotient in einfacher Genauigkeit

Dieser Befehl wird angewendet, wo das Produkt aus n1*n2 die Darstellungsmöglichkeit in 16 Bit sprengen würde, z.B. bei der Berechnung von 4/5 von 30000

```
30000  4  5  */          24000 OK
```

Typische Anwendung das 'scaling', Meßwerte müssen mit einem konstanten Faktor ausgewertet werden. - Analog liefert der nachfolgende Befehl auch den Divisionsrest ab

*/M times divide-mod (n1 n2 n3 -- rest quot)

Weitere Multiplikationsbefehle

U* unsigned multiply (u1 u2 -- ud)
M* Multiplikation wie vor, jedoch mit Berücksichtigung der Vorzeichen (n1 n2 -- d)
Faktoren einfacher Genauigkeit, Ergebnis in doppelter

Der Division dienen

U/ Vorzeichenfreie Division eines 32-Bit Dividenden durch 16-Bit Divisor, Rest und Quotient ergeben sich in einfacher Genauigkeit (ud u1 -- rest quot)
M/ wie U/, allerdings mit Berücksichtigung der Vorzeichen (d n1 -- rest quot)
M/MOD Division ohne Berücksichtigung der Vorzeichen mit 32-Bit Quotient und 16-Bit Rest
(ud1 u2 -- urest ud3)

D+– EXOR der Vorzeichen von d und n (d n --- d')

S→D Expansion einer 16-Bit Zahl auf eine doppelt genaue Zahl gleichen Vorzeichens (n -- d)

Als 'binärer' Befehl, der also auf 2 Operanden doppelter Genauigkeit wirkt, ist beim AIM 65 nur 'D+' implementiert

D+ d-plus, Addition in doppelter Genauigkeit (d1 d2 -- dsum)

Unäre Befehle sind

DABS Absolutwertbildung
DNEGATE Vorzeichenumkehr

Eine Reihe weiterer Befehle, die man leicht nachimplementieren kann, ist an anderer Stelle in diesem Heft abgedruckt. Die Auswahl und Benutzung einfacher oder doppelter Genauigkeit sowie der für sie anzuwendenden Befehle wird man von der zu lösenden Aufgabe abhängig machen.

6. Vergleichsbefehle

Auch bei den Vergleichsbefehlen müssen wir die 'unären', die den Zustand nur eines Operanden prüfen, von den 'binären' trennen, die zwei Operanden vergleichen. Vergleichsbefehle liefern auf dem Datenstack ein Flag ab, das als 'true flag' 1 ist und als 'false flag' eine 0 (wenn die Bedingung nicht zutrifft).

Unäre Prüfungen

0< Vorzeichenprüfung (n -- flag)
flag=1, wenn Operand negativ

0= Prüfung auf Null (n -- flag)
flag=1, wenn Operand=0

NOT Umkehr des flag 1=0 oder 0=1 (f -- f')

Forth hat also keinen vollständigen Satz von Vergleichsoperatoren, z.B. für 'ungleich 0' oder 'größer 0'. Die Abfrage auf 'ungleich 0' erfolgt daher mit

```
0= NOT IF ...
```

Für die Bestimmung 'größer 0' braucht man 2 Abfragen, die man ggfs. als ein Befehlswort definiert:

```
: >0  DUP  (DUPLIZIERE ØPERANDEN)
  0<  IF   DROP Ø  (FLAG=Ø)
      ELSE  =  IF  Ø  (FLAG=Ø)
               ELSE 1  (TRUE, NICHT KLEINER ODER GLEICH)
               THEN
      THEN ;
```

Entsprechend kann man sich Operatoren für 'kleiner oder gleich 0' und 'größer oder gleich 0' zusammenbauen:

```
: Ø<= DUP  Ø< IF DROP 1
              ELSE Ø= IF 1
                      ELSE Ø
                      THEN
              THEN ;
: >=Ø Ø<  NOT ;
```

Wir haben hier bereits weitere Kontrollstrukturen benutzt. Vergleiche aller Art dienen ja dazu, den Programmablauf im Rahmen solcher Strukturen zu beeinflussen. Mehr dazu in Abschnitt 7.

Binäre Vergleiche

Für den Vergleich zweier Operanden (16 Bit) haben wir 4 Operatoren, aus denen sich ggfs. ebenfalls wieder komplexere zusammenstellen lassen:

<	Vergleich auf kleiner flag=1, wenn n1<n2	(n1 n2 -- flag)
>	Vergleich auf größer flag=1, wenn n1 > n2	(n1 n2 -- flag)
=	Prüfung auf Gleichheit flag=1, wenn n1=n2	(n1 n2 -- flag)
U<	Vergleich auf kleiner für vorzeichenfreie Zahlen flag=1, wenn u1 < u2	(u1 u2 -- flag)

Vergleiche für doppelt genaue Zahlen sind an anderer Stelle in diesem Heft abgedruckt.

7. Kontrollstrukuren des FORTH

Wir haben jetzt das Werkzeug in der Hand, um ein Programm in Zähl- und Bedingungsschleifen zurückzuverzweigen oder um es in Abhängigkeit von Ergebnissen unterschiedliche Erledigungspfade einschlagen zu lassen.

Allen Kontrollstrukturen ist gemeinsam, daß sie nur innerhalb von Befehlswortdefinitionen zwischen Doppelpunkt und Semikolon (: Name ... Kontrollstruktur ... ;) verwendet werden dürfen (bzw. in Programmen für den FORTH-Assembler analog zwischen den Worten CODE und END-CODE). Siehe auch Abschnitt 3.3.

Es gibt folgende 3 Grundtypen der Kontrollstrukturen

```
DO ... LOOP
IF ... THEN
BEGIN ... UNTIL
```

Folgende Strukuren sind ähnlich

DO ... +LOOP	entspricht dem STEP in BASIC
IF ... ELSE ... THEN	2-seitige Programmverzweigung
BEGIN ... AGAIN	endlose Programmschleife
BEGIN ... WHILE ... REPEAT	Schleife mit Eingangsprüfung

Im einzelnen setzt die DO...LOOP voraus, daß sich vor ihrem Aufruf der Endparameter+1 und der Anfangszählparameter auf dem Datenstack befinden, siehe Abschnitt 3.6 in Heft 25. Als Parameter sind hier nur Integerzahlen (einfache Genauigkeit) zulässig.

Hinsichtlich des Kontrollwortes BEGIN muß man sich vorstellen, daß sich das FORTH-System seine Stelle im Programm als Rücksprungadresse (während der Kompilierung) merkt. BEGIN schafft also eine Art Marke oder 'label'. Das Gleiche ist zu THEN zu sagen, es stellt die Marke dar, hinter der es sowohl nach dem IF-Zweig wie nach ELSE gemeinsam weitergeht. Ebenso ist REPEAT der Fortsetzungspunkt, wenn die bei WHILE geprüfte Bedingung nicht mehr zutrifft.

So bleiben besonders noch die Worte IF, UNTIL und WHILE zu besprechen: Jedes von ihnen setzt ein Flag auf dem Datenstack voraus und entfernt es von dort im Zuge der Prüfung. Im einzelnen werden in Abhängigkeit vom Flag durchlaufen:

	Flag =	Es wird durchlaufen	Fortsetzung
IF	1	IF-Zweig	nach THEN
	0	ELSE-Zweig	nach THEN
BEGIN/	1	BEGIN bis WHILE und WHILE bis REPEAT	bei BEGIN
WHILE	0	BEGIN bis WHILE	nach REPEAT
BEGIN/	X	BEGIN bis UNTIL	
UNTIL	1		nach UNTIL
	0	BEGIN bis UNTIL	bei BEGIN
/AGAIN	x	BEGIN bis AGAIN	BEGIN

Der Unterschied zwischen BEGIN...UNTIL und BEGIN...WHILE...UNTIL besteht darin, daß im ersten Fall das gesamte Konstrukt mindestens einmal durchlaufen wird, weil die Bedingungsprüfung am Schluß steht. Im zweiten Fall erfolgt nach einem zwischen BEGIN und WHILE ggfs. durchlaufenen Vorspann möglicherweise sofort eine Verzweigung hinter REPEAT, wenn die Eingangsgedingung nicht zutrifft. In diesem Falle wird der dazwischenliegende Teil nicht durchlaufen.

Zur Kontrolle der sich ergebenden Abläufe gebe der Leser beispielsweise folgende Definitionen ein:

```
: IF-DEMO       IF ." 1. ZWEIG"
                ELSE ." 2. ZWEIG"
                THEN
                ." GEMEINSAME FORTSETZUNG" ;
: WHILE-DEMO    BEGIN  DUP 1- SWAP ." VORSPANN"
                 0= NOT
                WHILE ." INNENSCHLEIFE"
                REPEAT ." ENDE" DROP :
: UNTIL-DEMO    BEGIN  DUP 1- SWAP ." AUSFUEHRUNG"
                0=
                UNTIL ." ENDE" DROP :
```

Nach diesen Definitionen können wir versuchsweise aufrufen

```
0 IF-DEMO       oder 1 IF-DEMO
0 WHILE-DEMO    oder 3 WHILE-DEMO
0 UNTIL-DEMO    oder 2 UNTIL-DEMO
```

Mit diesen kleinen Versuchen dürfte das Ablaufprinzip schnell zu durchschauen sein. - Dem Assembler-Programmierer wird bei BEGIN...UTIL auffallen, daß er an vergleichbarer Stelle 'UNTIL' immer 'BNE' programmiert, Zurückverzweiung, solange der Parameter noch nicht Null ist. In FORTH wird vergleichbar formuliert 'rückverzweigen, bis der Parameter (das Flag) Null ist'.

Kontrollstrukturen dürfen tief ineinander verschachtelt werden, wie auch ein Blick auf schon veröffentlichte Programme zeigt. Man achte darauf, daß innenliegende Konstrukte immer zuerst abgeschlossen (aufgelöst) werden, ehe die Fortsetzung für das außenliegende erfolgt. Erlaubt sind also Abfolgen wie

```
IFa  IFi  THENi  ELSEa  THENa                          a=außenliegend
IFa  ELSEa  IFi  ELSEi  THENi  THENa                    i=innenliegend
BEGINa  IFi  THENi  WHILEa  BEGINi  UNTILi  REPEATa     usw.
```

Schließlich ist noch auf die +LOOP hinzuweisen, die die Laufvariable I der Schleife nicht um 1, sondern um eine vom Anwender bestimmbare Ganzzahl zu verändern gestattet. Beispiel die arithmetische Reihe mit STEP (Inkrement) 3:

```
: REIHE I . 3 +LOOP :

Aufruf z.B. 10 0 REIHE        ergibt 0 3 6 9
            20 10 REIHE       ergibt 10 13 16 19
```

Wie man sieht, bestimmt der oberste Parameter (auf dem Stack) den Startpunkt, der zweite die Grenze, die nicht überschritten werden darf.

Eine einfache Anwendung von Multiplikation und Schleifenprizip ist die Fakultätsberechnung (bis 8!):

```
: FAK 1+ 1 SWAP 1 DO I * LOOP 0 D' ;
1 FAK 1
2 FAK 2
8 FAK 40320
```

Innerhalb einer DO...LOOP kann man mit LEAVE einen Zwangsabbruch nach Ende des aktuellen Durchlaufes erzwingen. Dieses LEAVE mag man im Rahmen einer IF...THEN-Prüfung verwenden.

8. Auf Speicherzellen wirkende Befehle

Ein großer Reiz der Sprache FORTH speziell für die Steuerung von Interfaces oder die Veränderung benamter Systemvariablen liegt in ihrem Befehlssatz, der die direkte Ansprache von Speicherzellen ermöglicht, wobei die Operanden je nach gewählter Zahlenbasis hexadezimal, binär oder auch dezimal sein mögen. Die wesentlichen Funktionen umfassen das Holen oder Speichern von 1 oder 2 Bytes pro Befehl, die direkte Addition in Speicherzellenpaare, den Transport von Speicherblöcken und das Füllen mit wählbaren Byte-Mustern.

Befehle, die auf 1 Byte wirken

C@	1 Byte auf den Datenstack holen	(adr -- byte)
C!	1 Byte abspeichern	(byte adr --)
?	1 Byte holen und ausgeben	(adr --)
TOGGLE	Inhalt unter Adresse mit Byte logisch EXORen	(adr byte --)

Befehle, die auf 2 Byte wirken

@	1 WORD (2 Byte) auf den Datenstack holen	(adr -- n)
!	1 WORD nach Adresse abspeichern	(n adr --)
+!	Inhalt des Wortes unter Adresse um n erhöhen	(n adr --)

Befehle, die auf n Bytes wirken

CMOVE	n Bytes von nach transportieren	(vonadr nachadr n --) (aufsteigende Folge)
FILL	Fülle n Bytes ab Adresse gleichmäßig mit Byte b	(adr n b --)
ERASE	Fülle n Bytes ab Adresse mit 00	(adr n --)
BLANKS	Fülle n Bytes ab Adresse mit Blanks (hex 20)	(adr n --)

Eine Kontrolle für die Wirkung dieser Befehle oder einen immer einmal notwendigen Speicherabzug erhält man durch den Befehl DUMP, der n Bytes ab Adresse ausgibt (Zahlenbasis beachten!):

DUMP n Bytes ab Adresse ausgeben (adr n --)

9. Die Benutzung des FORTH-Assemblers

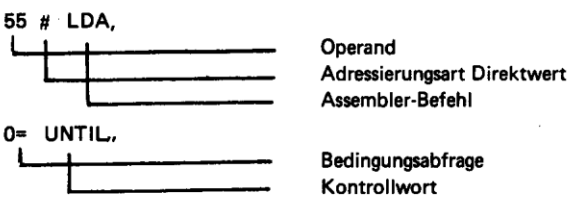
Auch wenn es nicht in einen ganz systematischen Stoffaufbau paßt, so soll bereits an dieser Stelle auf den Assembler unter FORTH eigegangen werden, weil er die Möglichkeiten der Sprache hervorragend unterstützt und weil er bisher zu wenig an anderer Stelle abgehandelt worden ist.

Dieser Assembler ist beim AIM 65 vom Assembler-ROM (ab hex D000) unabhängig, bei Commodore-Rechnern z.B. unabhängig vom Assembler auf der Diskette. Er ist vielmehr ein Teil der FORTH-Implementierung, ist wenigstens zum Teil in FORTH formuliert und arbeitet daher ebenfalls umgekehrt polnisch, woran man sich allerdings schnell gewöhnt.

Er hat weitere Eigenheiten: Er ist ein 1-Pass-Assembler, der im wesentlichen ohne Label arbeitet, dafür aber mit den in Abschnitt 7 besprochenen Kontrollstrukturen, die interne Label erzeugen. Daneben sind Symbole als Inhalt von FORTH-Konstanten heranziehbar sowie symbolische Adressen, die man sich auf anderem Weg aus dem System besorgt.

Die Kontrollstrukturen bringen es mit sich, daß die 8 landläufigen Verzweigungsbefehle der CPU 6502 nicht zum benutzbaren Wortschatz des FORTH-Assemblers gehören, gleichwohl erzeugt er an den Eckpunkten der Strukturen alle diese Befehle. Wir sehen es bei einer späteren Disassemblierung. Alle anderen 'mnemonics' und alle Adressierungsarten stehen dem Programmierer sonst jedoch zur Verfügung.

Alle mnemonischen Befehle und auch die Kontrollworte wie IF, THEN, BEGIN usw. sind nach dem letzten Buchstaben mit einem Komma abzuschließen, also z.B.



Die Notierung der Adressierungsarten weicht vom Standard-Assembler etwas ab:

FORTH	Standard-Assembler
.A ROL,	ROL A (.A zur Unterscheidung von hex A)
1 # LDX,	LDX#1
DATA ,X STA,	STA DATA,x
DATA ,Y CMP,	CMP DATA,Y
5 X) SBC,	SBC (05,X)
POINT)Y STA,	STA (POINT),Y
VECTOR) JMP,	JMP (VECTOR)

Das Vokabular des Assemblers ist bei der normalen Auslistung unter FORTH verborgen. Man muß zunächst sagen: ASSEMBLER, dann VLIST und erhält einen Kontrollabzug (wir brauchen im Moment nicht weiter auf die unter FORTH möglichen verschiedenen Vokabulare einzugehen).

Der Assembler wird mit dem Befehlswort 'CODE' hereingerufen und wird mit 'END-CODE' wieder verlassen. Mit dem Assembler erzeugte Befehlsworte sind hernach wie normale FORTH-Worte ausführbar, sie werden mit dem Aufruf ihres Namens zur Ausführung gebracht. Solche Worte haben folgendes Grundformat für die Kompilierung:

```
CODE  Name  Assemblerbefehle  NEXT JMP,  END-CODE  oder
CODE  Name  Assemblerbefehle  PUSH0A JMP,  END—CODE  oder
CODE  Name  Assemblerbefehle  PUSH JMP,  END-CODE
```

Die Begrenzer CODE und END-CODE haben also die gleiche Bedeutung wie der Doppelpunkt und das Semikolon bei der Kompilierung normaler FORTH-Worte. Die erste der hier gezeigten Versionen bringt beim Verlassen der Routine nichts auf den Datenstack, die zweite hinterläßt eine Null als high Byte und den Akku-Inhalt zuoberst auf dem Datenstack (Parameterübergabe an FORTH). Die dritte Version schließlich übergibt die Parameter aus Indexregister X und dem Akkumulator auf den Datenstack.

Auch vom Datenstack des FORTH her können wir in der Gegenrichtung Parameter an ein Assemblerprogramm übergeben, und zwar durch die Befehlsfolge

```
n #LDA,
SETUP JSR,                (Auslösung des Transports)
```

n ist dabei eine Zahl zwischen 1 und 4, die n*2 Bytes bezeichnet (also n Byte-Paare), die vom Datenstack in die Zeropage des Computers befördert werden sollen, wo sie als Pointer oder zum Zwecke des Rechnens weiterverwendet werden können. Alles ist automatisch: Der Pointer auf den Datenstack (das X-Register) wird mit der Entnahme erhöht, und die Bytes stehen anschließend in der sog. N-Area an definierter Stelle zur Verfügung. Das ist ein Zero Page-Bereich von symbolisch N—1 bis N+7. Das erste Byte-Paar landet dabei bei N, das zweite bei N+2 usw. (N= hex 97 beim AIM 65). Der Assembler kennt ferner den Einsprungspunkt von SETUP, so daß wir ihm nichts dekkarieren müssen. Er kennt auch den Wert des Symboles N, so daß wir hernach einfach programmieren können

```
N  LDA,                   (lade Akku aus Adresse 97)
N  3  +  LDA,             (lade Akku aus Adresse 9A)
N  2+  )Y  LDA,           (lade indirekt ,Y per Pointeradresse 99/9A)
```

Es gibt weitere dem Assembler bekannte Symbole. Zunächst ist da eine reservierte Speicherstelle, die der Rettung des Daten-Stackpointers dient, XSAVE

```
XSAVE  STX,               (Rettung beim Eintritt in eigene Routine)
XSAVE  LDX,               (Wiederherstellung des Pointers)
```

Für mehr hintergründige Fälle, in denen man in die Innereien des FORTH-Ablaufes eingreifen will, kann auch auf die Symbole IP, W und UP Bezug genommen werden (Interpretive Pointer, der auf das nächste Befehlswort hinweist, W der Pointer auf die Codefeldadresse des derzeit ausgeführten Wortes und UP der Pointer auf die User-Area).

Wichtiger sind zunächst die typischen Ablaufstrukturen, die auf dem physikalischen Prozessorstatus fußen: BEGIN, , THEN, , REPEAT, (immer mit Kommal) sind wie in Abschnitt 7 wieder unsichtbare Marken, die sich der Assembler für die Rückwärts- und Vorwärtssprünge bzw. Verzweigungen anlegt, während IF, , WHILE, und UNTIL, Stellen sind, an denen die Bedingungsabfrage erfolgt.

Unter Assembler des FORTH stehen uns folgende Verzweigungsbefehle zur Verfügung:

Statusabfrage	antivalente Statusabfrage	Status-Bit
CS carry set	CS NOT	C=1 bzw. C=0

0< negative, less than zero	0< NOT	N=1 bzw. N=0
0= equal to zero	0= NOT	Z=1 bzw. Z=0
VS overflow set	VS NOT	V=1 bzw. V=0

Die Formulierung der Verzweigungsbedingung an den Stellen IF, , WHILE, und UNTIL, fällt dem geübten Assemblerprogrammierer bei den ersten Malen noch schwer, wie die folgende Gegenüberstellung zeigt:

Normaler Assembler	FORTH-Assembler
LOOP	BEGIN,
DEY	DEY,
BNE LOOP	0= UNTIL,
oder	
LOOP	BEGIN,
DEX	DEX,
CPX # 5	5 # CPX,
BNE LOOP	0= UNTIL,
oder	
BIT A000	A000 BIT,
BVC xyz	VS IF, (Overflow Set)
	ELSE, (entspricht dem xyz nach BVC)
	THEN,
oder	VS NOT IF, (entspricht xyz nach BVC)
	THEN,

Man muß also darauf achten, daß im Gegensatz zur bisherigen Gewohnheit die jeweils antivalente Verzweigungsbedingung abzufragen ist (der Assembler erzeugt gleichwohl die gewohnte!).

Es wurde bereits erwähnt, daß der FORTH-Assembler in gewissem Umfang auch Symbole akzeptiert. Symbol kann dabei der Inhalt einer Konstanten sein. Wir erinnern, daß Konstante mit der Namensnennung ihren Inhalt auf den Datenstack befördern. Wenn wir also irgendwann erklärt haben

```
HEX A000 CONSTANT PORTB
```

so können wir das später verwerten, z.B. als

```
PORTB LDA,
```

Variable transportieren mit der Namensnennung nur ihre Adresse auf den Datenstack. Ihren Inhalt errreichen wir mit dem Fetch-Befehl @, also z.B.:

```
HEX A001 VARIABLE PORTA
...
PORTA @ LDA,
```

Man beachte jedoch: Als Adresse wird der Variableninhalt assembliert, der bei der Definition das Assemblerwortes bestand (nicht der ggfs. andere während der Ausführungszeit!).

Und es gibt eine dritte Art, wie man für den Assembler Symbole heranzieht: Gesetzt den Fall, wir haben ein Unterprogramm wie folgt definiert

```
CODE UMSCHALT
55   LDA,
PORTB  EOR,
PORTB STA,
RTS,
END-CODE
```

dann können wir unter Assembler mit dem Tick-Befehl (') die Adresse des Moduls UMSCHALT wie folgt heranziehen:

```
' UMSCHALT JSR,
```

Wir können bei einem anderen Modul sogar einen um 8 Bytes späteren Einsprungspunkt durch Adressenrechnung relativieren:

```
' AUSSCHALT 8 + JSR,
```

Der FORTH-Assembler des AIM 65 jedenfalls macht diese Spiele mit, und wir kommen damit wieder etwas mehr in die Nähe symbolischer Programmierung. Es ist sogar ausdrücklich darauf hinzuweisen, daß das normale FORTH-Vokabular auch in CODE-Definitionen benutzt werden kann, so daß für das Adressenrechnen alle 4 Grundrechenarten zur Verfügung stehen. Mit den weiteren Hilfsmitteln der Sprache ist es auch möglich, in FORTH einen Makro-Assembler zu schreiben, auch eine bedingte Assemblierung fällt dann nicht mehr schwer.

Man beachte, das vom FORTH-Assembler aufgerufene Unterprogramme immer mit RTS enden und nicht etwa mit NEXT JMP,.

In einem besonderen Abschnitt werden wir die Anwendung des FORT-Assemblers für die Interfacesteuerung darstellen, so daß hier als Beispiel nur einige kleine Programmablaufsteuerungen gegeben werden sollen. Der Leser sei ausdrücklich ermutigt, sich des Assemblers zu bedienen, die Umgewöhnung dauert nur kurz, man erhält Programmsegmente, die sich oftmals eleganter als unter FORTH ausführen lassen. Und man sollte sich nicht an den wenigen zusätzlichen JMP-Befehlen stören, die er erzeugt und die die Ausführungzeit nur wenig beeinflussen. Ein klarer Vorteil ist der Gewinn an Übersichtlichkeit an den Eckpunkten der Kontrollstrukturen.

Der Assembler des FORTH wird häufig für den 'zweiten Durchgang' empfohlen. Man teste zunächst die logische Programmstruktur in der höheren Sprache FORTH aus, um dann zeitkritische Module in Assembler umzuschreiben. Das ist sicher ein Weg. Wenn man aber den Eindruck hat, ein Modul werde schließlich in Assembler geschrieben sein müssen, dann sollte man sogleich mit einem entsprechenden Programmierversuch beginnen.

```
( ADDITION ZWEIER DATENFELDER)
( F1 UND F2 MIT SUMMENBILDUNG IN F1)
( DIE ANFANGSADRESSEN DER FELDER SIND AUF DEM DATENSTACK)
( UEBERGEBEN WORDEN)

CODE ADDITION  ( ADR ADR -- )
2 # LDA,    ( 2 PARAMETERPAARE IN DIE N-AREA TRANSPORTIEREN)
SETUP JSR,    ( AUSLOESUNG TRANSPORT)
4 # LDY,    ( FELDBREITE 4 BYTES)
            ( Y ZAEHLT UND ADRESSIERT)
CLC,    ( ADDITIONSVORBEREITUNG)
BEGIN,
   N )Y LDA,    ( ADRESSIERUNG INDIREKT,Y)
   N 2+ )Y ADC,    ( 2. OPERAND)
   N )Y STA,    ( ERGEBNISABLAGE)
   DEY,
0= UNTIL,    ( STATUSABFRAGE AUF SCHLEIFENENDE)

NEXT JMP,    ( ZURUECK ZU FORTH, ZUR LAUFZEIT)
END-CODE     ( ZURUECK ZU FORTH NACH DER ASSEMBLIERUNG)
```

```
( BEISPIEL EINER WARTESCHLEIFE)
( WARTEN, SOLGANGE BIT 7 IN PORTB NOCH NICHT GESETZT)
HEX
A000 CONSTANT PORTB    ( DEFINITION SYMBOLISCH UNTER FORTH)

CODE WARTEN    ( ASSEMBLERPROGRAMM)
BEGIN,
   PORTB BIT,
0< UNTIL,    ( WARTESCHLEIFE)
NEXT JMP,    ( ZURUECK ZU FORTH)
END-CODE

CODE WARTEN2    ( ALTERNATIVE LOESUNG)
BEGIN,
   PORTB BIT,    ( ABFRAGE)
0< WHILE,
REPEAT,
NEXT JMP,
END-CODE
```

```
<K>*=437
/10
 0437 A9 LDA #02
 0439 20 JSR B099
 043C A0 LDY #04
 043E 18 CLC
 043F B1 LDA (97),Y
 0441 71 ADC (99),Y
 0443 91 STA (97),Y
 0445 88 DEY
 0446 D0 BNE 043F
 0448 4C JMP B05A

<K>*=462
/03
 0462 2C BIT A000
 0465 10 BPL 0462
 0467 4C JMP B05A

<K>*=476
/04
 0476 2C BIT A000
 0479 10 BPL 047E
 047B 4C JMP 0476
 047E 4C JMP B05A
```

Nebenstehend der erzeugte Maschinen-Code

ADDITION

WARTEN

WARTEN2

(WIRD FORTGESETZT) R.L. #

Thomas Asche, 4790 Paderborn

Dynamische Speicherverwaltung in FORTH

In meinem System (AIM 65, 20K RAM) mit Kassettenrecorder als Massenspeicher benutze ich, um mit meinem Screen-Editor und anderen FORTH-Systemen kompatibel zu sein, das Befehlswort R/W wie es im FORTH Installation Manual (1) beschrieben ist. Da mein virtueller Datenspeicher aber nur 8K groß ist, konnte ich nur 8 Seiten Text auf einmal laden, was für größere Programme bald nicht mehr ausreichend war. Deshalb wurden im neuen Format alle Blanks, die keine Information tragen, beim Speichern ausgeblendet und beim Lesen wieder angehängt.

Dies wird durch das Wort NEW–R/W realisiert, das genau wie in (1) dargestellt arbeitet. Wenn das Programm NEW–R/W aufruft, wird der Text in den Disk-Buffern nicht auf eine Floppy Disk geschrieben, sondern sie wird im Speicher simuliert und beim Laden aus diesem virtuellen Diskspeicher wieder ausgelesen. Nachdem ein Programm geschrieben wurde, kann der Text aus dem virtuellen Speicher auf Kassette gespeichert werden. Und beim Laden vom Kassettenrecorder werden alle Zeiger, Werte und der Text wieder in den Diskspeicher geschrieben.

Da die Textseiten im virtuellen Speicher unterschiedliche Längen haben, ist eine Verwaltung dieses Speichers notwendig. Soll eine Screen, angezeigt durch die Blocknummer (BLK #) in den Discbuffer geschrieben werden (F=1, READ), wird dieser erst mit Blanks gefüllt. Danach wird der Text zeilenweise übertragen. Im anderen Fall (F=0, WRITE) wird der Text vom Discbuffer in den Discspeicher geschrieben. Hierbei ist einiges zu beachten und zu verändern. Als erstes werden die Zeichen im Discbuffer abgezählt und die Kontrollzeichen (0D) gesetzt (SET0D). Die Länge der Screen wird dann mit der Länge der alten Screen im Disc-Speicher verglichen und es wird entschieden, ob der benutzte Speicher wachsen (GROW) bzw. kleiner (COMPRESS) werden muß. Erst nachdem der Inhalt des Disc-Speichers mit allen Zeigern an die neue Länge angepaßt wurde, wird der Text in komprimierter Form vom Discbuffer in die virtuelle Disc geschrieben (BLKWRITE). Sollte der zur Verfügung stehende virtuelle Disc-Speicher verbraucht sein, meldet sich das System mit einer Fehlermeldung 6 (DISC RANGE?).

Mit dem hier vorgestellten Programm erzielte ich mit dem abgebildeten Listing eine Speichereinsparung von 40%.

(1) FIG-FORTH Installation Manual, William F. Rangsdale, Nov 80, FORTH Interest Group, PO Box 1105, San Carlos, CA 94070.

```
                FUNKTIONSZUSAMMENFASSUNG

-CMOVE          ( FROM+COUNT-3,TO+COUNT-2,COUNT-1 --- )
                VERSCHIEBT DEN SPEICHERINHALT DER LAENGE COUNT
                VON ADRESSE-3 ZUR ADRESSE-2.
?DISC           ( BLK#-1 --- )
                ZAEHLT DIE VORHANDENEN VIRTUELLEN BUFFER UND
                EROEFFNET GEGEBENENFALLS NEUE BUFFER.
BLKREAD         ( FROMADR-2,TOADR-1 --- )
                LIEST EINEN BLOCK VOM BUFFER IN DEN DISCBUFFER.
BLKWRITE        ( FROMADR-2,TOADR-1 --- )
                SCHREIBT EINEN DISCBUFFER IN GEPACKTER FORM IN DEN
                BUFFER.
BOUNDS          ( ADR-2,COUNT-1 --- ADR+COUNT-2,ADR-1 )
                DAS GLEICHE WIE OVER + SWAP .
```

```
COMPRESS      ( POINTADR-3,OLD-2,NEW-1 --- )
              VERKLEINERT DEN DURCH DIE POINTADRESSE ANGEZEIGTE
              BUFFER AUF DIE NEUE LAENGE. OLD>NEW
GROW          ( POINTADR-3,OLD-2,NEW-1 --- )
              DER VIRTUELLE BUFFER WIRD VERGROESSERT. NEW>OLD
DISCPOINTER   ( --- ADR-1 )
              ANFANGSADRESSE DES SPEICHERBEREICHS IN DEM DIE
              VIRTUELLE DISC GESPEICHERT WIRD.
DISCEND?      ( --- ADR-1 )
              ENDADRESSE DES SPEICHERS DER VIRTUELLEN DISC.
INITDISC      ( --- )
              INITIALISIERT DEN ERSTEN DISCPOINTER.
NEW-R/W       ( ADR-3,BLK#-2,F-1 --- )
              DIE GLEICHE FUNKTION WIE R/W IN FIG-FORTH.
NEWDISC       ( POINTADR-1 --- )
              ERDOEFFNET EINEN NEUEN BUFFER.
NONBL         ( FROM-2,TO-1 --- START-3,END-2,TF-1/FF-1 )
              DER SPEICHERBEREICH FROM-TO WIRD NACH BLANKS ABGE
              FRAGT. FLAG=0 WENN NUR BLANKS, ODER FLAG=1 WENN
              KEINE BLANKS IM BEREICH START BIS END.
SETOD         ( BUFFERADR-1 --- COUNT-2,BUFFERADR-1 )
              ZAEHLT DIE ZU UEBERTRAGENDEN ZEICHEN IM DISCBUFFE
              UND SETZT DAS CARRIAGE-RETURN ZEICHEN OD.
TOP           ( --- ADR-1 )
              LETZTE ADRESSE DES ZUR VERFUEGUNG STEHENDEN SPEI-
              CHERS DER VIRTUELLEN DISC.
```

DARSTELLUNG DES VIRTUELLEN DISCSPEICHERS
MIT ZEIGERN UND WERTEN

```
                I-----------------------------------------------I
DISCPOINTER--> I    END   I START-1 I START 2 I START N I 0000   I
               I LAENGE 1 I.......... TEXT SCREEN 1 ............I
               I .........I LAENGE 2 I.........................I
               I .. TEXT SCREEN 2 .........I LAENGE N I........I
               I .... TEXT SCREEN N ...........................I
       END --> I***********************************************I
               I*** NOCH ZUR VERFUEGUNG STEHENDER SPEICHER ****I
               I***********************************************I
       TOP --> I-----------------------------------------------I
```

```
SCR# 1
0   ( INITDISC/SETOD                                1     TA 29.4.82
1   FORTH DEFINITIONS FORGET TASK   HEX
2   LIMIT CONSTANT DISCPOINTER ( START OF VIRTUELL MEMORY )
3   5000  CONSTANT TOP ( TOP OF MEMORY )
4   : INITDISC ( INIT FIRST VIRTUELL BUFFER )
5   DISCPOINTER 4 + DISCPOINTER ! 0 DISCPOINTER 2+ ! ;
6   : SETOD ( ADR-1 --- COUNT-2,ADR-1 )
7   0 SWAP 10 0
8   DO C/L I * OVER + DUP >R ( FROM) C/L BOUNDS ( TO)
9      EDITOR NONBL -DUP
A         IF SWAP DROP 1+ DUP R> - DUP C/L =
B            IF SWAP DROP ELSE OD ROT C! 1+ THEN ROT +
```

```
C          ELSE OD R> C! SWAP 1+ THEN SWAP
D   LOOP ; -->
E
F

SCR# 2
0   ( NEWDISC/COMPRESS                                    TA 29.4.82 )
1   : NEWDISC ( COUNT-3,BADR-2,POINTADR-1 --- COUNT-2,BADR-1   )
2   DISCPOINTER @ ( END) DUP TOP =        ( OPEN NEW DISC-BUFFER)
3   IF 6 ERROR THEN
4   ( POINTADR,END) 2DUP SWAP - ( COUNT) SWAP DUP 2+ ROT
5   ( FROM,TO,COUNT) -CMOVE ( POINTADR) DISCPOINTER 2 OVER +! @
6   0 OVER ! OVER ( NEW POINTADR) ! DISCPOINTER
7   DO 2 I +! 2 +LOOP ;
8   : COMPRESS ( POINTADR-3,OLD-2,NEW-1 ---   )
9   DUP >R - ( OFFSET) >R 2+ DUP ( NEXT POINTADR) @ -DUP NOT
A   IF ( LAST DISC) DISCPOINTER DUP @ R> - SWAP !
B   ( POINTADR,DISCSTART)
C   ELSE DISCPOINTER @ ( END) OVER - ( COUNT) OVER R - SWAP CMOVE
D        DISCPOINTER DUP @ R - SWAP ! ( NEW END) DUP
E        BEGIN DUP @ R - OVER ! 2+ DUP @ 0= UNTIL DROP R> DROP
F   THEN R> SWAP 2- @ ! ; -->

SCR# 3
0   ( GROW/?DISC                                      3   TA 29.4.82 )
1   : GROW ( POINTADR-3,OLD-2,NEW-1 --- )
2   DUP >R ( NEW LENGTH ) SWAP - ( OFFSET) >R DUP 2+ DUP @ 0=
3   IF DROP DISCPOINTER DUP @ R> + DUP TOP >
4      IF R> 6 ERROR THEN SWAP !
5   ELSE
6      DISCPOINTER @ ( END) OVER @ OVER SWAP - OVER R + DUP TOP >
7      IF R> R> 6 ERROR THEN
8      DUP DISCPOINTER ! SWAP 2+ -CMOVE
9      BEGIN DUP @ R + OVER ! 2+ DUP @ 0= UNTIL R> 2DROP
A   THEN
B      R> SWAP @ ! ;
C   : ?DISC ( BLK#-1 --- )
D   DISCPOINTER 2+ SWAP 0 DO DUP I 2 * + DUP @ 0=
E                            IF NEWDISC ELSE DROP THEN
F                         LOOP DROP ; -->

SCR# 4
0   ( BLKWRITE/BLKREAD                                4   TA 29.4.82 )
1   : BLKWRITE ( FROMADR-2,TOADR-1 --- )
2   10 0 DO OVER C/L I * + OVER  ( FROM,TO ) C/L
3        BEGIN >R ( COUNT) OVER C@ DUP 0D =
4              IF ( LINE END) SWAP C! DROP R> 1- C/L SWAP - + 1
5              ELSE OVER C! 1+ SWAP 1+ SWAP R> 1- -DUP
6                   IF 0 ELSE 2DROP C/L + 1 THEN
7              THEN
8        UNTIL
9   LOOP 2DROP ;
A   : BLKREAD ( FROMADR-2,TOADR-1 --- )
B   DUP B/BUF BL FILL SWAP
C   10 0 DO OVER C/L I * + C/L BOUNDS
D           DO DUP C@ DUP 0D = IF DROP LEAVE ELSE I C! THEN 1+
E           LOOP
F   LOOP 2DROP ; -->
```

```
SCR# 5
0   ( NEW-R/W                                          5    TA 29.4.82 )
1   : NEW-R/W ( BUFFERADR-3,BLK#-2,F=1 READ/F=0 WRITE --- )
2   OVER ?DISC ( OPEN DISCBUFFER)
3   IF    ( READ) DISCPOINTER SWAP 2 * + @ DUP @
4         IF ( NOT EMPTY) 2+ SWAP BLKREAD
5         ELSE ( EMPTY) DROP B/BUF BL FILL
6         THEN ;S
7   THEN ( WRITE) SWAP SETOD ROT DISCPOINTER SWAP 2 * + DUP >R
8         ROT OVER @ @ SWAP
9         2DUP < IF GROW
A                ELSE 2DUP > IF COMPRESS ELSE 2DROP DROP THEN
B                THEN
C         R> @ 2+ ( DISCSTART) BLKWRITE ;
D   : DISCEND? ( --- N / LAST USED MEMORY LOCATION+1 )
E   DISCPOINTER @ 0 D. ; -->
F

SCR# 6
0   ( OLD/NEW/DDUMP                                    6    TA 20.5.82 )
1   : OLD '  R/W CFA UR/W ! ;
2   : NEW ' NEW-R/W CFA UR/W ! ;
3   : DDUMP ( ADR-2,COUNT-1 --- )
4   0 DO CR DUP 0 5 D.R SPACE 8 SWAP OVER
5         0 DO DUP C@ 3 .R 1+ LOOP
6         SWAP ?TERMINAL IF LEAVE THEN
7   +LOOP DROP CR ;
8   : TASK ; ;S
9
A
B
C
D
E
F                                                                    #
```

Michael Zimmermann, 6102 Pfungstadt

FORTH im Eigenbau (2)

Im ersten Abschnitt dieser Reihe wurden bereits die in Assemblersprache formulierten Funktionen zur Verwaltung eines FORTH-Interpreters beschrieben und gelistet (Heft 25, Seiten 30 ff.). Aus Platzgründen konnten dort die auch schon vorgestellten Routinen zur Ausführung von Funktionen für den Anwender nicht mehr gelistet werden, die rechnenden Routinen, Lade- und Speicheroperationen und logische Operationen. Sie sind jetzt nachfolgend gelistet, wobei zur Erklärung auf Heft 25 hinzuweisen ist.

```
026C    ------------------------------------------------------------
026E    DROPB DROP BYTE
026E
026E    026E 7002    0083 DROPB  .WOR 8+2
0270    0270 E8      0084        INX           ;STACK-POINTER ERHOEHEN
0271    0271 4C3002  0085        JMP NEXT
0274    ------------------------------------------------------------
```

```
0274        DROPW DROP WORD
0274
0274        0274 4B02    0087 DROPW  .WOR TCALL      ;AUFRUF ALS FORTH-UNTERROUTINE
0276        0276 6E02    0088        .WOR DROPB      ;ERSTES BYTE FALLEN LASSEN
0278        0278 6E02    0089        .WOR DROPB      ;UND ZWEITES BYTE
027A        027A 6302    0090        .WOR TRET
027C        ------------------------------------------------------------
027C        DUPB DUPLICATE BYTE
027C
027C        027C 7E02    0092 DUPB   .WOR *+2        ;CODE
027E        027E CA      0093        DEX             ;DATA-STACK-POINTER ERNIEDRIGEN
027F        027F B502    0094        LDA 2,X         ;ALTEN TOP-OF-STACK WERT HOLEN
0281        0281 9501    0095        STA 1,X         ;UND ALS NEUEN TOP-OF-STACK WERT ABSPEICHERN
0283        0283 4C3002  0096        JMP NEXT
0286        ------------------------------------------------------------
0286        DUPW DUPLICATE WORD
0286
0286        0286 8802    0098 DUPW   .WOR *+2
0288        0288 CA      0099        DEX             ;STACK-POINTER UM 2BYTE ERNIEDRIGEN
0289        0289 CA      0100        DEX
028A        028A B503    0101        LDA 3,X         ;ALTES TOP-OF-STACK-WORD ENTNEHMEN
028C        028C 9501    0102        STA 1,X         ;UND ALS NEUEN TOP ABSPEICHERN
028E        028E B504    0103        LDA 4,X
0290        0290 9502    0104        STA 2,X
0292        0292 4C3002  0105        JMP NEXT
0295        ------------------------------------------------------------
0295        SWAPB SWAP BYTE
0295
0295        0295 9702    0107 SWAPB  .WOR *+2
0297        0297 B502    0108        LDA 2,X         ;NEXT-ON-STACK IN A LADEN
0299        0299 B401    0109        LDY 1,X         ;TOP-OF-STACK IN Y
029B        029B 9501    0110        STA 1,X         ;JETZT A IN TOP
029D        029D 9402    0111        STY 2,X         ;UND Y IN NEXT-ON-STACK ABLEGEN
029F        029F 4C3002  0112        JMP NEXT
02A2        ------------------------------------------------------------
02A2        SWAPW SWAP WORDS
02A2
02A2        02A2 A402    0114 SWAPW  .WOR *+2        ;CODE
02A4        02A4 B503    0115        LDA 3,X         ;NEXT-ON STACK IN A
02A6        02A6 B401    0116        LDY 1,X         ;TOP IN Y LADEN
02A8        02A8 9501    0117        STA 1,X         ;A IN TOP
02AA        02AA 9403    0118        STY 3,X         ;Y IN NEXT-ON-STACK ABLEGEN
02AC        02AC B504    0119        LDA 4,X         ;WEITER MIT ZWEITEM BYTE
02AE        02AE B402    0120        LDY 2,X
02B0        02B0 9502    0121        STA 2,X
02B2        02B2 9404    0122        STY 4,X
02B4        02B4 4C3002  0123        JMP NEXT
02B7        ------------------------------------------------------------
02B7        INCR-ROUTINE
02B7
02B7        02B7 B902    0125 INCR   .WOR *+2
02B9        02B9 F601    0126        INC 1,X
02BB        02BB D002    0127        BNE *+4
02BD        02BD F602    0128        INC 2,X
02BF        02BF 4C3002  0129        JMP NEXT
02C2        ------------------------------------------------------------
```

```
02C2        DECR DECREMENT-ROUTINE
02C2
02C2        02C2 C402    0131 DECR   .WOR *+2
02C4        02C4 B501    0132        LDA 1,X
02C6        02C6 D601    0133        DEC 1,X          ;ERSTES ZEICHEN ERNIEDRIGEN
02C8        02C8 C900    0134        CMP #00
02CA        02CA D002    0135        BNE *+4          ;ZWEITES ZEICHEN UEBERSPRINGEN WENN KEIN NULL
02CC        02CC D602    0136        DEC 2,X          ;ZWEITES BYTE ERNIEDRIGEN
02CE        02CE 4C3002  0137        JMP NEXT
02D1        ------------------------------------------------------------
02D1        PLUS ADDITION AUF STACK
02D1
02D1        02D1 D302    0139 PLUS   .WOR *+2         ;CODE
02D3        02D3 2481    0140        BIT DFLAG
02D5        02D5 1001    0141        BPL *+3
02D7        02D7 F8      0142        SED
02D8        02D8 18      0143        CLC
02D9        02D9 B503    0144        LDA 3,X          ;ERSTES BYTE VERARBEITEN
02DB        02DB 7501    0145        ADC 1,X
02DD        02DD 9503    0146        STA 3,X
02DF        02DF B504    0147        LDA 4,X
02E1        02E1 7502    0148        ADC 2,X
02E3        02E3 9504    0149        STA 4,X
02E5        02E5 E8      0150        INX
02E6        02E6 E8      0151        INX
02E7        02E7 D8      0152        CLD
02E8        02E8 4C3002  0153        JMP NEXT
02EB        ------------------------------------------------------------
02EB        MINUS SUBTRAKTION AUF STACK
02EB
02EB        02EB ED02    0155 MINUS  .WOR *+2
02ED        02ED 2481    0156        BIT DFLAG
02EF        02EF 1001    0157        BPL *+3
02F1        02F1 F8      0158        SED
02F2        02F2 38      0159        SEC
02F3        02F3 B503    0160        LDA 3,X
02F5        02F5 F501    0161        SBC 1,X
02F7        02F7 9503    0162        STA 3,X
02F9        02F9 B504    0163        LDA 4,X
02FB        02FB F502    0164        SBC 2,X
02FD        02FD 9504    0165        STA 4,X
02FF        02FF E8      0166        INX
0300        0300 E8      0167        INX
0301        0301 D8      0168        CLD
0302        0302 4C3002  0169        JMP NEXT
0305        ------------------------------------------------------------
0305        PUSHB PUSH BYTE
0305
0305        0305 0703    0171 PUSHB  .WOR *+2
0307        0307 CA      0172        DEX              ;ERNIEDRIGE STACK-POINTER
0308        0308 A000    0173        LDY #0
030A        030A B182    0174        LDA (PC),Y       ;LADE BYTE, AUF DAS DER PROGRAMM-COUNTER ZEIGT
030C        030C 9501    0175        STA 1,X UND SPEICHERS ES IM STACK
030E        030E E682    0176        INC PC           ;ERHOEHE PROGRAMM-COUNTER UM 1
0310        0310 9002    0177        BCC *+4
0312        0312 E683    0178        INC PC+1
0314        0314 4C3002  0179        JMP NEXT
0317        ------------------------------------------------------------
```

```
0317        PUSHW PUSH WORD
0317
0317        0317 1903    0181 PUSHW  .WOR *+2
0319        0319 CA      0182        DEX             ;VERMINDERE STACK-POINTER
031A        031A CA      0183        DEX
031B        031B A000    0184        LDY #0
031D        031D B182    0185        LDA (PC),Y      ;LADE WORT, AUF DAS DER PROGRAMM-COUNTER ZEIGT
031F        031F 9501    0186        STA 1,X         ;UND SPEICHERE ES IM STACK
0321        0321 C8      0187        INY
0322        0322 B182    0188        LDA (PC),Y
0324        0324 9502    0189        STA 2,X
0326        0326 18      0190        CLC
0327        0327 A582    0191        LDA PC          ;ERHOEHE PROGRAMM-COUNTER UM 2
0329        0329 6902    0192        ADC #2
032B        032B 8582    0193        STA PC
032D        032D 9002    0194        BCC *+4
032F        032F E683    0195        INC PC+1
0331        0331 4C3002  0196        JMP NEXT
0334        ------------------------------------------------------------
0334        FETB FETCH BYTE
0334
0334        0334 3603    0198 FETB   .WOR *+2
0336        0336 B501    0199        LDA 1,X         ;KOPIERE ADRESSE AM TOP-OF-STACK
0338        0338 8584    0200        STA PTR         ;IN DEN POINTER
033A        033A B502    0201        LDA 2,X
033C        033C 8585    0202        STA PTR+1
033E        033E E8      0203        INX             ;ERHOEHE STACK-POINTER
033F        033F A000    0204        LDY #0
0341        0341 B184    0205        LDA (PTR),Y     ;LADE BYTE AUS SPEICHER
0343        0343 9501    0206        STA 1,X         ;UND LEGE IM TOP AB
0345        0345 4C3002  0207        JMP NEXT
0348        ------------------------------------------------------------
0348        FETW FETCH WORD
0348
0348        0348 4A03    0209 FETW   .WOR *+2        ;CODE
034A        034A B501    0210        LDA 1,X         ;KOPIERE ADRESSE IM TOP
034C        034C 8584    0211        STA PTR         ;IN DEN POINTER
034E        034E B502    0212        LDA 2,X
0350        0350 8585    0213        STA PTR+1
0352        0352 A000    0214        LDY #0
0354        0354 B184    0215        LDA (PTR),Y HOLE BYTE AUS SPEICHER
0356        0356 9501    0216        STA 1,X UND LEGE IM STACK AB
0358        0358 C8      0217        INY
0359        0359 B184    0218        LDA (PTR),Y
035B        035B 9502    0219        STA 2,X
035D        035D 4C3002  0220        JMP NEXT
0360        ------------------------------------------------------------
0360        STB STORE BYTE
0360
0360        0360 6203    0222 STB    .WOR *+2
0362        0362 B501    0223        LDA 1,X         ;KOPIERE TOP-OF-STACK
0364        0364 8584    0224        STA PTR         ;IN POINTER
0366        0366 B502    0225        LDA 2,X
0368        0368 8585    0226        STA PTR+1
036A        036A E8      0227        INX
036B        036B E8      0228        INX
036C        036C B501    0229        LDA 1,X         ;LADE TOP UND
036E        036E A000    0230        LDY #0
```

```
0370        0370 9184    0231        STA (PTR),Y     ;LEGE IN SPEICHER DURCH POINTER ADRESSIERT AB
0372        0372 E8      0232        INX             ;ERHOEHE STACK-POINTER
0373        0373 4C3002  0233        JMP NEXT
0376        ------------------------------------------------------------
0376        STW STORE WORD
0376
0376        0376 7B03    0235 STW    .WOR *+2
0378        0378 B501    0236        LDA 1,X         ;KOPIERE ADRESSE IN TOP
037A        037A 8584    0237        STA PTR         ;IN DEN POINTER
037C        037C B502    0238        LDA 2,X
037E        037E 8585    0239        STA PTR+1
0380        0380 E8      0240        INX
0381        0381 E8      0241        INX
0382        0382 B501    0242        LDA 1,X         ;LADE TOP
0384        0384 A000    0243        LDY #0
0386        0386 9184    0244        STA (PTR),Y UND SPEICHERE IN MEMORY
0388        0388 C8      0245        INY
0389        0389 B502    0246        LDA 2,X
038B        038B 9184    0247        STA (PTR),Y
038D        038D E8      0248        INX             ;ERHOEHE STACK
038E        038E E8      0249        INX
038F        038F 4C3002  0250        JMP NEXT
0392        ------------------------------------------------------------
0392        SAND STACK - AND
0392
0392        0392 9403    0252 SAND   .WOR *+2
0394        0394 B501    0253        LDA 1,X         ;LADE TOP
0396        0396 3502    0254        AND 2,X         ;VERKNUEPFE MIT NEXT
0398        0398 E8      0255        INX             ;UEBERGEHE ALTEN TOP
0399        0399 9501    0256        STA 1,X         ;SPEICHERE RESULTAT IN TOP
039B        039B 4C3002  0257        JMP NEXT
039E        ------------------------------------------------------------
039E        SOR STACK - OR
039E
039E        039E A003    0259 SOR    .WOR *+2
03A0        03A0 B501    0260        LDA 1,X         ;LADE TOP
03A2        03A2 1502    0261        ORA 2,X         ;VERKNUEPFE MIT NEXT
03A4        03A4 E8      0262        INX             ;UEBERGEHE ALTEN TOP
03A5        03A5 9501    0263        STA 1,X         ;SPEICHERE RESULTAT IN TOP
03A7        03A7 4C3002  0264        JMP NEXT
03AA        ------------------------------------------------------------
03AA        SEOR STACK -EXCLUSIE OR
03AA
03AA        03AA AC03    0266 SEOR   .WOR *+2
03AC        03AC B501    0267        LDA 1,X         ;LADE TOP
03AE        03AE 5502    0268        EOR 2,X         ;VERKNUEPFE MIT NEXT
03B0        03B0 E8      0269        INX             ;UEBERGEHE ALTEN TOP
03B1        03B1 9501    0270        STA 1,X SPEICHERE RESULTAT IN TOP
03B3        03B3 4C3002  0271        JMP NEXT
03B6        ------------------------------------------------------------
03B6        DOT AUSGABE WORD IM TOP ALS ZAHL
03B6
03B6        03B6 BB03    0273 DOT    .WOR *+2        ;CODE
03B8        03B8 B502    0274        LDA 2,X         ;LADE BYTE IM TOP
03BA        03BA 2046EA  0275        JSR NUMA        ;GEBE ALS HEXA-ZAHL AUS
03BD        03BD B501    0276        LDA 1,X
03BF        03BF 2046EA  0277        JSR NUMA
03C2        03C2 E8      0278        INX             ;MACHE TOP UNGUELTIG
```

```
03C3        03C3 E8      0279        INX
03C4        03C4 4C3002  0280        JMP NEXT
03C7
03C7        ------------------------------------------------------------
03C7        BNUM BRACKET-NUMBER NUMERISCHE EINGABE
03C7
03C7        03C7 C903    0282 BNUM   .WOR *+2          ;CODE
03C9        03C9 8680    0283        STX XSAV          ;RETTE X
03CB        03CB 20B1EA  0284        JSR ADDNE         ;HEXA-ZAHL HOLEN
03CE        03CE A680    0285        LDX XSAV          ;X REKONSTRUIEREN
03D0        03D0 CA      0286        DEX               ;UND PLATZ AUF STACK MACHEN
03D1        03D1 CA      0287        DEX
03D2        03D2 AD1CA4  0288        LDA ADDR          ;EINGEGEBEN ZAHL IM STACK ABLEGEN
03D5        03D5 9501    0289        STA 1,X
03D7        03D7 AD1DA4  0290        LDA ADDR+1
03DA        03DA 9502    0291        STA 2,X
03DC        03DC 4C3002  0292        JMP NEXT
03DF
03DF        ------------------------------------------------------------
03DF        SPACE AUSGABE SPACE
03DF
03DF        03DF E103    0294 SPACE  .WOR *+2
03E1        03E1 203EE8  0295        JSR BLANK         ;LINK ZUR AIM-ROUTINE
03E4        03E4 4C3002  0296        JMP NEXT
03E7
03E7        ------------------------------------------------------------
03E7        CR CARRIAGE RETURN
03E7
03E7        03E7 E903    0298 CR     .WOR *+2
03E9        03E9 20F0E9  0299        JSR CRLF          ;LINK ZUR AIM-ROUTINE'
03EC        03EC 4C3002  0300        JMP NEXT
03EF
03EF        ------------------------------------------------------------
03EF        KEY KEYBOARD-ZEICHEN EINGABE
03EF
03EF        03EF F103    0302 KEY    .WOR *+2          ;CODE
03F1        03F1 CA      0303        DEX               ;PLATZ AUF STACK MACHEN
03F2        03F2 2093E9  0304        JSR INALL         ;ZEICHEN EINLESEN
03F5        03F5 9501    0305        STA 1,X AUF STACK ABLEGEN
03F7        03F7 4C3002  0306        JMP NEXT
03FA
03FA        ------------------------------------------------------------
03FA        EMIT ZEICHEN VOM STACK AUSGEBEN
03FA
03FA        03FA FC03    0308 EMIT   .WOR *+2          ;CODE
03FC        03FC B501    0309        LDA 1,X           ;ZEICHEN LADEN
03FE        03FE 20BCE9  0310        JSR OUTALL        ;UND AUSGEBEN
0401        0401 E8      0311        INX               ;STACK VERKUERZEN
0402        0402 4C3002  0312        JMP NEXT
0405
0405        ------------------------------------------------------------
0405        RUN-PACKAGE
0405
0405        0405         0314        *=$800
0800        0800 4B02    0315        .WOR TCALL
0802        0802 0503    0316        .WOR PUSHB        ;NULL AUF STACK BRINGEN
0804        0804 00      0317        .BYT 0
0805        0805 C703    0318        .WOR BNUM         ;ADRESSE=ZAHL VON KEYBOARD EINGEBEN
0807        0807 3403    0319        .WOR FETB         ;BYTE AN ADRESSE HOLEN
0809        0809 0503    0320        .WOR PUSHB        ;MASKE 7F AUF STACK LEGEN
080B        080B 7F      0321        .BYT $7F
080C        080C 9E03    0322        .WOR SOR          ;UND LOGISCHES OR AUSFUEHREN
080E        080E B702    0323        .WOR INCR         ;UM 1 ERHOEHEN UM UEBERLAUF ZU ERHALTEN
0810        0810 6E02    0324        .WOR DROPB        ;NIEDERES BYTE VERGESSEN
```

```
0812        0812 0503   0325        .WOR PUSHB      ;MASKE $30=0 AUF STACK BRINGEN
0814        0814 30     0326        .BYT $30
0815        0815 9E03   0327        .WOR SOR        ;UND HIERMIT LOGISCHES OR AUSFUEHREN
0817        0817 DF03   0328        .WOR SPACE      ;LEERZEICHEN AUSGEBEN
0819        0819 FA03   0329        .WOR EMIT       ;BYTE IN TOP ALS CHARACTER AUSGEBEN
081B        081B 6302   0330        .WOR TRET
081D        081D        0331        .END
081D          ERRORS= 000                                                          #
```

StDir. Peter Rix, 2350 Neumünster

Das AIM 65 Math Package

Das Package besteht aus einem 4K-ROM für den Adreßbereich D000-DFFF und aus einem 80-seitigen Handbuch (Rockwell Document Nr. 29651N22 vom Februar 1982). Der Verfasser hatte Gelegenheit, dieses Package zu erproben und danach diesen Erfahrungsbericht abzugeben.

Im RAM werden die Variablenbereiche AB-C4 und 25C-27E beansprucht. Von der Adreßraumbelegung abgesehen, sind die ROM-Routinen als systemunabhängige Unterprogramme gestaltet, so daß eine Nutzung des Package in beliebigen 65xx-Systemen möglich ist.

Als Dienstleistungen bietet das Package Arithmetik mit 40-Bit-Fließkommazahlen (9 Dezimalstellen, Zahlenbereich ±10↑-39 bis ±10↑38), Quadratwurzel, Potenz, Betrag, Integerfunktion, Vorzeichenfunktion, Exponentialfunktion, dekadischen und natürlichen Logarithmus, trigonometrische Funktionen, Arcustangens, Grad-Radiantenumwandlung und Polynomberechnung. Hinzu kommen Routinen zur ASCII-Fließkommakonvertierung, Ausgabeformatierung, 16-Bit-Integer-Konvertierung und Fehlerbehandlung.

Rockwell ist den sicheren Weg gegangen, die langjährig in vielen Microsoft-BASIC-Interpretern bewährten Mathematikprogramme in sein Package zu übernehmen, die Routinen arbeiten denn auch mit der vom BASIC her gewohnten Zuverlässigkeit und mit den bekannten kleinen Rundungsschwächen aller rein binären Formate. Beachtlich ist die Arbeitsgeschwinddigkeit der vom Verwaltungsüberhang des BASICinterpreters befreiten Routinen, eine Multiplikation erfordert typisch etwa 2 ms, ein Polynom 2. Grades ist in 5 ms berechnet.

Gut gelöst ist die Fehlerbehandlung. Über einen Vektor kann zu einer ROMresidenten Fehlermeldung oder zu einer benutzerspezifischen Fehlerbehandlung verzweigt werden. Über den X-Index werden die Fehlerarten DIVIDE BY ZERO, OVERFLOW, NON PLUS LOG und NUMBER TOO LARGE signalisiert. Die im Handbuch dokumentierte Fehlerart UNDERFLOW kann allerdings nicht eintreten, weil Zahlen kleiner 10E-39 als Null gedeutet werden.

Für den Assemblerprogrammierer werden Parameterübergabe, Einsprungadressen und Wirkung der einzelnen Routinen vollständig dokumentiert. Das hektographierte Manuskript des Handbuches enthält allerdings noch einige kleine Fehler und Unklarheiten. So liefert z.B. die ausgewiesene Negationsroutine statt eines veränderten Vorzeichens eine komplementierte Mantisse mit verblüffenden Werten ab.

Einsatzschwerpunkt des Package dürfte die Erweiterung des FORTH-Systems um Gleitkommabefehle werden. Die Adreßraumbelegung ist auf diese Anwendung abgestimmt, und die Einbindung in FORTH ist prinzipiell gut gelöst. Nach Installtion des ROMs im Sockel Z24 der AIM-Platine kann FORTH über Taste N aufgerufen werden. Dadurch werden 32 im ROM bereits enthaltene Gleitkomma-FORTH-Worte zum FORTH-Vokabular hinzugefügt. Die gute Idee hat aber im Detail noch erhebliche Schwächen, weil fast alle Befehle akkumulator-orientiert sind und wenig Rücksicht auf die Stackstruktur von FORTH nehmen. Alle Operanden müssen einzeln zwischen Memory und Fließkomma-Akkus bewegt werden. Das ist lästig. Ein Gleitkommastapel begrenzter Tiefe (ca. 6 Register) und stapelorientierte Arithmetik sind für FORTH unbedingt notwendig.

Vorversuche haben gezeigt, daß die Einrichtung eines solchen Stapels einfach möglich ist. Mit über 500 freien Byte im ROM steht dafür auch genügend Programmplatz zur Verfügung.

Wünschenswert bleibt eine verbesserte Formatierroutine für die Ausgabe. Zwar kann der Anwender Ausgabefeldlänge und die Zahl der Nachkommastellen freizügig wählen, er hat aber, weil auch hier die BASIC-Routinen übernommen worden sind, keinen Einfluß auf die Umschaltung zwischen dezimaler und exponentieller Schreibweise. Dieser Standard mag 1978 akzeptabel gewesen sein, er wird den Anforderungen von 1982 nicht mehr gerecht.

Hinzuweisen bleibt auf einige simple Fehler im ROM. Die Standard-Fehlerroutine wird nicht ordnungsgemäß beendet, bei Dezimalzahlen mit Beträgen 0,01≤ \|x\| <1 geht das Vorzeichen in der Ausgabe verloren, beim Abschneiden von Endnullen verschwindet auch immer das letzte noch bedeutsame Zeichen, und die Umschaltung auf das Abschneiden von Endnullen erfolgt über die falsche Adresse.

Zusammenfassend verspricht das MATH PACKAGE von Rockwell ein gut dokumentiertes, für den Assemblerprogrammierer nützliches und für den FORTH-Programmierer unentbehrliches Dienstleistungspaket zu werden. Die Konzeption ist im Ansatz richtig, in die Details ist noch etwas zu wenig Arbeit investiert worden. Es bleibt zu hoffen, daß Rockwell in der freigegebenen Version noch eine Überarbeitung im Hinblick auf Benutzerkomfort vornehmen wird. Das Verhältnis zwischen Rockwell-Schale und Microsoft-Inhalt ist gegenwärtig noch nicht ausgeglichen genug (Spätlese im Pappbecher).

Nützlich im Math-Package ist die Ausweisung auch der im BASIC bereits enthaltenen Polynomberechnungsroutinen. Dazu nachfolgend als Anwendung ein kleines FORTH-Programm:

```
( RIX, NMS, 07/82 )
( POLYNOMBERECHNUNG MIT AIM-65 MATH. PACKAGE )

( MINIMALE FELDWEITE VORGEBEN )
14 MIN-WIDTH !

( NACHKOMMASTELLEN VORGEBEN )
6 DEC-LENGTH !

0 VARIABLE ASCII-BUFFER C/L ALLOT

: FZIN ( EINGABE --> FAC)
       ASCII-BUFFER DUP C/L
       CR ." X=" EXPECT FIN ;

: FZOUT ( FAC --> AUSGABE)
        ASCII-BUFFER DUP FOUT
        CR ." Y=" COUNT TYPE CR ;

( EINGABE POLYNOMGRAD N UND N+1 KOEFFIZIENTEN
( DYNAMISCHE SPEICHERZUWEISUNG
( N, CN, ... C2, C1, C0 AB PAD )
: KIN ( N --- )
      DUP PAD C!
      1+ 0 ( LAUFINDEX)
      DO ASCII-BUFFER DUP C/L
         CR EXPECT FIN
         PAD 1+ I 5 * + F>M
      LOOP ;
```

```
( BERECHNUNG VON POLYNOMWERTEN )
: PWERT FZIN ( ARGUMENTEINGABE)
        PAD POLY ( RECHNUNG)
        FZOUT ( WERTAUSGABE) ;

FINIS

END
<
 >?
<N>
 AIM 65 FORTH V1.3
SOURCE IN=M
OK
( POLYNOM 2. GRADES  Y=-2.25X^2+8.732X-0.511 ) OK
 2 KIN ( GRAD N UND KOEFFIZIENTEN )
-2.25
 8.732
-0.511 OK
PWERT ( BERECHNUNG )
X= 0
Y=       -.511000
OK
PWERT
X= 2
Y=       7.953000
OK
PWERT
X=-12.5
Y=    -461.223500
OK
PWERT
X= 5.5E21
OVERFLOW ERROR

 AIM 65 FORTH V1.3                                              #
```

Dirk Sanders, 6100 Darmstadt

Low Cost Typenraddrucker (2)

Der Autor beschrieb in Heft 24 des 65xx MICRO MAG Schaltung und Programm zum Betrieb der Typenrad-Schreibmaschinen Olivetti Praxis 30 und 35 sowie von Quelle, Best.-Nr. 027.304. Zwischenzeitlich konnte eine Verbesserung vorgenommen werden, und zwar hinsichtlich des dort noch nicht für alle Fälle befriedigend zu bemessenden Zeitbedarfes für Wagenrücklauf/Zeilentransport. Während dieses CR/LF oder eines TAB läuft der Step-Motor. Bedingung für das Ende des CR ist nun, daß auf der ersten der vier Step-Motor-Leitungen 50 ms lang kein Impuls registriert wird, dann nämlich ist der Step-Motor nicht mehr in Betrieb.

Dafür muß eine der vier Step-Motor-Leitungen nun wie folgt angeschlossen werden:

CA1 an STEP1 =3872 (MASTER), dort Pin 3

Das Programm ist an den nachfolgend bezeichneten Stellen auf diese neue Beschaltung geändert worden. Es ist zu beachten, daß sich durch die neuen Befehle für den Rest des Programmes Verschiebungen im benutzten Adreßraum ergeben. Das Programm setzt ferner für ein LF allein eine Wartezeit von 200 ms voraus, was in etwa der von Olivetti programmierten Zeit entspricht.

Bei den Schreibmaschinen scheint die Bezeichnung der ICs MASTER und SLAVE z.T. von den bisher genannten abweichend zu sein. Festgestellt wurden auch:

MASTER	3870/42 MK16139N-5
SLAVE	3870/20 MK14567N-5

FÜR NACHSTEHENDES PROGRAMM WIRD EIN WEITERES SYMBOL EINGEFÜHRT:

```
TIME3       =50000      ;1/4 WARTEZEIT FÜR LF
```

DIE 6 BEFEHLE AB PROGRAMMSTELLE $0EA0 WERDEN WIE FOLGT ABGEÄNDERT:

```
            AD0CA0 LDA PCR
            29EE   AND ß$EE
            8D0CA0 STA PCR        ;CB1, CA1 : HIGH TO LOW
            AD0EA0 LDA IER
            29CD   AND ß$CD
            8D0EA0 STA IER        ;KEIN INTERRUPT
```

AB PROGRAMMSTELLE $0EFF FOLGEN JETZT DIESE BEFEHLE, DURCH DIE DER REST DES PROGRAMMES AB LABEL RETURN VERSCHOBEN WIRD:

```
            68     PLA            ;WAR ES EINE FUNKTIONS-
            C96F   CMP ß$6F       ;TASTE, DANN KOMMT KEIN
            F027   BEQ SPACE      ;READY-IMPULS
            C96E   CMP ß$6E
            F030   BEQ LFEED
            C965   CMP ß$65
            F038   BEQ BS
            C978   CMP ß$78
            F034   BEQ TABULA
            C976   CMP ß$76
            F030   BEQ CR
            2970   AND ß$70       ;WENN TIME=0,
            F013   BEQ FUNKT
            C970   CMP ß$70       ;ODER TIME=7 DANN WAR ES
            F00F   BEQ FUNKT      ;EINE FUNKTIONSTASTE

            A901   LDA ß$01       ;WENN NICHT, DANN
PRI08       2C01A0 BIT ORRA       ;WARTE BIS DRUCKMAGNET...
            F0FB   BEQ PRI08      ; ...ON?
PRI09       2C01A0 BIT ORRA
            D0FB   BNE PRI09      ; ...WIEDER OFF?
            4CXXXX JMP RETURN


            ;BEI FUNKTIONSTASTE TIME1 WARTEN
SPACE                             ;TIME1 ENTSPRICHT TIME
                                  ;FUER SPACE, WIRD ABER
                                  ;FUER VIELE FUNKTIONS-
FUNKT       A0A8   LDY ßkTIME1    ;TASTEN BENUTZT
            A961   LDA ßgTIME1
            8C08A0 STY T2
            8D09A0 STA T2+1       ;USER TIMER 2 STARTEN
            4C XXXX JMP RETURN

            ;LANGDAUERNDE FUNKTIONSTASTEN:
LFEED       A050   LDY ßkTIME3    ;MINDESTENS 200 MS FUER
            A203   LDX ß3         ;ZEILEN VORSCHUB
```

```
LFE00   A9C3    LDA ßgTIME3     ;HIER ERST 150
        20XXXX  JSR TIMER
        CA      DEX
        D0F8    BNE LFE00
TABULA
        ;WARTEN BIS TAB-MOTOR STILL STEHT
CR
BS
LFE01   2C01A0  BIT ORRA        ;CLEAR CA1 INT.-FLAG
        A050    LDY ßkTIME3
        A9C3    LDA ßgTIME3     ;HIER WEITERE 50 MS FUER
        20XXXX  JSR TIMER       ;ZEILENVORSCHUB
        A902    LDA ß$02
        2C0DA0  BIT IFR
        D0EF    BNE LFE01       ;SCHALTFLANKE REGISTRIERT

RETURN  A920    LDA ß$20        ;$20 WIRKT BEI RUECKKEHR
         ERRORS= 0000
```

In TAB, 2. Zeile:

```
| .BYT $65,$78,$76,$6F,$6F,$6E,$6F,$6F ;08
```

#

- Alle Angaben ohne Gewähr -

Jürgen Rüd, 7800 Freiburg

PETARI (CBM)

für CBM 3001 (4001) mit Grafik-Tastatur

Jeder CBM-Besitzer, der sich bereits mit dem Atari-Computer beschäftigt hat, wird sich über kurz oder lang über seinen CBM ärgern. Ganz abgesehen von den tollen grafischen Möglichkeiten besitzt der Atari noch weitere hervorstechende Merkmale. Einige dieser Merkmale, die dem CBM beigebracht werden sollen, betreffen den Screen-Editor. So verfügt der Atari über die Möglichkeit, ganze Zeilen einzufügen oder zu löschen, wobei der Rest entsprechend nach oben bzw. nach unten verschoben wird. Dies hat den Vorteil, daß man bei Änderungen an einem Programm sich im Listing schnell Platz für Einfügungen schaffen kann. Weiterhin besitzt er einen überaus komfortablen Tabulator und natürlich auch einen Tastenrepeat. Diese Hilfsmittel gaben bei mir den Ausschlag, dem etwa stiefmütterlichen CBM etwas auf die Sprünge zu helfen.

Außer dem bereits beschriebenen Insert und Delete einer Zeile folgten so noch weitere Zutaten: Im Gegensatz zum normalen Delete sollte es auch möglich sein, ein Zeichen rechts vom Cursor zu löschen und den Rest nach links aufzuschieben (siehe dazu auch 65xx MICRO MAG Nr. 21, S. 61). Ferner sollte das Löschen des Bildschirms vom Cursor ab und bis zum Cursor möglich sein. Und dasselbe auch in einer Zeile; per Tastendruck kann ab Cursor der Rest der Zeile gelöscht werden oder aber der Beginn der Zeile bis zum Cursor. Auch ein Tabulator, bei dem der Cursor bei jedem Tastendruck um 15 Stellen weiterspringt, kann ganz nützlich sein. Um die HOME-Funktion zu ergänzen, kam noch eine Taste hinzu, mit der man den Cursor auf den Anfang der letzten Zeile versetzen kann. Und schließlich und endlich eine Taste, bei deren Drücken der Rechner von Groß/Grafik auf Klein/Groß-Zeichen und umgekehrt umschaltet. Daß das Programm eine Repeat-Funktion enthält, ist heutzutage schon Standard, genauso wie seine automatische Desaktivierung bei Kassettenoperationen, da es im System-IRQ läuft.

Beim Betrachten der CBM-Tastatur zeigte sich aber schon wieder eine Schäche des CBM: Er hat keinerlei freie Funktionstaste, die man mit den neuen Funktionen belegen könnte. Hier half aber ein kleiner Kniff. Außen den Shift-Tasten, die man übrigens getrennt softwaremäßig abfragen kann,

besitzt der Rechner nur noch die RUN–STOP-Taste und Die RVS-Taste, die alleine gedrückt, keine sofortige Wirkung auf den Bildschirm zur Folge haben. Deshalb wurde der RUN/STOP-Taste die Routine zum Löschen eines Zeichens rechts vom Cursor zugeordnet. Die RVS-Taste dient zu weiteren Dekodierung für die anderen Funktionen. Dafür wurden noch die Tasten CLR, Cursor Down, Cursor Right, Delete, 7, 8, 9 / und ↑ ausgewählt, da diese vorteilhaft im 10er-Block liegen. Das sieht nun praktisch so aus: Soll eine der Funktionen ausgeführt werden, so drückt man die RVS-Taste und gleichzeitig die zur Funktion gehörende Taste. Hierzu die nachstehende Tabelle aller Funktionen von PETARI:

Kürzel	Tastendruck	Beschreibung
CLA	RVS + CLR	löscht Bildschirm bis Cursor
CLR	RVS + 7	löscht Bildschirm ab Cursor
DEZ	RVS + CSRDOWN	löscht Zeile und schiebt den Rest hoch
INZ	RVS + 8	fügt eine Zeile ein und schiebt den Rest nach unten
TAB	RVS + CSRRIGHT	entspricht 15 * Cursor rechts
END	RVS + 9	setzt Cursor an den Anfang der letzten Zeile
DEB	RVS + DEL	löscht Anfang der Zeile bis Cursor
DER	RVS + /	löscht Rest der Zeile ab Cursor
SWI	RVS + ↑	schaltet zwischen Grafik und Kleinschreibung um
DRC	RUN/STOP	löscht 1 Zeichen rechts vom Cursor

Die RVS-Taste funktioniert übrigens normal, solange sie alleine gedrückt wird. Wird eine Funktionstaste gleichzeitig gedrückt, so bleibt der RVS-Modus solange ausgeschaltet, wie die RVS-Taste gedrückt bleibt. Erst mit nochmaligem Drücken der RVS-Taste kann der RVS-Modus wieder eingeschaltet werden. - Zur Repeat-Funktion ist zu bemerken, daß sie sich verlangsamt, wenn der Cursor in der letzten Zeile angekommen ist und CRSDOWN gedrückt wird. Bei RVS spricht die Routine ebenfalls nicht an.

Zum leichteren Verständnis des Programms noch einige Anmerkungen zum Bildschirm-Handling des CBM: Der Rechner überprüft im IRQ, ob eine Taste gedrückt ist. Wenn ja, dann weist er dieser einen Zahlenwert zu, der auch über PEEK(166) abzufragen ist. Dieser wird auch nach TAST gespeichert. Nun überprüft er, ob der Wert mit dem von TASTE übereinstimmt. Wenn ja, dann erfolgt keine Weitergabe an Bildschirm oder Programm. Im anderen Fall wird der Wert nach TASTE abgespeichert und in das X-Register transferiert. Der ASCII-Wert der Taste wird dann X-indiziert aus der ASCIITABLE geholt. Wenn die SHIFT-Taste gedrückt ist, wird er mit hex 80 OR-verknüpft und in den Tastaturpuffer gebracht. Ein Repeat wird somit einfach dadurch erreicht, daß man, bevor der IRQ abgearbeitet wird, ein hex FF nach TASTE abspeichert, wobei natürlich noch eine zeitliche Verzögerung erforderlich ist.

Im Programm wird nach dem Repeat die IRQ-Routine des Betriebssystems abgearbeitet. Dazu werden Rücksprungadresse, Prozessorstatus und die Register A, X und Y auf den Stack gebracht und danach über den direkten Sprung in die IRQ-Routine gesprungen. Ist diese abgearbeitet, dann erfolgt der Rücksprung über RTI ins PETARI-Programm. Dort wird dann noch getestet, ob eine der Funktionstasten gedrückt ist. Wenn nein, dann wird der IRQ verlassen, sonst wird die entsprechenden Routine abgearbeitet.

Die Stellung des Cursors im 25 Zeilen zu je 40 Zeichen umfassenden Bildschirm legen die Adressen CURADR, ZEILE und SPALTE (siehe Listing) fest. CURADR enthält die absolute Adresse des Anfangs der Zeile, in der sich der Cursor befindet. SPALTE stellt den Vektor von CURADR zur echten Cursoradresse dar. ZEILE wird intern dazu benutzt, um CURADR zu korrigieren, wenn der Cursor die Zeile wechselt. Setzt man ZEILE und SPALTE auf die gewünschten Werte und ruft dann die Systemroutine SETCURSOR auf, so läßt sich der Cursor auf einfache Weise an jede beliebige Stelle des Bildschirm positionieren. Was macht nun die Routine SETCURSOR? Der Rechner hat intern zwei Tabellen: Zum einen die Tabelle der LOW-Bytes der Zeilenanfänge bei ZEIADRL im ROM, sowie die Tabelle der Zeilenanfänge bei ZEILADRH im RAM. Wie bekannt, kann eine logische Zeile auf dem Bildschirm (nur!) bis zu zwei physikalische Zeilen lang sein. Wie merkt der Rechner nun, ob der Cursor gerade in einer Einfach- oder in einer Doppelzeile steht? Wie gesagt

hat der Rechner die Tabelle ZEIADRH im RAM. Steht der Cursor in einer Doppelzeile, so wird bei dem High-Byte der zweiten Zeile das höchstwertige Bit (hex 80) gelöscht. Zur Veranschaulichung folgt hier ein Hexauszug der Tabelle a) 1. und 2. Zeile = Einzelzeile und b) 1. und 2. Zeile = Doppelzeile:

a) 00E0 80 80 80 ... b) 00E0 80 00 80 ...

Die Routine SETCURSOR holt nun einfach die Adresse ZEILEN-indiziert aus den Tabellen und speichert sie nach CURADR. Holt man die zum Verschieben (zeilenweise mit Hilfe der Tabellen) benötigten Pointer aus diesen Tabellen, so muß man das HIGH-Byte mit hex 80 OR-verknüpfen, um auch auf jeden Fall die Adresse zu erhalten. Ferner ist es nach dem Verschieben notwendig, die Tabelle ZEILADRH zu korrigieren, indem man die gesetzten bzw. gelöschten hex 80-Bits ebenfalls verschiebt.

Das Programm liegt im oberen RAM-Bereich des Rechners und wird mit SYS 32060 aufgerufen. Es sichert dann selbst den benötigten Speicherbereich vor dem Zugriff durch das BASIC. Weiterhin wird der IRQ-Vektor auf das Programm gerichtet und schließlich ein NEW ausgeführt. Dadurch wird beim Aufruf von PETARI jedes im Speicher befindliche BASIC-Programm samt Variablen gelöscht. Sollte der Rechner eine Kassettenoperation durchführen, so stellt PETARI den IRQ-Vektor wieder auf seinen normalen Wert. Ein separates Ausschalten von PETARI entfällt damit. Danach muß man PETARI wieder mit SYS 32060 initialisieren.

```
PETARI
0003            0002 ;SYMBOLE
0003            0003 IRQADR =$90
0003            0004 TASTE  =$97
0003            0005 FROM   =$5E
0003            0006 TO     =$60
0003            0007 FROMPT =$62
0003            0008 TOPT   =$63
0003            0009 SHIFT  =$98
0003            0010 ANZAHL =$9E
0003            0011 RVSFLG =$9F
0003            0012 TAST   =$A6
0003            0013 CURFLG =$A7
0003            0014 CURBLK =$A8
0003            0015 CURFLA =$AA
0003            0016 CURADR =$C4
0003            0017 SPALTE =$C6
0003            0018 ZEILE  =$D8
0003            0019 ZEIFLG =$D5
0003            0020 INSERT =$DC
0003            0021 ZEIADH =$E0
0003            0022 REPDEC =$0F
0003            0023 TASTF  =$10
0003            0024 TOPRAM =$34
0003            0025 TASTUR =$026F
0003            0026 PORTA  =$E810
0003            0027 PORTB  =$E812
0003            0028 FFLAG  =$83FE
0003            0029 IRQ    =$E62E          ;$E455
0003            0030 IRQEND =$E6E4          ;$E600
0003            0031 SETCUR =$E25D          ;$E07F
0003            0032 ZEIADL =$E74B          ;$E65B
0003            0033 CURRE  =$CA40          ;$BB41
0003            0034 IOSCHL =$F8E6          ;$F92B
0003            0035 SETIRQ =$FC7B          ;$FCC0
0003            0036 NEW    =$C55D          ;$B5D3
```

```
0003            0037 PCR     =$E84C
0003            0038 ASCTAB  =$E6F7          ;$E60A
0003            0039
-------------------------------------------------------------
INITIALISIERT IRQ-POINTER
0003            0041         *=$7D3C
7D3C 78         0042 IRQINI  SEI
7D3D A951       0043         LDA #<IRQEIN
7D3F 8590       0044         STA IRQADR
7D41 A97D       0045         LDA #>IRQEIN
7D43 8591       0046         STA IRQADR+1
7D45 58         0047         CLI
7D46 A93C       0048         LDA #<IRQINI
7D48 8534       0049         STA TOPRAM
7D4A A97D       0050         LDA #>IRQINI
7D4C 8535       0051         STA TOPRAM+1
7D4E 4C5DC5     0052         JMP NEW
7D51            0053
-------------------------------------------------------------
REPEAT-ROUTINE
7D51 BD0501     0055 IRQEIN  LDA $0105,X     ;UEBERPRUEFT CASSETTENOPER
7D54 C9E6       0056         CMP #<IOSCHL    ;OB CBM IN I/O-ROUTINE
7D56 D00A       0057         BNE IOAB1
7D58 BD0601     0058         LDA $0106,X
7D5B C9F8       0059         CMP #>IOSCHL
7D5D D003       0060         BNE IOAB1
7D5F 207BFC     0061         JSR SETIRQ
7D62 A5A7       0062 IOAB1   LDA CURFLG      ;WENN CURSOR AUS,
7D64 D03E       0063         BNE SETSTK      ;REPEAT UEERSPRINGEN
7D66 A697       0064         LDX TASTE       ;TEST OB TASTE GEDRUECKT
7D68 E8         0065         INX
7D69 F031       0066         BEQ SETREP
7D6B C60F       0067         DEC REPDEC
7D6D D02B       0068         BNE REPAUS
7D6F A610       0069         LDX TASTF
7D71 E8         0070         INX
7D72 E000       0071         CPX #$00
7D74 D006       0072         BNE JMPO
7D76 A697       0073         LDX TASTE
7D78 E008       0074         CPX #$08
7D7A F020       0075         BEQ SETREP
7D7C A9FF       0076 JMPO    LDA #$FF
7D7E 8597       0077         STA TASTE
7D80 8510       0078         STA TASTF
7D82 A903       0079         LDA #$03
7D84 85A8       0080         STA CURBLK
7D86 A6D8       0081         LDX ZEILE
7D88 E017       0082         CPX #$17
7D8A 300C       0083         BMI STRPFL
7D8C A6A6       0084         LDX TAST
7D8E E042       0085         CPX #$42        ;=CD
7D90 D006       0086         BNE STRPFL
7D92 A698       0087         LDX SHIFT
7D94 D002       0088         BNE STRPFL
7D96 A910       0089         LDA #$10
7D98 850F       0090 STRPFL  STA REPDEC      ;WIEDERHOLZEIT = #$03
7D9A D008       0091 REPAUS  BNE SETSTK
7D9C A9FF       0092 SETREP  LDA #$FF
```

```
7D9E 8510     0093          STA TASTF
7DA0 A91F     0094          LDA #$1F
7DA2 D0F4     0095          BNE STRPFL
7DA4          0096
-------------------------------------------------------------
SETZT RTI-ADR. & REG. AUF STACK
7DA4 A97D     0098 SETSTK LDA #>FTASTE     ;SETZT RUECKSPRUNGADRESSE
7DA6 48       0099          PHA            ;AUF STACK
7DA7 A9B1     0100          LDA #<FTASTE
7DA9 48       0101          PHA
7DAA 08       0102          PHP
7DAB 48       0103          PHA
7DAC 48       0104          PHA
7DAD 48       0105          PHA
7DAE 4C2EE6   0108          JMP IRQ        ;FUEHRT IRQ AUS
7DB1          0109
-------------------------------------------------------------
TEST OB FUNKTIONSTASTE GEDRUECKT
7DB1          0111 ;WENN JA ENTHAELT Y DEN TASTENCODE
7DB1 EA       0112 FTASTE NOP
7DB2 A5A7     0113          LDA CURFLG     ;WENN CURSOR OFF,
7DB4 D078     0114          BNE ENDE       ;DANN ENDE DES IRQ
7DB6 A000     0115          LDY #$00
7DB8 A2F9     0116          LDX #$F9
7DBA 8E10E8   0117          STX PORTA
7DBD AE12E8   0118          LDX PORTB
7DC0 E0EF     0119          CPX #$EF       ;RUM/STOP-TASTE
7DC2 D002     0120          BNE JMP1
7DC4 A08A     0121          LDY #$8A
7DC6 E0FE     0122 JMP1     CPX #$FE       ;RVS-TASTE
7DC8 D050     0123          BNE JMP9
7DCA A2F0     0124          LDX #$F0
7DCC 8E10E8   0125          STX PORTA
7DCF AE12E8   0126          LDX PORTB
7DD2 E0BF     0127          CPX #$BF       ;CLR-TASTE
7DD4 D002     0128          BNE JMP2
7DD6 A082     0129          LDY #$82
7DD8 E07F     0130 JMP2     CPX #$7F       ;CR-TASTE
7DDA D002     0131          BNE JMP3
7DDC A083     0132          LDY #$83
7DDE EE10E8   0133 JMP3     INC PORTA
7DE1 AE12E8   0134          LDX PORTB
7DE4 E0BF     0135          CPX #$BF       ;CU-TASTE
7DE6 D002     0136          BNE JMP4
7DE8 A084     0137          LDY #$84
7DEA E07F     0138 JMP4     CPX #$7F       ;DEL-TASTE
7DEC D002     0139          BNE JMP5
7DEE A085     0140          LDY #$85
7DF0 EE10E8   0141 JMP5     INC PORTA
7DF3 AE12E8   0142          LDX PORTB
7DF6 E0BF     0143          CPX #$BF       ;7-TASTE
7DF8 D002     0144          BNE JMP6
7DFA A086     0145          LDY #$86
7DFC E07F     0146 JMP6     CPX #$7F       ;9-TASTE
7DFE D002     0147          BNE JMP7A
7E00 A087     0148          LDY #$87
7E02 E0DF     0149 JMP7A    CPX #$DF       ;^-TASTE
7E04 D002     0150          BNE JMP7
```

```
7E06 A081     0151          LDY #$81
7E08 EE10E8   0152 JMP7     INC PORTA
7E0B AE12E8   0153          LDX PORTB
7E0E E0BF     0154          CPX #$BF          ;8-TASTE
7E10 D002     0155          BNE JMP8
7E12 A088     0156          LDY #$88
7E14 E07F     0157 JMP8     CPX #$7F          ;/-TASTE
7E16 D002     0158          BNE JMP9
7E18 A089     0159          LDY #$89
7E1A C000     0160 JMP9     CPY #$00
7E1C D018     0161          BNE JMP10
7E1E A410     0162          LDY TASTF
7E20 A69E     0163          LDX ANZAHL
7E22 BD6F02   0164          LDA TASTUR,X
7E25 C912     0165          CMP #$12
7E27 D005     0166          BNE ENDE
7E29 ADFE83   0167          LDA FFLAG
7E2C F008     0168          BEQ JMP10
7E2E A901     0169 ENDE     LDA #$01
7E30 8DFE83   0170          STA FFLAG
7E33 4CE4E6   0171          JMP IRQEND
7E36 A59E     0172 JMP10    LDA ANZAHL        ;LOESCHT LETZTE
7E38 F002     0173          BEQ NODEC         ;IM TASTATURPUFFER
7E3A C69E     0174          DEC ANZAHL
7E3C A900     0175 NODEC    LDA #$00          ;LOESCHT RVS-FLAG
7E3E 8DFE83   0176          STA FFLAG
7E41 859F     0177          STA RVSFLG
7E43 C410     0178          CPY TASTF
7E45 D003     0179          BNE EXEC
7E47 4CE4E6   0180          JMP IRQEND
7E4A 8410     0181 EXEC     STY TASTF
7E4C 98       0182          TYA
7E4D 48       0183 DER      PHA               ;CURSOR OFF, UM ZU
7E4E 46AA     0184          LSR CURFLA        ;VERHINDERN, DASS UNER-
7E50 3006     0185          BMI BLNKOF        ;WUENSCHTE RVS-ZEICHEN
7E52 A4C6     0186          LDY SPALTE        ;AN DER CURSORSTELLE
7E54 A5A9     0187          LDA $A9           ;ERSCHEINEN
7E56 91C4     0188          STA (CURADR),Y
7E58 68       0189 BLNKOF   PLA
7E59 A000     0190          LDY #$00
7E5B 84DC     0191          STY INSERT
7E5D          0192
-------------------------------------------------------------
DELETE 1 ZEICHEN RECHTS VOM CURSOR
7E5D C98A     0194          CMP #$8A
7E5F D01A     0195          BNE TAB
7E61 A4C6     0196          LDY SPALTE        ;WENN CURSOR AM ENDE
7E63 C4D5     0197          CPY ZEIFLG        ;DER ZEILE, DANN ENDE
7E65 F053     0198          BEQ ENDE1
7E67 C8       0199          INY
7E68 C4D5     0200 MOVE1    CPY ZEIFLG
7E6A F009     0201          BEQ FILS
7E6C C8       0202          INY
7E6D B1C4     0203          LDA (CURADR),Y
7E6F 88       0204          DEY               ;VERSCH 1 ZCHN. 1X LINKS
7E70 91C4     0205          STA (CURADR),Y
7E72 C8       0206          INY
7E73 D0F3     0207          BNE MOVE1
```

```
7E75 A920     0208 FILS   LDA #$20         ;ZCHN. RECHTS AUSSEN
7E77 91C4     0209        STA (CURADR),Y   ;WIRD ZU SPACE
7E79 D03F     0210        BNE ENDE1
7E7B          0211
-------------------------------------------------------------
TABULATOR: JEWEILS 15 * CURSOR RECHTS
7E7B C983     0213 TAB    CMP #$83
7E7D D00A     0214        BNE CLR
7E7F A20F     0215        LDX #$0F         ;ANZAHL CURS-RE
7E81 2040CA   0216 TAB1   JSR CURRE
7E84 CA       0217        DEX
7E85 D0FA     0218        BNE TAB1
7E87 F031     0219        BEQ ENDE1
7E89          0220
-------------------------------------------------------------
LOESCHT REST DES BILDSCHIRMS
7E89 C986     0222 CLR    CMP #$86
7E8B D030     0223        BNE DEZ
7E8D A5C4     0224        LDA CURADR       ;INITIALISIERT POINTER
7E8F 855E     0225        STA FROM
7E91 A5C5     0226        LDA CURADR+$01
7E93 855F     0227        STA FROM+1
7E95 A000     0228        LDY #$00         ;FUELLT REST DES SCHIRMS
7E97 A920     0229        LDA #$20         ;MIT SPACE AUF
7E99 915E     0230 CLR0   STA (FROM),Y
7E9B E65E     0231        INC FROM
7E9D D002     0232        BNE CLR1
7E9F E65F     0233        INC FROM+1
7EA1 A65F     0234 CLR1   LDX FROM+1
7EA3 E083     0235        CPX #$83
7EA5 D0F2     0236        BNE CLR0
7EA7 A65E     0237        LDX FROM
7EA9 E0E8     0238        CPX #$E8
7EAB D0EC     0239        BNE CLR0
7EAD A6D8     0240        LDX ZEILE        ;KORRIGIERT MSB'S DER
7EAF B5E0     0241 CLR2   LDA ZEIADH,X     ;ZEILENANFAENGE
7EB1 0980     0242        ORA #$80
7EB3 95E0     0243        STA ZEIADH,X
7EB5 E8       0244        INX
7EB6 E017     0245        CPX #$17
7EB8 90F5     0246        BCC CLR2
7EBA 4C2E7E   0247 ENDE1  JMP ENDE
7EBD          0248
-------------------------------------------------------------
LOESCHT 1 ZEILE UND SCHIEBT RESTLICHE NACH
7EBD C984     0250 DEZ    CMP #$84
7EBF D063     0251        BNE DRZ
7EC1 A6D8     0252        LDX ZEILE
7EC3 E018     0253        CPX #$18
7EC5 F0F3     0254        BEQ ENDE1
7EC7 E018     0255 DEZ0   CPX #$18
7EC9 F024     0256        BEQ DEZ2
7ECB B5E0     0257        LDA ZEIADH,X     ;INITIALISIERT POINTER
7ECD 0980     0258        ORA #$80
7ECF 8561     0259        STA TO+1
7ED1 BD48E7   0260        LDA ZEIADL,X
7ED4 8560     0261        STA TO
7ED6 B5E1     0262        LDA ZEIADH+1,X
```

```
7ED8 0980     0263          ORA ##80
7EDA B55F     0264          STA FROM+1
7EDC BD49E7   0265          LDA ZEIADL+1,X
7EDF 855E     0266          STA FROM
7EE1 A000     0267          LDY ##00          ;VERSCH. 1 ZEILE NACH OBEN
7EE3 B15E     0268 DEZ1     LDA (FROM),Y
7EE5 9160     0269          STA (TO),Y
7EE7 C8       0270          INY
7EE8 C028     0271          CPY ##28
7EEA D0F7     0272          BNE DEZ1
7EEC E8       0273          INX
7EED D0D8     0274          BNE DEZ0
7EEF A000     0275 DEZ2     LDY ##00          ;FUELLT LETZTE ZEILE DES
7EF1 A920     0276          LDA ##20          ;SCHIRMS MIT SPACE AUF
7EF3 915E     0277 DEZ3     STA (FROM),Y
7EF5 C8       0278          INY
7EF6 C028     0279          CPY ##28
7EF8 D0F9     0280          BNE DEZ3
7EFA A6D8     0281          LDX ZEILE         ;KORRIGIERT MSB'S DER
7EFC E017     0282 DEZ6     CPX ##17          ;ZEILENANFAENGE
7EFE F00F     0283          BEQ DEZ4
7F00 B5E0     0284          LDA ZEIADH,X
7F02 0980     0285          ORA ##80
7F04 B4E1     0286          LDY ZEIADH+1,X
7F06 3002     0287          BMI DEZ5
7F08 0980     0288          ORA ##80
7F0A 95E0     0289 DEZ5     STA ZEIADH,X
7F0C E8       0290          INX
7F0D D0ED     0291          BNE DEZ6
7F0F B5E1     0292 DEZ4     LDA ZEIADH+1,X
7F11 0980     0293          ORA ##80
7F13 A6D8     0294          LDX ZEILE
7F15 E018     0295          CPX ##18
7F17 F007     0296          BEQ DEZ7
7F19 E8       0297          INX
7F1A B5E0     0298          LDA ZEIADH,X
7F1C 3002     0299          BMI DEZ7
7F1E A04F     0300          LDY ##4F
7F20 84D5     0301 DEZ7     STY ZEIFLG
7F22 D096     0302          BNE ENDE1
7F24          0303
-------------------------------------------------------------

7F24 C989     0305 DRZ      CMP ##89
7F26 D00D     0306          BNE SWI
7F28 A4C6     0307          LDY SPALTE
7F2A A920     0308          LDA ##20
7F2C 91C4     0309 DRZ1     STA (CURADR),Y
7F2E C4D5     0310          CPY ZEIFLG
7F30 F00F     0311          BEQ ENDE2
7F32 C8       0312          INY
7F33 D0F7     0313          BNE DRZ1
7F35          0314
-------------------------------------------------------------
UMSCHALTEN ZW. KLEIN/GROSS U. GROSS/GRAFIK
7F35 C981     0316 SWI      CMP ##81
7F37 D00B     0317          BNE CLA
7F39 AD4CE8   0318          LDA PCR           ;PERIPH. CONTROL REGISTER
```

```
7F3C 4902     0319          EOR #$02        ;EXOR #$02
7F3E 8D4CE8   0320          STA PCR
7F41 4C2E7E   0321 ENDE2    JMP ENDE
7F44          0322
-------------------------------------------------------------
LOESCHT BEGINN DES BILDSCHIRMS
7F44 C982     0324 CLA      CMP #$82
7F46 D031     0325          BNE DBZ
7F48 A900     0326          LDA #$00        ;INITIALISIERT POINTER
7F4A 8560     0327          STA TO
7F4C A980     0328          LDA #$80
7F4E 8561     0329          STA TO+1
7F50 A561     0330 CLA0     LDA TO+1
7F52 C5C5     0331          CMP CURADR+1
7F54 D006     0332          BNE CLA1
7F56 A560     0333          LDA TO
7F58 C5C4     0334          CMP CURADR
7F5A F00E     0335          BEQ CLA2
7F5C A000     0336 CLA1     LDY #$00        ;FUELLT BEGINN DES SCHIRMS
7F5E A920     0337          LDA #$20        ;MIT SPACE AUF
7F60 9160     0338          STA (TO),Y
7F62 E660     0339          INC TO
7F64 D0EA     0340          BNE CLA0
7F66 E661     0341          INC TO+1
7F68 D0E6     0342          BNE CLA0
7F6A A200     0343 CLA2     LDX #$00
7F6C E4D8     0344 CLA3     CPX ZEILE
7F6E F0D1     0345          BEQ ENDE2
7F70 B5E0     0346          LDA ZEIADH,X
7F72 0980     0347          ORA #$80
7F74 95E0     0348          STA ZEIADH,X
7F76 E8       0349          INX
7F77 D0F3     0350          BNE CLA3
7F79          0351
7F79          0352
-------------------------------------------------------------
LOESCHT BEGINN DER ZEILE
7F79 C985     0354 DBZ      CMP #$85
7F7B D00D     0355          BNE INZ
7F7D A000     0356          LDY #$00
7F7F A920     0357          LDA #$20
7F81 C4C6     0358 DBZ0     CPY SPALTE
7F83 F0BC     0359          BEQ ENDE2
7F85 91C4     0360          STA (CURADR),Y
7F87 C8       0361          INY
7F88 D0F7     0362          BNE DBZ0
7F8A          0363
-------------------------------------------------------------
INSER 1 ZEILE, SCROLLING DER RESTLICHEN ZEILEN
7F8A C988     0365 INZ      CMP #$88
7F8C D05B     0366          BNE END
7F8E A5D8     0367          LDA ZEILE
7F90 C918     0368          CMP #$18
7F92 F0AD     0369          BEQ ENDE2
7F94 A927     0370          LDA #$27
7F96 85D5     0371          STA ZEIFLG
7F98 A218     0372          LDX #$18        ;SETZT VERSCHIEBE-POINTER
7F9A B5E0     0373 INZ0     LDA ZEIADH,X
```

```
7F9C 0980     0374          ORA ##80
7F9E 8561     0375          STA TO+1
7FA0 BD48E7   0376          LDA ZEIADL,X
7FA3 8560     0377          STA TO
7FA5 CA       0378          DEX
7FA6 B5E0     0379          LDA ZEIADH,X
7FA8 0980     0380          ORA ##80
7FAA 855F     0381          STA FROM+1
7FAC BD48E7   0382          LDA ZEIADL,X
7FAF 855E     0383          STA FROM
7FB1 A000     0384          LDY ##00          ;VERSCHIEBT EINE ZEILE
7FB3 B15E     0385 INZ1     LDA (FROM),Y
7FB5 9160     0386          STA (TO),Y
7FB7 C8       0387          INY
7FB8 C028     0388          CPY ##28
7FBA D0F7     0389          BNE INZ1
7FBC E4D8     0390          CPX ZEILE
7FBE D0DA     0391          BNE INZ0
7FC0 A000     0392          LDY ##00          ;FUELLT ZEILE MIT
7FC2 A920     0393 INZ2     LDA ##20          ;SPACE AUF
7FC4 915E     0394          STA (FROM),Y
7FC6 C8       0395          INY
7FC7 C028     0396          CPY ##28
7FC9 D0F7     0397          BNE INZ2
7FCB A218     0398          LDX ##18          ;KORRIGIERT MSB'S DER
7FCD B5E0     0399 INZ3     LDA ZEIADH,X      ;ZEILENANFAENGE
7FCF 0980     0400          ORA ##80
7FD1 CA       0401          DEX
7FD2 B4E0     0402          LDY ZEIADH,X
7FD4 3002     0403          BMI INZ4
7FD6 297F     0404          AND ##7F
7FD8 E8       0405 INZ4     INX
7FD9 95E0     0406          STA ZEIADH,X
7FDB CA       0407          DEX
7FDC E4D8     0408          CPX ZEILE
7FDE D0ED     0409          BNE INZ3
7FE0 B5E0     0410          LDA ZEIADH,X      ;INSERT ZEILE,
7FE2 0980     0411          ORA ##80          ;EINFACHZEILE
7FE4 95E0     0412          STA ZEIADH,X
7FE6 4C417F   0413          JMP ENDE2
7FE9          0414
-------------------------------------------------------------
SETZT CURSOR AUF ANFANG DER LETZTEN ZEILE
7FE9 C987     0416 END      CMP ##87
7FEB D00B     0417          BNE ENDE3
7FED A000     0418          LDY ##00          ;SPALTE
7FEF A218     0419          LDX ##18          ;ZEILE
7FF1 84C6     0420          STY SPALTE
7FF3 86D8     0421          STX ZEILE
7FF5 205DE2   0422          JSR SETCUR        ;BERECHNET CURSORPOSITION
7FF8 4C417F   0423 ENDE3    JMP ENDE2
7FFB          0424          .END
```

#

Dipl.-Ing. (FH) Frank H. Koether, 5000 Köln 91

Timesharing

Überlegungen zu einem Time Sharing System für Personal Computer

1. Was ist Time Sharing?

Nach einem Artikel von C.C. Foster in A. Ralston (Hrsg.) "Encyclopedia Of Computer Science", New York 1976, 1. Auflage, ist Time Sharing "eine Methode, einen Computer so zu betreiben, daß zwei oder mehr Benutzer in der Lage sind, quasi simultan ihre Probleme in diesen Computer einzugeben und die Antworten dazu zu erhalten".

Zum Time Sharing muß der Verarbeitungsprozeß im Computer 'irgendwann', noch bevor er abgeschlossen ist, unterbrochen werden, um die Verarbeitung für einen oder mehrere Benutzer aufzunehmen. Der Computer verhält sich dabei für den Benutzer wie ein ausschließlich dem Benutzer zur Verfügung stehendes System. Folglich muß der Verarbeitungsprozeß für jedes System (für jeden Benutzer) in sehr kleine Teilabläufe unterteilbar sein. Jedes System wird dabei einmal innerhalb eines Zyklus angesprungen. Zykluszeit nennt man die Zeit, die erforderlich ist, jedes System einmal zu bedienen (Cycle Time).

Die Zeit, die innerhalb eines Zyklus für jedes System zur Verfügung steht, heißt Zeitscheibe (Time Slice oder Time Slot). Jedes Benutzerproblem wird im Grunde 'scheibchenweise' bearbeitet. Deshalb nennt man diese Betriebsart auch 'Time Slicing' oder 'Time Multiplexing' (=Zeit-Multiplex-Betrieb). Abbildung 1 zeigt den prinzipiellen Aufbau eines solchen Zeitmultiplex-Computers.

2. Was muß ein Time-Sharing-Betriebssystem haben?

2.1 Initialisierung

Der verfügbare Speicherplatz sollte unter die Benutzer aufgeteilt werden (Partitioning) für:

Programm
Daten
Eingabe
Ausgabe

Die Prioritäten der einzelnen Benutzer müssen hier festgelegt werden.

2.2 Zyklussteuerung
Damit wird die zyklische Bearbeitung für die Benutzersysteme gesteuert.

2.3 Steuerungsmodul
Die Ablaufsteuerung des Benutzerprogramms muß vorhanden sein.

2.4 Ablaufsteuerung
Für die Zeitscheibe müssen jedem Benutzerprogrammteil die erforderlichen Parameter zugewiesen und wieder gespeichert bzw. ausgegeben werden. Entsprechend muß auch ein Benutzer-Systemstatus geführt werden.

Vereinfachtes Flußdiagramm: Siehe Abbildung 2.

3. Ein Simulationsmodell zur Implementierung auf einen Personal Computer

3.1 Selbstauferlegte Randbedingungen

Es steht nur 1 Personal Computer zur Verfügung. Das heißt, daß nur 1 Eingabe- und vielleicht 2 Ausgabemedien sinnvoll für die Simulation nutzbar sind: Tastatur und Bildschirm/Drucker.

Folglich soll die Tastatur 2 Benutzer simulieren. Der Bildschirm oder Drucker soll als Ausgabemedium dienen und zugleich den inneren Ablauf des Systems zeigen. Um die Simulation in eine praktische Nutzung zu überführen, soll es hinreichend sein, Ein- und Ausgaberoutinen für zusätzliche Schnittstellen zu modifizieren. Damit diese Simulation auf möglichst viele Personal Computer adaptiert werden kann, wurde als Programmiersprache BASIC verwendet. Dies machte es allerdings auch erforderlich, auf echtes 'time slicing' zu verzichten.

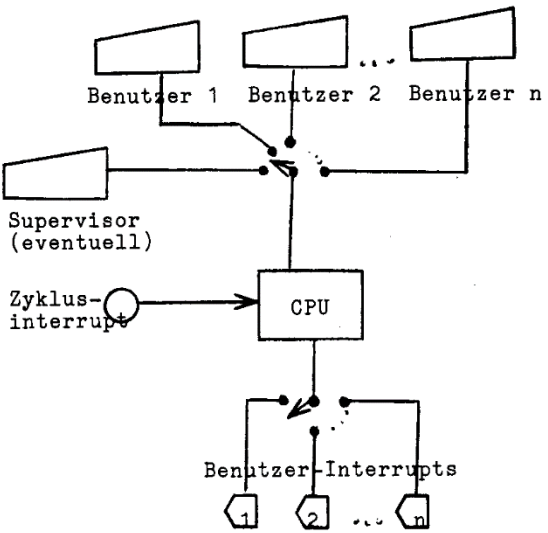


Abb. 1:
Vereinfachtes Time-Sharing-System

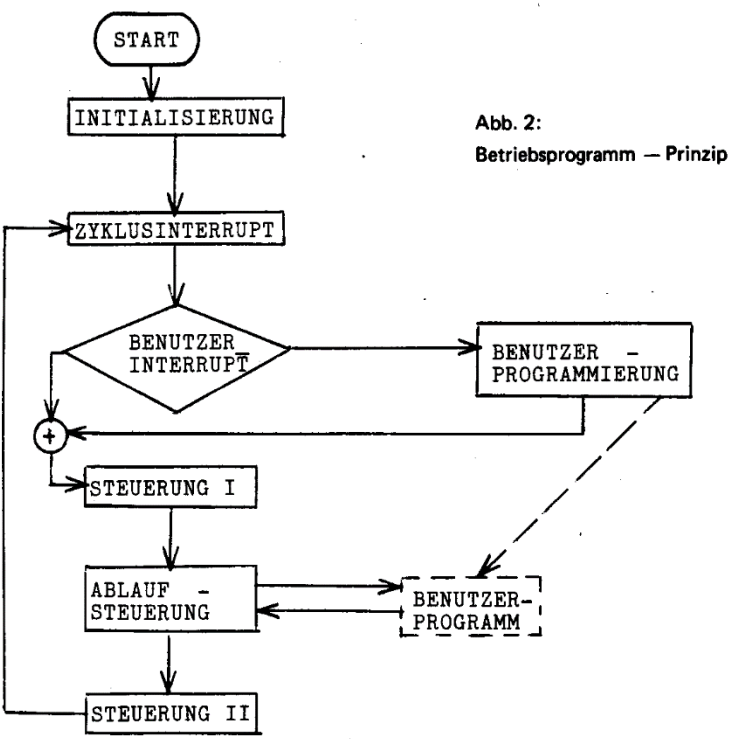


Abb. 2:
Betriebsprogramm — Prinzip

Als Anwenderprogramm wurden einfache BASIC-Subroutinen verwendet. Dieses ist durch den implementierenden Programmierer schnell änderbar.

3.2 Hinweis

Das ursprüngliche Programm wurde auf einem Radio Shack Color Computer mit Extended Color Basic programmiert und ausgetestet. In den hier wiedergegebenen Listings wurde versucht, Extended Color Basic-spezifische Befehle zu vermeiden.

3.3 Der Programmablauf

3.3.1 Initialisierung

In der Initialisierung wird der Umfang des Gesamtsystems festgelegt. Jeder Schritt des Anwenderprogramms soll als Quatupel dargestellt sein:

Anwenderbefehl = (Befehl, Operand 1, Operand 2, Status)

Zusätzlich gibt es noch einen Benutzer-System-Zwischenspeicher. Dieser Speicher dient der Parameterübergabe zwischen den Zwischenschritten der Ausführung. Abschließend wurde versucht, zwei unterschiedliche Ausgabeformate zu implementieren, für den AIM 65 und für andere BASIC-Computer (hier z.B. Color Computer).

3.3.2 Interuptgenerator/Zyklussteuerung (Cycle Control)

Aus Gründen der Vereinfachung wurde auf einen echten Interruptgenerator verzichtet Vielmehr wurde einfach eine automatische Benutzersystemumschaltung programmiert. In die Systemumschaltung integriert wurde auch die 'Benutzerbedienung': Über Tastaturinterrupt kann das jeweilige System angesprochen und gegebenenfalss programmiert werden.

Die Zyklussteuerung übergibt an das Steuerungsmodul I.

3.3.3 Steuerungsmodul I (Allocation)

Dieser Programmteil überprüft den Status des jeweiligen Benutzersystems. Entweder ist das System nicht programmiert = 'IDLE' oder der im Benutzerprogramm zur Ausführung vorgesehene Schritt wird der Ablaufsteuerung zugewiesen.

3.3.4 Ablaufsteuerung (Execution Control)

Die Ablaufsteuerung übernimmt aus dem Speicher die für die Ausführung erforderlichen Parameter. Dann übergibt sie an die 'Befehlsausführung' (Subroutine). Der Status wird entsprechend geführt.

3.3.5 Steuerungsmodul II (Deallocation)

Dieses Modul sorgt für die Übergabe des Systemparameters T an den Systemzwischenspeicher. Zusätzlich müßte dieses Modul die Speicherung von Zwischenwerten der Befehlsausführung kontrollieren. Hier ist es darauf beschränkt, diese Werte einer Ausgaberoutine zuzuweisen. Dann übergibt dieses Modul an die Zyklussteuerung.

4. Anmerkungen zum systemtheoretischen Hintergrund

4.1 Zyklussteuerung

Ein derartiges Modul muß normalerweise als Interruptservice programmiert werden. Die Interruptfrequenz ist so festzulegen, daß der längste Schritt in der Befehlsausführung zusammen mit dem Speichern von Zwischenwerten und Parameternn noch zuverlässig ausgeführt werden kann.

4.2 Anwenderprogrammierung

Dieser Problemkreis berührt zu einem erheblichen Teil das Gebiet der Entwicklung von künstlichen Sprachen. Hier wurde eine sehr primitive Form gewählt:

1 numerischer Kode bezeichnet Verknüpfungen
auf jede Verknüpfung müssen 2 Operanden folgen

Die Benutzung des Ergebnisses der vorhergehenden Verknüpfung für die folgende ist hier aus Gründen der Einfachheit nicht zugelassen.

Interessant ist es jetzt, hier eine Sprache für die Anwenderprogrammierung einzubringen, die

Verkettungen von Operationen (Verknüpfungen) erlaubt. Der Grundstock ist hier bereits durch die Ausführungsliste gelegt. Als Folge davon wird das Steuerungsmodul I und die Ablaufsteuerung erheblich komplizierter werden müssen. Beide Module sind zu einem vollwertigen Interpretor hin zu entwickeln.

Abschließend sei noch vermerkt, daß hier das Quatupel des Anwenderbefehls zur einfacheren Behandlung im Programmablauf in seine Spaltenvektoren aufgelöst wurde.

5. Schlußbemerkung

5.1 Anregung für den Benutzer

Schreiben Sie die Ausgaberoutine doch so um, daß jedes System ein anderes Ausgabemedium benutzt! Modifizieren Sie die Routine für die Benutzerprogrammierung in der Weise, daß diese auch 'scheibchenweise' abgearbeitet wird. Nun sollten Sie versuchen, die Zyklussteuerung als echte Interrupt-Serviceroutine aufzubauen.

5.2 'Feedback'

Wenn Interesse seitens der Leserschaft an weiteren 'Programmbeispielen' oder Simulationen dieser Art besteht, so bittet der Autor um Zuschrift. Mögliche Themen könnten sein: Simulation einer Mehrprozessoranlage mit einem Personal Computer; Simulation des Betriebsverhaltens von plattenspeicherartigen Medien (Flopp/Hard Disks); Hard- und Softwaretechniken zur Speichererweiterung in Personal Computern. Bei der Komplexität dieser Themen könnte eine Art lose Fortsetzungsserie in Betracht gezogen werden.

```
100 REM ============
110 REM TIME SHARING
120 REM DEMOPROGRAMM
130 REM COPYRIGHT BY
140 REM DIPL-ING(FH)
150 REM FRANK KOETHER
160 REM LUDW.-QUID-
170 REM DE-PLATZ 4
180 REM 5000 KOELN
190 REM     91
200 REM (0221)8902554
210 REM (C) 1982
220 REM ============
1020 REM INITIALISIERUNG
1030 DIM T(1,5) : REM BENUTZERPROGRAMM (MAXIMAL 6 SCHRITTE)
1040 DIM P1(1,5) : REM EINGANGSPARAMETERLISTE 1
1050 DIM P2(1,5) : REM EINGANGSPARAMETERLISTE 2
1060 DIM S(1,5) : REM BENUTZER-PROGRAMM-STATUS-LISTE
1070 DIM B(1) : REM BENUTZER-SYSTEM-ZWISCHENSPEICHER
1080 DIM M$(5),B$(3) : REM TEXT
1090 M$(0)="AIM-65" : M$(2)="IDLE" : M$(3)="BUSY"
1100 B$(1)="S(T)" : B$(2)="H(T)" : B$(3)="V(T)"
1110 PRINT"TIME SHARING DEMO :"
1120 PRINT"====================="
1180 GOTO 1220
1190 REM -----------
1200 IF M=0 THEN M=1 : GOTO 1220 : REM ZYKLUSINTERRUPT
1210 IF M=1 THEN M=0
1220 GOSUB 2280 : REM BENUTZERINTERRUPT
1230 IF I$<>RIGHT$(STR$(M),1) THEN 1280
1240 SC=0 : FOR I=0 TO 5 : SC=SC+S(M,I) : NEXT I
1250 IF SC<>0 THEN B$=M$(3) : GOSUB 2070 : GOTO 1280
1260 GOSUB 1790 : GOTO 1200
1270 REM -----------
```

```
1280 S=0 : REM STEUERUNGSMODUL 1
1290 IF S(M,S)=0 THEN S=S+1 : GOTO 1310
1300 GOTO 1360 : REM ZUR ABLAUFSTEUERUNG
1310 IF S>5 THEN 1330
1320 GOTO 1290
1330 B$=M$(2) : GOSUB 2070
1340 GOTO 1200
1350 REM -----------
1360 REM ABLAUFSTEUERUNG
1370 V=P1(M,S)
1380 A=P2(M,S)
1390 T=B(M)
1400 TT=2*V*SIN(A/57.2958)/9.81
1410 IF TT<T THEN T=TT : S(M,S)=0
1420 ON T(M,S) GOSUB 1480,1520,1560 : REM ZUWEISUNG
1430 REM -----------
1440 IF S(M,S)<>0 THEN B(M)=T+.1 : REM STEUERUNGSMODUL 2
1450 IF S(M,S)=0 THEN B(M)=0
1460 GOSUB 2100 : GOTO 1200
1470 REM ------------
1480 REM BERECHNUNG DES WEGES NACH DER ZEIT T
1490 O=V*T*COS(A/57.2958)
1500 RETURN
1510 REM -----------
1520 REM BERECHNUNG DER HOEHE NACH DER ZEIT T
1530 O=V*T*SIN(A/57.2958)-9.81*T*T/2
1540 RETURN
1550 REM -----------
1560 REM BERECHNUNG DER STEIGGESCHWINDI1A
1570 O=V*SIN(A/57.2958)-9.81*T
1580 RETURN
1590 REM ------------
1600 DATA "BERECHNUNGEN ZUM", "SCHIEFEN WURF :"
1610 DATA "DIE PROGRAMMLISTE =", "ANWENDERPROGRAMM"
1620 DATA "KANN 6 SCHRITTE AUF-", "NEHMEN (0-5)."
1630 DATA "ZUSAETZLICH SIND JE", "SCHRITT 2 "
1640 DATA "EINGANGSPARAMETER", "EINZUGEBEN :"
1650 DATA "*STARTGESCHWINDIG-", "      KEIT V IN M/S"
1660 DATA "*ABSCHUSZWINKEL ", "      A IN GRAD ."
1670 DATA "IN JEDEM ZYKLUS WIRD", "1 WERT FUER DEN GE-"
1680 DATA "SAMTEN BAHNVERLAUF", "BERECHNET UND AUS-"
1690 DATA "GEGEBEN. FOLGENDE", "WERTE = SCHRITTBEFEH"
1700 DATA "-LE SIND MOEGLICH :", " "
1710 DATA "1 - WEG, S(T)", " "
1720 DATA "2 - HOEHE, H(T)", " "
1730 DATA "3 - STEIGGESCHWINDIG", "      -KEIT, V(T)"
1735 DATA "'/' - ENDE DER BENU-", "TZERPROGRAMMIERUNG."
1740 DATA " ", " "
1750 DATA "GRENZWERTE SIND : ", "STARTGESCHWINDIGKEIT"
1760 DATA "     : 0<=V<=800 M/S",   "ABSCHUSZWINKEL : "
1770 DATA "     : 0<=A<=90 GRAD", " ", #
1780 REM -----------
1790 RESTORE : REM BENUTZERPROGRAMMIERUNG
1800 REM BENUTZERANWEISUNG
1810 IF N=1 THEN 1850
1820 READ M$(4) : IF M$(4)="#" THEN 1930
1830 PRINT M$(4) : GOTO 1820
1850 PRINT"BEFEHLSLISTE  :"
```

```
1860 PRINT"--------------------"
1870 FOR I=1 TO 3 : PRINTI;". ";B$(I) : NEXT I
1910 REM -----------
1920 REM ERGAENZUNG DES BENUTZERPROGRAMMS
1930 N=1 : FOR I=0 TO 5
1940 INPUT"BEFEHL : ";T$ : IF T$="/" THEN 2050
1950 IF T$<"1" OR T$>"3" THEN PRINT"FALSCHER BEFEHL !" : GO
1960 INPUT"STARTGESCHW. :";V
1970 IF V>800 OR V<0 THEN PRINT"FEHLER : 800<V<0 !" : GOTO
1980 INPUT"ABSCHUSZWINKEL :";A
1990 IF A<0 OR A>90 THEN PRINT"FEHLER : 90>A>0 !" : GOTO 19
2000 T(M,I)=VAL(T$)
2010 P1(M,I)=V
2020 P2(M,I)=A
2030 S(M,I)=1
2040 NEXT I
2050 RETURN
2060 REM -----------
2070 PRINT"SYSTEM "M" "B$ : REM (STATUS)AUSGABE
2080 RETURN
2090 REM -----------
2100 REM PARAMETERAUSGABE
2110 T1$=STR$(INT(T)) : T2$=STR$(T-INT(T))
2120 O1$=STR$(INT(O)) : O2$=STR$(O-INT(O))
2130 IF VAL(T2$)<.01 THEN T2$=" .00"
2140 IF VAL(O2$)<.01 THEN O2$=" .00"
2150 IF LEN(T2$)<4 THEN T2$=T2$+"0"
2160 IF LEN(O2$)<4 THEN O2$=O2$+"0"
2170 T2$=RIGHT$(LEFT$(T2$,4),3)
2180 O2$=RIGHT$(LEFT$(O2$,4),3)
2190 T$=RIGHT$("      "+T1$+T2$,6)
2200 O$=RIGHT$("      "+O1$+O2$,8)
2210 B$="T= "+T$
2220 GOSUB 2070
2240 PRINTB$(T(M,S))";" = ";O$
2260 RETURN
2270 REM -----------
2280 REM BENUTZERINTERRUPT
2290 GET I$
2300 RETURN
2320 REM -----------
2400 END
```

Protokoll eines Probelaufes vorstehenden Programmes

```
<6>
RUN100
TIME SHARING DEMO :
====================
SYSTEM  0  IDLE
SYSTEM  1  IDLE
SYSTEM  0  IDLE
SYSTEM  1  IDLE
SYSTEM  0  IDLE
SYSTEM  1  IDLE
SYSTEM  0  IDLE
SYSTEM  1  IDLE
SYSTEM  0  IDLE
SYSTEM  1  IDLE
```

```
BERECHNUNGEN ZUM
SCHIEFEN WURF :
DIE PROGRAMMLISTE =
ANWENDERPROGRAMM
KANN 6 SCHRITTE AUF-
NEHMEN (0-5).
ZUSAETZLICH SIND JE
SCHRITT 2
EINGANGSPARAMETER
EINZUGEBEN :
*STARTGESCHWINDIG-
     KEIT V IN M/S
*ABSCHUSZWINKEL
     A IN GRAD .
```

```
IN JEDEM ZYKLUS WIRD
1 WERT FUER DEN GE-
SAMTEN BAHNVERLAUF
BERECHNET UND AUS-
GEGEBEN. FOLGENDE
WERTE = SCHRITTBEFEH
-LE SIND MOEGLICH :

1 - WEG, S(T)

2 - HOEHE, H(T)

3 - STEIGGESCHWINDIG
     -KEIT, V(T)
```

```
'/' - ENDE DER BENU-
TZERPROGRAMMIERUNG.


GRENZWERTE SIND :
STARTGESCHWINDIGKEIT
    : 0<=V<=800 M/S
ABSCHUSZWINKEL :
    : 0<=A<=90 GRAD

BEFEHL : ? 1
STARTGESCHW. :? 600
ABSCHUSZWINKEL :? 61
BEFEHL : ? 1
STARTGESCHW. :? 610
ABSCHUSZWINKEL :? 61
BEFEHL : ? /
SYSTEM  0  IDLE
SYSTEM  1  T=    0.00
S(T) =     0.00
BEFEHLSLISTE  :
--------------------
 1 . S(T)
 2 . H(T)
 3 . V(T)
```

```
BEFEHL : ? 3
STARTGESCHW. :? 400
ABSCHUSZWINKEL :? 77
BEFEHL : ? 3
STARTGESCHW. :? 410
ABSCHUSZWINKEL :? 77
BEFEHL : ? /
SYSTEM  1  T=    0.10
S(T) =     29.08
SYSTEM  0  T=    0.00
V(T) =     389.74
SYSTEM  1  T=    0.20
S(T) =     58.17
SYSTEM  0  T=    0.10
V(T) =     388.76
SYSTEM  1  T=    0.30
S(T) =     87.26
SYSTEM  0  T=    0.20
V(T) =     387.78
SYSTEM  1  T=    0.40
S(T) =     116.35
SYSTEM  0  T=    0.30
V(T) =     386.80
SYSTEM  1  T=    0.50
S(T) =     145.44
```

```
SYSTEM  0  T=    0.40
V(T) =     385.82
SYSTEM  1  T=    0.60
S(T) =     174.53
SYSTEM  0  T=    0.50
V(T) =     384.84
SYSTEM  1  T=    0.70
S(T) =     203.62
SYSTEM  0  T=    0.60
V(T) =     383.86
SYSTEM  1  T=    0.80
S(T) =     232.70
SYSTEM  0  T=    0.70
V(T) =     382.88
SYSTEM  1  T=    0.90
S(T) =     261.79
SYSTEM  0  T=    0.80
V(T) =     381.89
SYSTEM  1  T=    1.00
S(T) =     290.88
SYSTEM  0  T=    0.90
V(T) =     380.91
SYSTEM  1  T=    1.10
S(T) =     319.97
SYSTEM  0  T=    1.00
V(T) =     379.93
```

#

Hans-Georg Lange, 1000 Berlin 30

Compactor-Review (CBM)

Anpassung des BASIC-Compactors für CBM 3032 und 8032

Das in (1) wiedergegebene Programm zur Verdichtung von BASIC-Programmen wurde für die CBM-Rechner angepaßt. Das nachfolgende Listing ist für den 3032 geschrieben, in Klammern sind jedoch auch die Einsprungspunkte für Rechner mit BASIC 4 angegeben, so daß eine entsprechende Assemblierung nicht schwer fällt.

Es mußten nicht nur die 'tokens' des BASIC angepaßt werden, sondern natürlich auch die Pointer und die Adressen der ROM-residenten Routinen. Gegenüber dem in (1) für den AIM 65 abgedruckten Programm konnten einige Beschränkungen aufgehoben und Verbesserungen angebracht werden: Dort konnten REMARK-Zeilen kein Sprungziel sein, weil sie durch den Compactor entfernt werden. Hier nun sucht SRCHLN die durch LINGET gefundene Zeilennummer im BASIC-Text. Wird diese nicht gefunden, so setzt CONVRT stattdessen die Nummer der nächsten Zeile ein. Es gilt daher nur noch die Einschränkung, daß eine REM-Zeile, wenn sie Sprungziel ist, nicht das letzte Statement mit einer 1-, 2-, 3- oder 4-stelligen Zeilennummer sein darf, was aber wohl selten vorkommt.

Eine Eigenschaft des BASIC-Interpreters ist unangenehm: Folgen auf ein GOSUB mehr als 255 Zeichen in einer Zeile, so stürzt er ab, da er das GOSUB nach Retten der Pointer wie ein GOTO behandelt. Bei der Ausführung des GOTO aber wird der Rest der dieses Statement enthaltenden Zeile als Kommentar behandelt und übergangen; der Interpreter sucht in einer Routine ab hex C811 die nächste Zeile (3) in einer Y-indizierten Schleife, in der er in diesem Fall hängenbleibt. Bei Auffinden eines GOSUB muß also die Kompaktierung wie hier nach spätestens weiteren 255 Zeichen abgebrochen werden.

Findet der Kompaktor ein Zeichen in einem String, welches einem Token entspricht, so führte dies bisher zu einem Abbruch der Kompaktierung (z.B. entspricht 'CURSOR UP' dem Token für ON). Auch diese unerwünschte Eigenschaft konnte entfernt werden. Weiterhin wird vermieden, daß am Ende einer Zeile zwei Doppelpunkte auftreten, wenn dort ein REM auf einen Doppelpunkt folgte.

Betriebserfahrungen mit dem Compactor

Durch Kompaktieren läßt sich eine Speicherplatzeinsparung von 20 bis 40% erreichen. Die Geschwindigkeitserhöhung beträgt typisch 10%. Zur Kompaktierung eines 20K-Programmes werden ca. 40 Sek. benötigt. Das komprimierte Programm läßt sich noch mit SAVE abspeichern. Enthält es jedoch nur eine einzige Zeile mit mehr als 255 Zeichen, so kann es nicht mehr verändert werden. Das bloße Abschicken einer Zeile führt dann zum Absturz des BASIC-Interpreters.

Nach den Beobachtungen läßt sich ein extern abgespeichertes und kompaktiertes BASIC-Programm nicht mehr im Direktmodus einladen. Es ist vielmehr von einem BASIC-Programm her aufzurufen, z.B. 10 LOAD "..." oder 10 DLOAD "...". Es lädt dann ab hex 0400 ff. und überschreibt dabei das Ladeprogramm.

Literatur: (1) H. Steder: BASIC-Compactor, 65xx MICRO MAG, Heft 25, Seite 48-53. (2) H. Kohorst: Der geknackte CBM, MC, Heft 1/82, Seiten 30-32. (3) R. Martin, D. Smode: ROM und RAM bei PET und CBM, FUNKSCHAU-Sonderheft 33, Seiten 58-69.

```
LINE# LOC  CODE LINE

0001  0000                    .OPT MEMORY
0002  0000              ; ******************************
0003  0000              ; *                            *
0004  0000              ; *  BASIC-COMPACTOR FOR       *
0005  0000              ; *  CBM 3032                  *
0006  0000              ; *                            *
0007  0000              ; *  HANS - GEORG LANGE        *
0008  0000              ; *          10/07/82          *
0009  0000              ; *                            *
0010  0000              ; ******************************
0011  0000              ;
0012  0000              ; CHARS AND TOKENS TO BE LOOKED AT
0013  0000              ;
0014  0000              SPACE  = $20
0015  0000              QUOTE  = $22
0016  0000              COLON  = $3A
0017  0000              GOTO   = $89
0018  0000              IF     = $8B
0019  0000              GOSUB  = $8D
0020  0000              REM    = $8F
0021  0000              ON     = $91
0022  0000              TO     = $A4
0023  0000              THEN   = $A7
0024  0000              GO     = $CB
0025  0000              RUN    = $8A
0026  0000              ;
0027  0000              ; POINTERS
0028  0000              ;
0029  0000              PNTR1  = $77          ; CHARGET POINTER
0030  0000                     * = $20
0031  0020              PNTR2  *=*+2
0032  0022              PNTR3  *=*+2
0033  0024              PNTR4  *=*+2
```

```
0034  0026                    PSFLG  *=*+1
0035  0027                    ACTEMP = 1
0036  0027                           *=$B9
0037  00B9                    GTFLAG *=*+1
0038  00BA                    CCNT   *=*+1
0039  00BB                    QTFLAG *=*+1
0040  00BC                    ;
0041  00BC                    ; BASIC AND OS - POINTERS
0042  00BC                    ;
0043  00BC                    HXNUM  = $11
0044  00BC                    BASTRT = $28
0045  00BC                    VARST  = $2A
0046  00BC                    INDEX  = $1F
0047  00BC                    LOWTR  = $5C
0048  00BC                    ;
0049  00BC                    ; ROM-ROUTINES
0050  00BC                    ;
0051  00BC                    CHRGET = $70
0052  00BC                    CLRVAL = $C575          ;($B5EA)
0053  00BC                    LINGET = $C873          ;($B8F6)
0054  00BC                    BASIC  = $C389          ;($B3FF)
0055  00BC                    RDRUB  = $FFCF
0056  00BC                    CRLF   = $FDD0          ;($D534)
0057  00BC                    OUTDIS = $FFD2
0058  00BC                    SRCHLN = $C52C          ;($B5A3)
0059  00BC                    ADRFAC = $DB55          ;($CD7F)
0060  00BC                    FACSTR = $DCEB          ;($CF95)
0061  00BC                    OUTSTR = $DCE6          ;($BB1D)
0062  00BC                    ;
0063  00BC                           *=32000
0064  7D00                    ;
0065  7D00  A0 7F             PASS1  LDY #>TXT1
0066  7D02  A9 3E                    LDA #<TXT1
0067  7D04  20 E6 DC                 JSR OUTSTR
0068  7D07  A9 00                    LDA #0
0069  7D09  85 26                    STA PSFLG
0070  7D0B  20 B8 7E          PASS14 JSR SETBEG
0071  7D0E  4C 57 7D                 JMP LP36
0072  7D11                    ;
0073  7D11                    ; MAIN PARSE-LOOP
0074  7D11                    ;
0075  7D11  B1 77             LOOP1  LDA (PNTR1),Y
0076  7D13  F0 3E                    BEQ LP32
0077  7D15  C9 22             LOOP14 CMP #QUOTE
0078  7D17  D0 0F                    BNE TST1
0079  7D19  91 20                    STA (PNTR2),Y
0080  7D1B  20 61 7E          QUOT   JSR SHIFT
0081  7D1E  B1 77                    LDA (PNTR1),Y
0082  7D20  91 20                    STA (PNTR2),Y
0083  7D22  F0 33                    BEQ LP36
0084  7D24  C9 22                    CMP #QUOTE
0085  7D26  D0 F3                    BNE QUOT
0086  7D28  C9 8F             TST1   CMP #REM
0087  7D2A  D0 15                    BNE TST2
0088  7D2C  20 A3 7F                 JSR DECPT
0089  7D2F  B1 77                    LDA (PNTR1),Y
0090  7D31  C9 3A                    CMP #COLON
```

```
0091  7D33  F0 03        BEQ RLP
0092  7D35  20 61 7E     JSR SHIFT
0093  7D38  20 67 7E     RLP   JSR SHFT1
0094  7D3B  B1 77        LDA (PNTR1),Y
0095  7D3D  D0 F9        BNE RLP
0096  7D3F  F0 12        BEQ LP32
0097  7D41  C9 20        TST2  CMP #SPACE
0098  7D43  F0 07        BEQ LP2
0099  7D45  20 61 7E     JSR SHIFT
0100  7D48  90 07        BCC LP3
0101  7D4A  B0 2B        BCS END
0102  7D4C  20 67 7E     LP2   JSR SHFT1
0103  7D4F  B0 26        BCS END
0104  7D51  B1 77        LP3   LDA (PNTR1),Y
0105  7D53  91 20        LP32  STA (PNTR2),Y
0106  7D55  D0 BE        BNE LOOP14
0107  7D57  A0 05        LP36  LDY #5
0108  7D59  B1 77        LDA (PNTR1),Y
0109  7D5B  C9 8F        CMP #REM
0110  7D5D  D0 08        BNE NXTL
0111  7D5F  20 67 7E     RLP1  JSR SHFT1
0112  7D62  88           DEY
0113  7D63  D0 FA        BNE RLP1
0114  7D65  F0 AA        BEQ LOOP1
0115  7D67  A2 05        NXTL  LDX #5
0116  7D69  A0 00        LDY #0
0117  7D6B  20 61 7E     NXTL4 JSR SHIFT
0118  7D6E  B1 77        LDA (PNTR1),Y
0119  7D70  91 20        STA (PNTR2),Y
0120  7D72  CA           DEX
0121  7D73  D0 F6        BNE NXTL4
0122  7D75  F0 9A        BEQ LOOP1
0123  7D77              ;
0124  7D77              ; WALK-THROUGH COMPLETE
0125  7D77              ; NOW CORRECT LINKS AND
0126  7D77              ; POINTERS
0127  7D77              ;
0128  7D77  20 E7 7E     END   JSR LNKADJ
0129  7D7A  20 67 7E     JSR SHFT1
0130  7D7D  20 67 7E     JSR SHFT1
0131  7D80  A5 77        LDA PNTR1
0132  7D82  85 2A        STA VARST
0133  7D84  A5 78        LDA PNTR1+1
0134  7D86  85 2B        STA VARST+1
0135  7D88  20 75 C5     JSR CLRVAL
0136  7D8B  A5 26        END1  LDA PSFLG
0137  7D8D  F0 03        BEQ PASS2
0138  7D8F  4C B9 C3     END14 JMP BASIC
0139  7D92              ;
0140  7D92              ; IN THE SECOND PASS
0141  7D92              ; CONCATENATE ALL LINES
0142  7D92              ; AS POSSIBLE
0143  7D92              ;
0144  7D92  A0 7F        PASS2 LDY #>TXT2
0145  7D94  A9 35        LDA #<TXT2
0146  7D96  20 E6 DC     JSR OUTSTR
0147  7D99  20 D0 FD     JSR CRLF
0148  7D9C  E6 26        INC PSFLG
```

```
0149  7D9E  20 B8 7E          JSR SETBEG
0150  7DA1  20 D1 7E          JSR SETTAB
0151  7DA4  A0 03             LDY #3
0152  7DA6  20 14 7F          JSR SVNUM
0153  7DA9  A9 00             LDA #0
0154  7DAB  85 B9             STA GTFLAG
0155  7DAD  85 BA             STA CCNT
0156  7DAF  85 BB             STA QTFLAG
0157  7DB1  20 DE 7E  LOOP2   JSR XSHIFT
0158  7DB4  90 03             BCC LOOP25
0159  7DB6  4C 39 7E          JMP PASS21
0160  7DB9  20 70 00  LOOP25  JSR CHRGET
0161  7DBC  C9 00      LOOP28  CMP #0          ; END OF LINE?
0162  7DBE  F0 F1             BEQ LOOP2
0163  7DC0  C9 22             CMP #QUOTE
0164  7DC2  D0 08             BNE LOOP21      ;WASN'T A QUOTE
0165  7DC4  48                PHA
0166  7DC5  A5 BB             LDA QTFLAG      ;WAS QUOTE, INVERT FLAG
0167  7DC7  49 FF             EOR #$FF
0168  7DC9  85 BB             STA QTFLAG
0169  7DCB  68                PLA
0170  7DCC  24 BB      LOOP21  BIT QTFLAG      ;INSIDE QUOTED STRING
0171  7DCE  D0 1C             BNE LOOP27      ;YES, IGNORE TOKENS
0172  7DD0  C9 89             CMP #GOTO
0173  7DD2  F0 29             BEQ GETNUM
0174  7DD4  C9 8D             CMP #GOSUB
0175  7DD6  F0 25             BEQ GETNUM
0176  7DD8  C9 A7             CMP #THEN
0177  7DDA  F0 21             BEQ GETNUM
0178  7DDC  C9 8A             CMP #RUN
0179  7DDE  F0 1D             BEQ GETNUM
0180  7DE0  C9 8B             CMP #IF
0181  7DE2  F0 41             BEQ IFLP
0182  7DE4  C9 91             CMP #ON
0183  7DE6  F0 3D             BEQ IFLP
0184  7DE8  C9 CB             CMP #GO
0185  7DEA  F0 0E             BEQ LOOP29
0186  7DEC  A5 B9      LOOP27  LDA GTFLAG      ;PENDING GOSUB ?
0187  7DEE  F0 C9             BEQ LOOP25      ;NO
0188  7DF0  E6 BA             INC CCNT        ;YES INCREMENT COUNT
0189  7DF2  A5 BA             LDA CCNT
0190  7DF4  C9 AF             CMP #$FF-80     ;LINELENGTH > 175 ?
0191  7DF6  B0 2D             BCS IFLP        ;YES, ABORT CONCATENATI
0192  7DF8  90 BF             BCC LOOP25
0193  7DFA  20 70 00  LOOP29  JSR CHRGET
0194  7DFD  C9 8D      GETNUM  CMP #GOSUB      ;FOUND GOSUB ?
0195  7DFF  D0 02             BNE LOOP2A
0196  7E01  E6 B9             INC GTFLAG      ;YES, SET FLAG
0197  7E03  20 70 00  LOOP2A  JSR CHRGET
0198  7E06  B0 B4             BCS LOOP28
0199  7E08  48                PHA
0200  7E09  A5 B9             LDA GTFLAG      ;PENDING GOSUB ?
0201  7E0B  F0 07             BEQ LOOP2B      ;NO
0202  7E0D  A5 BA             LDA CCNT
0203  7E0F  18                CLC             ;YES,
0204  7E10  69 05             ADC #5          ;ADD 5 TO COUNT FOR TO
0205  7E12  85 BA             STA CCNT        ;AND LINE-#
0206  7E14  68        LOOP2B  PLA
```

```
0207  7E15  20 57 7F          JSR CONVRT
0208  7E18  48               PHA
0209  7E19  20 1F 7F          JSR INTAB
0210  7E1C  68               PLA
0211  7E1D  F0 92            BEQ LOOP2
0212  7E1F  C9 2C            CMP #',
0213  7E21  F0 DA            BEQ GETNUM
0214  7E23  D0 94            BNE LOOP25
0215  7E25  C8          IFLP  INY
0216  7E26  B1 77            LDA (PNTR1),Y
0217  7E28  D0 FB            BNE IFLP
0218  7E2A  C8               INY
0219  7E2B  C8               INY
0220  7E2C  C8               INY
0221  7E2D  20 14 7F          JSR SVNUM
0222  7E30  A9 00            LDA #0
0223  7E32  85 BA            STA CCNT        ;RESET CHARACTER-COUNT
0224  7E34  85 B9            STA GTFLAG      ;AND FLAG
0225  7E36  4C B9 7D          JMP LOOP25
0226  7E39                ;
0227  7E39                ; REPLACE ALL LINE-NUMBERS
0228  7E39                ; NOT FOUND BY SPACES
0229  7E39                ;
0230  7E39  20 B8 7E    PASS21 JSR SETBEG
0231  7E3C  20 DE 7E    PS213  JSR XSHIFT
0232  7E3F  20 67 7E    PS21   JSR SHFT1
0233  7E42  90 03            BCC *+5
0234  7E44  4C 0B 7D          JMP PASS14
0235  7E47  B1 77            LDA (PNTR1),Y
0236  7E49  D0 F4            BNE PS21
0237  7E4B  A5 2A            LDA VARST
0238  7E4D  85 22            STA PNTR3
0239  7E4F  A5 2B            LDA VARST+1
0240  7E51  85 23            STA PNTR3+1
0241  7E53  20 78 7E          JSR COMPAR
0242  7E56  90 E4            BCC PS213
0243  7E58  A9 3A            LDA #COLON
0244  7E5A  91 77            STA (PNTR1),Y
0245  7E5C  20 AB 7E          JSR NSPACE
0246  7E5F  F0 DE            BEQ PS21
0247  7E61                ;
0248  7E61  E6 20       SHIFT  INC PNTR2
0249  7E63  D0 02            BNE *+4
0250  7E65  E6 21            INC PNTR2+1
0251  7E67  E6 77       SHFT1  INC PNTR1
0252  7E69  D0 02            BNE *+4
0253  7E6B  E6 78            INC PNTR1+1
0254  7E6D  A5 78            LDA PNTR1+1
0255  7E6F  C5 2B            CMP VARST+1
0256  7E71  90 04            BCC *+6
0257  7E73  A5 77            LDA PNTR1
0258  7E75  C5 2A            CMP VARST
0259  7E77  60               RTS
0260  7E78                ;
0261  7E78  A0 03       COMPAR LDY #3
0262  7E7A  A2 00            LDX #0
0263  7E7C  B1 77            LDA (PNTR1),Y
0264  7E7E  C1 22            CMP (PNTR3,X)
```

```
0265  7E80  F0 0A        BEQ CMP1
0266  7E82  20 9A 7E     JSR INCP
0267  7E85  20 9A 7E     CMPO   JSR INCP
0268  7E88  90 EE        BCC COMPAR
0269  7E8A  B0 0B        BCS CRET
0270  7E8C  20 9A 7E     CMP1   JSR INCP
0271  7E8F  C8           INY
0272  7E90  B1 77        LDA (PNTR1),Y
0273  7E92  C1 22        CMP (PNTR3,X)
0274  7E94  D0 EF        BNE CMPO
0275  7E96  18           CLC
0276  7E97  A0 00        CRET   LDY #0
0277  7E99  60           RTS
0278  7E9A              ;
0279  7E9A  E6 22        INCP   INC PNTR3
0280  7E9C  D0 02        BNE *+4
0281  7E9E  E6 23        INC PNTR3+1
0282  7EA0  A5 23        LDA PNTR3+1
0283  7EA2  C5 25        CMP PNTR4+1
0284  7EA4  90 04        BCC *+6
0285  7EA6  A5 22        LDA PNTR3
0286  7EA8  C5 24        CMP PNTR4
0287  7EAA  60           RTS
0288  7EAB              ;
0289  7EAB  A2 04        NSPACE LDX #4
0290  7EAD  20 67 7E     NSPC2  JSR SHFT1
0291  7EB0  A9 20        LDA #$20
0292  7EB2  91 77        STA (PNTR1),Y
0293  7EB4  CA           DEX
0294  7EB5  D0 F6        BNE NSPC2
0295  7EB7  60           RTS
0296  7EB8              ;
0297  7EB8  A5 28        SETBEG LDA BASTRT
0298  7EBA  85 77        STA PNTR1
0299  7EBC  85 20        STA PNTR2
0300  7EBE  A5 29        LDA BASTRT+1
0301  7EC0  85 78        STA PNTR1+1
0302  7EC2  85 21        STA PNTR2+1
0303  7EC4  A5 77        LDA PNTR1
0304  7EC6  D0 04        BNE *+6
0305  7EC8  C6 78        DEC PNTR1+1
0306  7ECA  C6 21        DEC PNTR2+1
0307  7ECC  C6 77        DEC PNTR1
0308  7ECE  C6 20        DEC PNTR2
0309  7ED0  60           RTS
0310  7ED1              ;
0311  7ED1  A5 2A        SETTAB LDA VARST
0312  7ED3  85 22        STA PNTR3
0313  7ED5  85 24        STA PNTR4
0314  7ED7  A5 2B        LDA VARST+1
0315  7ED9  85 23        STA PNTR3+1
0316  7EDB  85 25        STA PNTR4+1
0317  7EDD  60           RTS
0318  7EDE              ;
0319  7EDE  A2 04        XSHIFT LDX #4
0320  7EE0  20 67 7E     JSR SHFT1
0321  7EE3  CA           DEX
0322  7EE4  D0 FA        BNE XSHIFT+2
```

```
0323  7EE6  60              RTS
0324  7EE7                  ;
0325  7EE7                  ;  ADJUST THE LINKS OF
0326  7EE7                  ;  THEN CONCATENATED LINES
0327  7EE7                  ;
0328  7EE7  20 B8 7E     LNKADJ JSR SETBEG
0329  7EEA  20 61 7E            JSR SHIFT
0330  7EED  A0 01        LNK1   LDY #1
0331  7EEF  B1 77               LDA (PNTR1),Y
0332  7EF1  F0 41               BEQ LNK3
0333  7EF3  A5 77               LDA PNTR1
0334  7EF5  85 20               STA PNTR2
0335  7EF7  A5 78               LDA PNTR1+1
0336  7EF9  85 21               STA PNTR2+1
0337  7EFB  20 DE 7E            JSR XSHIFT
0338  7EFE  88                  DEY
0339  7EFF  20 67 7E     LNK2   JSR SHFT1
0340  7F02  B1 77               LDA (PNTR1),Y
0341  7F04  D0 F9               BNE LNK2
0342  7F06  20 67 7E            JSR SHFT1
0343  7F09  A5 77               LDA PNTR1
0344  7F0B  91 20               STA (PNTR2),Y
0345  7F0D  C8                  INY
0346  7F0E  A5 78               LDA PNTR1+1
0347  7F10  91 20               STA (PNTR2),Y
0348  7F12  D0 D9               BNE LNK1
0349  7F14  B1 77        SVNUM  LDA (PNTR1),Y
0350  7F16  85 11               STA HXNUM
0351  7F18  C8                  INY
0352  7F19  B1 77               LDA (PNTR1),Y
0353  7F1B  85 12               STA HXNUM+1
0354  7F1D  A0 00               LDY #0
0355  7F1F  A5 11        INTAB  LDA HXNUM
0356  7F21  91 24               STA (PNTR4),Y
0357  7F23  A5 12               LDA HXNUM+1
0358  7F25  C8                  INY
0359  7F26  91 24               STA (PNTR4),Y
0360  7F28  88                  DEY
0361  7F29  18           INTB   CLC
0362  7F2A  A9 02               LDA #2
0363  7F2C  65 24               ADC PNTR4
0364  7F2E  85 24               STA PNTR4
0365  7F30  90 02               BCC *+4
0366  7F32  E6 25               INC PNTR4+1
0367  7F34  60           LNK3   RTS
0368  7F35                  ;
0369  7F35  0D           TXT2   .BYT 13,13,' PASS2',0
0369  7F36  0D
0369  7F37  20 50
0369  7F3D  00
0370  7F3E  93           TXT1   .BYT $93,'BASIC-COMPACTOR',13,13,
0370  7F3F  42 41                                          ' PASS1',(
0370  7F4E  0D
0370  7F4F  0D
0370  7F50  20 50
0370  7F56  00
0371  7F57                  ;
```

```
0372  7F57  85 01        CONVRT STA ACTEMP
0373  7F59  A5 77               LDA PNTR1
0374  7F5B  48                  PHA
0375  7F5C  A5 78               LDA PNTR1+1
0376  7F5E  48                  PHA
0377  7F5F  A5 01               LDA ACTEMP
0378  7F61  20 73 C8            JSR LINGET
0379  7F64  85 01               STA ACTEMP
0380  7F66  20 2C C5            JSR SRCHLN
0381  7F69  90 05               BCC NXTLIN
0382  7F6B  68                  PLA
0383  7F6C  68                  PLA
0384  7F6D  18                  CLC
0385  7F6E  90 2E               BCC PTEND
0386  7F70  A0 02        NXTLIN LDY #2
0387  7F72  B1 5C               LDA (LOWTR),Y
0388  7F74  C8                  INY
0389  7F75  85 11               STA HXNUM
0390  7F77  B1 5C               LDA (LOWTR),Y
0391  7F79  85 12               STA HXNUM+1
0392  7F7B  A5 11               LDA HXNUM
0393  7F7D  85 60               STA $60
0394  7F7F  A5 12               LDA HXNUM+1
0395  7F81  85 5F               STA $5F
0396  7F83  A2 90               LDX #$90
0397  7F85  38                  SEC
0398  7F86  20 55 DB            JSR ADRFAC
0399  7F89  20 EB DC            JSR FACSTR
0400  7F8C  68                  PLA
0401  7F8D  85 5D               STA LOWTR+1
0402  7F8F  68                  PLA
0403  7F90  85 5C               STA LOWTR
0404  7F92  A0 00               LDY #0
0405  7F94  B9 00 01     PTLOP  LDA $100,Y
0406  7F97  F0 05               BEQ PTEND
0407  7F99  91 5C               STA (LOWTR),Y
0408  7F9B  C8                  INY
0409  7F9C  D0 F6               BNE PTLOP
0410  7F9E  A0 00        PTEND  LDY #0
0411  7FA0  A5 01               LDA ACTEMP
0412  7FA2  60                  RTS
0413  7FA3
0414  7FA3  A5 77        DECPT  LDA PNTR1
0415  7FA5  38                  SEC
0416  7FA6  E9 01               SBC #1
0417  7FA8  85 77               STA PNTR1
0418  7FAA  B0 02               BCS DECPT2
0419  7FAC  C6 78               DEC PNTR1+1
0420  7FAE  A5 20        DECPT2 LDA PNTR2
0421  7FB0  E9 01               SBC #1
0422  7FB2  85 20               STA PNTR2
0423  7FB4  B0 02               BCS PTRTS
0424  7FB6  C6 21               DEC PNTR2+1
0425  7FB8  60           PTRTS  RTS
0426  7FB9                     .END

ERRORS = 0000
```

Dr. A. Schnell, 5100 Aachen

Disassemblieren des CBM-DOS

Über das Betriebssystem der Commodore Diskettenlaufwerke 3040, 4040 und 4031 ist relativ wenig bekannt. Dies liegt wohl auch daran, daß der Speicher innerhalb des Diskettenlaufwerkes nicht durch PEEK oder POKE direkt zugänglich ist. Das untenstehende Programm zeigt jedoch eine einfache Möglichkeit, auf einzelne Bytes im Diskettenlaufwerk zuzugreifen. Falls dieser Programmteil anstatt von PEEK-Befehlen in einem BASIC-Disassemblerprogramm eingebaut wird, läßt sich das DOS-Betriebssystem auslisten und analysieren. Statt des üblichen X=PEEK(Y), wobei Y die zu untersuchende Adresse ist, programmiert man für DOS-Adressen:

```
AA=Y:GOSUB10000:X=AA

10000 OPEN15,8,15
10010 B=INT(AA/256)
10020 AA=AA-B*256:HH=B:LL=AA
10030 PRINT#15,"M-R"CHR$(LL)CHR$(HH)
10040 GET#15,AA$:IF AA$="" THEN AA$=CHR$(0)
10050 AA=ASC(AA$)
10060 CLOSE15
10070 RETURN
```

#

Dirk Sanders, 6100 Darmstadt

Textrettung (AIM 65)

Bei AIM 65 Editor-Files

Wenn in einer Editor-Leerzeile (hex 20 oder 0D) mit dem C-Befehl (Change) ein hex 20 (=SPACE) gelöscht wird, so bleibt hex 0D 0D übrig. Wenn dieser Text auf Band gespeichert wird, bleibt später der Ladevorgang an dieser Stelle stehen (2x RETURN wird als Textende angesehen). Dabei läuft das Band ohne ERROR-Meldung weiter, der Rest des Files wird aber nicht mehr geladen. Unbeabsichtigt erzeugt man diesen Zustand meist am Anfang eines einmalig abgelegten langen Files (siehe AIM Spezial 9).

Das folgende Programm mit IN=USER unterdrückt aufeinanderfolgende hex 0D und ermöglicht die Wiedergewinnung solcher Files. Nach dem Ladevorgang muß man RESET geben.

```
0000          ;WIRD IN EINER EDITOR LEERZEILE ( $0D $20 $0D )
0000          ;MIT DEM "C" BEFEHL  $20 (= SPACE) GELOESCHT,
0000          ;BLEIBT ( $0D $0D ) UEBRIG. WIRD DER TEXT NUN
0000          ;AUF BAND GESPEICHERT, BLEIBT SPAETER DER LADE-
0000          ;VORGANG  AN DIESER STELLE STEHEN ( WIE 2 MAL
0000          ;RETURN), DAS BAND LAUFT OHNE ERROR MEL-
0000          ;DUNG WEITER, ABER DER REST DES FILES WIRD
0000          ;NICHT MEHR GELADEN.
0000          ;UNBEABSICHTIGT ERZEUGT MAN DIESEN ZUSTAND
0000          ;MEIST AM ANFANG VON EINMALIG ABGE-
0000          ;LEGTEN LANGEN FILES (SIEHE AIM SPEZIAL 9 )
0000          ;FOLGENDES PROGRAMM MIT USER-IN AUFRUF UNTER-
0000          ;DRUEKT AUFEINANDERFOLGENDE $0D UND MAN BEKOM-
0000          ;MT SEIN FILE WIEDER. (Nach Ladevorgang: RESET)
```

```
0000                    *=$10C
010C            4CDB0F  JMP BEG         ;F1 UM USER VERKTOR AUSZURICHTEN
010F                    *=$FDB
0FDB FNAM               =$E8A2
0FDB LOADTA             =$E32F
0FDB TIBYTE             =$ED3B
0FDB INFLG              =$A412
0FDB UIN                =$108
0FDB
0FDB SAVE               =$FF            ;ODER ANDERE SPEICHERSTELLE
0FDB
0FDB BEG        A9E6    LDA ßkANF       ;USER INPUT LADEN
0FDD            8D0801  STA UIN
0FE0            A90F    LDA ßgANF
0FE2            8D0901  STA UIN+1
0FE5            60      RTS
0FE6
0FE6 ANF        B00A    BCS TRANS       ;UIN ZIELADRESSE
0FE8            A900    LDA ß0          ;SAVE NEUTRALISIEREN
0FEA            85FF    STA SAVE        ;ALSO NICHT $0D !
0FEC            20A2E8  JSR FNAM        ; FILE NAME UND TAPE ?
0FEF            4C2FE3  JMP LOADTA      ;ERSTEN BLOCK LADEN
0FF2
0FF2 TRANS
0FF2            203BED  JSR TIBYTE      ;HOLE CHR VON TAPE-PUFFER
0FF5            C90D    CMP ß$0D        ;RETURN ?
0FF7            D004    BNE TRANS1
0FF9            C5FF    CMP SAVE        ;JA, WAR LETZTER AUCH $0D?
0FFB            F0F5    BEQ TRANS       ;---)JA, DANN NEUEN CHR HOLEN
0FFD TRANS1     85FF    STA SAVE        ;IN SAVE AUFHEBEN
0FFF            60      RTS
1000                    .END
1000             ERRORS= 0000
```

#

Errata

Berechnung von Pi mit großer Genauigkeit, Nr. 22, S. 41: Der Autor, Herr Horst Brettin weist auf einen Wiedergabefehler in der Formel von John Machin hin. Sie lautet richtig:

$$\pi = 16*ARCTAN(1/5) - 4*ARCTAN(1/239)$$

6502 Multiplikation, Division, Nr. 22, S. 34: An der Programmstelle hex 0258 ist der Befehl CLC hinzuzufügen. Der Rest der Routinen verschiebt sich dadurch. (Vielen Dank Herr Albrecht!)

Einkommensteuerberechnung 1981, Nr. 21, S. 46 ff.: Herrn Dr. Trefny in GE-Buer weist auf das Folgende hin: Die Zeilen 1830, 1840 und 3310 sind unvollständig. Hinter GB(0) muß =1 stehen, hinter GB(1) muß =1 stehen. Da der Commodore-Drucker alle Texte ab Zeile 3010 als Grafik druckt, sollten alle Texte ab dort in Kleinbuchstaben stehen. Bei den Einkünften von Zeile 2400 bis 2470 werden auch negative Zahlen (Verluste) angenommen. Das muß verhindert werden, da sonst falsche Ergebnisse entstehen. Dazu folgende Änderungsvorschläge:

```
365 IF X <Ø THEN X=Ø:PRINT "VERLUSTVORTRAG POSITIV AM SCHLUSS EINZUTRAGEN
366 RETURN
```

Die Zeilen 2160 und 2170 sind vor dem RETURN mit GOSUB 365 : zu ergänzen.

AIM 65 FORTH User's Guide (Rockwell International): Wegen des allgemeinen Interesses werden die in Heft 7 (Jan. 1982) der Hauszeitschrift INTERACTIVE veröffentlichten Fehlerberichtigungen nachfolgend entnommen und parallel veröffentlicht:

ERRATA AND UPDATE TO FORTH MANUAL

p. 4-30 If you get a compilation error when compiling from the text editor, the 16 bit pointer at location $00DF will indicate where the error occurred in the text buffer.

Perform a

```
HEX DF @ 20 - CR 20 TYPE
```

to see the last 20 characters before the error occurred.

p. 4-31 The last sentence in the warning should start "FORGETing TASK then . . ."

p. 4-37 The explanation for LEAVE is incomplete. Refer to the definition on page B-29.

p. 4-55 The short sequence at the top of the page SHOULD read

```
BASE @ DUP DECIMAL.
```

The word BASE? should be defined as follows:

```
: BASE? BASE @ DUP DECIMAL .
        BASE ! ;
```

Otherwise, BASE gets changed to decimal whenever BASE? is executed.

p. 4-57 The sequence at the top of the page should read:

```
DECIMAL 65 EMIT
```

p. 5-1 The second paragraph in section 5.1.1 should read:

". . . quotient (top of the stack) and the remainder (second on the stack) . . .

p. 5-14 Delete the two numbers (500 5) from the example at the bottom of the page. Those numbers will be on the stack from the sequence immediately above. You must press the RETURN key after CR TYPE CR.

p. 5-19 Change the number 7F to 1F in the second code sequence from the bottom of the page. The last code sequence should read:

```
LATEST CR ID.
```

p. 6-6 The hex data at location $030F should be changed from 04 to D4.

p. 6-11 RP! does not work correctly. It returns 101 decimal (65 hex) instead of 101 hex. If you need to use it in your assembly language you'll have to redefine it.

```
HEX 101 CONSTANT RP!
```

p. 6-13 The text in the parenthesis at the bottom of the page should read:

(the <space> bar was pressed here)

p. 7-4 Delete the second line (ASSEMBLER) in the example in 7.2.1. The assembler will be called when CODE is executed.

p. 7-10 In 7.3.4d the code should read:

```
: DUMMY ARM [ SMUDGE
```

see J-8 for ARM.

p. 8-6 Don't forget to insert a space immediately after the first parenthesis in all the comment lines in both program examples.

p. 8-10 Again the comment in the ?STROBE example needs to have a space after the first parenthesis.

p. 10-5 The first sentence in paragraph 9 should start "Set and verify . . ."

p. 10-6 Change step 12 to read:

(12) Restore dummy word TASK, verify that TASK is entered into the FORTH dictionary and its PFA is $309 and read the dictionary pointer (DP).

```
: TASK ;
VLIST
309 TASK <space>
        DP @ . <return> 30B OK
```

Note that any new words now added to the dictionary will be located at $30B on up.

p. 10-7 Change step 13 also.

(13) Display the link field address of TASK to get the address of the last application word for use in step 15.

```
' TASK LFA .S @ 0 D.
        <return> 305 LAST
```

p. 10-7 Change the assembly language instruction in Section 15 from LDA A0,Y to LDA (A4),Y

p. 10-11 At the top of the page change * - 800 to * - D000.

Appendix B-41 Remove 'addr' from the stack notation of 'WORD'. Also delete the last sentence in the definition.

Appendix G-3 The number of bytes for parameter name MODE is 2 (not 8).

Appendix I-2 255 places should be allotted to the variable IB (not 25).

```
0 VARIABLE IB
255 ALLOT
```

Appendix I Forth String Words.

To make this string package more nearly compatible with the one found in BASIC several changes are necessary.

Change the definitions for MID$, LEFT$ and RIGHT$ to:

```
: MID$ DROP 1- ROT + SWAP ;
: LEFT$ DROP SWAP ;
: RIGHT$ DUP 4 PICK MIN - + SWAP ;
```

Appendix I-6 Under explanation for S!, there should be a space inserted between the double quotes and 'COWS'

```
" COWS not "COWS
```

change the explanation of MID$ to the following:

MID$ gets M characters of a string starting at the Nth character position, for example

```
6 3 AS MID$ TYPE
```

will print the word EAT.

Appendix I-7 Under the explanation for VAL delete the space following the number 128.

```
" 128" VAL D.
```

Appendix I-8 Under the explanation for SUB the second program line should read:

```
" ATE"6 3 AS MID$ SUB
```

Appendix K-1 In the definition for OFF, the word DROP right before the semicolon should be deleted. It should read:

```
: OFF A004 @ 12B + DUP CR IF FFFF
DNEGATE D. ELSE . THEN ;
```

Aus der Branche

Commodore veranstaltet auch im 2. Halbjahr 1982 wieder zahlreiche Kurse in seinem neuen Schulungszentrum in 6 Frankfurt 71, Lyoner Str. 38 (Info: Herr Volker Berlipp). Themen u.a. BASIC, Arbeiten mit der Floppy, PASCAL, Assembler, IEEE-Workshop. Weiterhin sind Trainings geplant für WORDCRAFT, VISICALC, FIBU und für die Mini Mainframe (Assembler, FORTRAN, APL).

Die IHK für München und Oberbayern führt ab November 1982 verschiedene Seminare für Betreiber von Kleincomputern in ihrem Bildungszentrum in 8152 Feldkrichen-Westerham, Von Adrian-Str. 5, durch. Info von dort. Te.: 08063 - 19 41.

Rockwell International: Die Electronics Devices Division ist in Kalifornien in ein größeres Anwesen umgezogen. Neue Anschrift: 4311 Jamboree Rd., USA Newport Beach, Ca. 92660.

Über Rockwell und seine Distributoren ist jetzt ein Kurzformkatalog der 4-, 8- und 16-Bit Microprozessoren, der Einplatinen-Computer und der integralen LSI-Modems kostenlos erhältlich. Eine weitere 12-seitige Broschüre (Best.-Nr. R-20) zeigt die Fortschritte der LSI-Modemtechnologie, Änderungen der Regierungsbestimmungen und Trends, Modems als Untersysteme einzusetzen.

Für das RM-65 Computersystem (Rockwell) steht jetzt auch ein intelligenter Floppy Disk Controller zur Verfügung (RM-5101) für 5 1/4 und 8" und SS, DS, SD und DD zusammen mit Betriebssoftware, auch für AIM 65 und AIM 65/40.

Die IEEE-488 Interfaces für Olivetti-Schreibmaschinen ETxxx (ET 121, 201, 221, 231) zum Betrieb am CBM der Fa. Hard & Soft R. S. Microcomputer GmbH, Gagernstr. 4 in 8580 Bayreuth, erhielten ein neues verbessertes Betriebssystem, u.a. mit Proportionalschrift im Blocksatz, SPACE-Optimierung, vereinfachter Zusammenfassung von Befehlsketten u.a.m.. Die Maschinen schreiben dadurch mit der Maximalgeschwindigkeit von 25 Zeichen/sek. Die Firma liefert jetzt auch ihren neuen **NEWTIM (für CBM)** mit und ohne Prommer und das **Assemblersystem 8000** mit einem leistungsfähigen Editor, bedingter Assemblierung und Macro-Assembler nebst Bedienungsanleitung (in Deutsch). Tel.: 0921 - 68 877.

Editorial

Auch in diesem Heft konnte der Sprache FORTH wiederum ein gebührender Platz eingeräumt werden. Das 65xx MICRO MAG dürfte damit hierzulande eine Spitzenstellung in den Darstellungen für FORTH haben. Dabei wurden bisher weniger Anwendungs- als vielmehr Systemprogramme veröffentlicht, ein Stringpaket, Daten-SETs, Operationen mit doppelt genauen Zahlen u.a.m.. Daneben die begleitende Einführung in die Sprache und 'FORTH im Eigenbau'. Beide Themen werden noch fortgesetzt. Die systematischen Artikel sind damit keineswegs erschöpft. Viel wird noch zur inneren Struktur der Sprache zu sagen sein, zu Vokabularien und zur Beeinflussung der Kompilierung. Wichtige weitere Abschnitte betreffen die Interfaceprogrammierung, den Aufbau von Assemblern und Cross-Assemblern für andere Systeme, das CASE-Statement (ON .. GOTO) u.a..

Wegen der besonderen Anpassungsfähigkeit der Sprache wird man in einiger Zeit sicher auch Artikel über Macro-Assemblierung (auch bedingt), assoziative Tabellen und Datenbanksysteme finden. Die Leser dieser Zeitschrift seien ausdrücklich ermuntert, auch Anwenderprogramme einzureichen, an denen man sieht, wie man das eine oder andere Problem beispielhaft löst. - Wir haben heute bereits ein unübersehbar großes Angebot an fertig beziehbaren Computern und an Platinen für Anwendungscomputer. Und demnächst treten sowohl für die 65xx- wie auch für die 68xx-Familie neue Zentraleinheiten hinzu. Das gemeinsame Band für die Systembetreiber werden damit vorwiegend die Hochsprachen (wobei die Möglichkeiten des BASIC als gut dargestellt gelten dürfen) und die Assemblersprachen. Zu ihnen darf man darauf hinweisen, daß die Befehlssätze und die Adressierungsarten der CPU's viele Gemeinsamkeiten haben. Das trifft auch auf die Programmierung der Interfacebausteine zu. Mit etwas Übung sind daher die für ein System entwickelten Lösungen auf ein anderes der Familien übertragbar. In dieser Zeit des schnellen Fortschrittes darf man nicht darauf vertrauen, mit den Kenntnissen für nur eine CPU lange Zeit bestehen zu können,

wir finden ja in den neueren Computern durchaus schon mehrere Zentraleinheiten verschiedener Familien. Nach den Jahren der Einarbeitung ist das Hinzulernen eine wichtige Aufgabe.

Bei der Auswahl im großen Angebot der Systeme sollte man sich weiterhin vor Augen halten, daß die ausreichende Dokumentation für Hard- und Software sowie für die Hantierung ein entscheidendes Kriterium für den später erfolgreichen und kostengünstigen Betrieb ist. Informationslücken ziehen immer zeit- und kostenaufwendige Fehlschläge in der Entwicklung nach sich. Auf gute Entwicklungshilfsmittel für die Programmierung und Programmerprobung kann man künftig noch weniger verzichten als früher und auch nicht auf die zur Problemlösung geeigneten Sprachen, Assembler, höhere Sprachen mit ihren Datenstrukturen, Zugriff auf externe Datenfiles und wahlfreie Zugriffe. Der Anwender wird sich auch vergegenwärtigen, daß Programmentwicklungen langwierig und kostenaufwendig sind. Vorhandene Programme und Dateien stellen daher einen Wert dar, den es bei Umstellungen oder Erweiterungen unter dem Gesichtspunkt der Kompatibilität zu betrachten gilt. Kompabiltität ist weiterhin wichtig für Computer, die im täglichen Einsatz gebraucht werden. Störungsfälle sind immer möglich. Für sie sollte der Kundendienst schnell erreichbar sein. Noch besser ist es, wenn er für die Reparaturzeit ein Austauschgerät zur Verfügung stellen kann.

Die besondere Leistungsfähigkeit der vielen neuen Systeme kann daher nicht das einzige Kriterium für ihren Betrieb sein. Gleichwohl sollte man neue Möglichkeiten nutzen, wo immer das vertretbar ist (und auch darüber berichten), man erhält nach entsprechend sorgfältiger Auswahl meistens eine verbesserte Arbeitsbasis.

Für Anwendungsprozessoren ist auf die zahlreichen schon vorhandenen und kommenden 1-Chip-Prozessoren der Familien 65xx und 68xx besonders hinzuweisen, die einen sehr einfachen Schaltungsaufbau zulassen, indem der Chip nicht nur die CPU, sondern auch mehrere Interface-Ports, ggfs. A/D-Wandler, Timer und Arbeits-RAM enthält. Daneben gibt es Versionen, die auch ein EPROM nebst Brennprogramm für das EPROM auf dem Chip tragen (z.B. MC68705) und die daher für kleine Serien sehr geeignet sind. Als Steuerungsprozessoren werden sie insbesondere auch die Computer-Peripherie 'intelligenter' machen. In vielen Geräten sehen wir bereits heute zwei Prozessoren, den einen z.B. als E/A-Prozessor, den anderen in der Verarbeitung. Dieser Trend setzt sich fort, so daß man ihn bei Schaltungsentwürfen berücksichtigen sollte. - Im Gegensatz zu diesem Trend hatten wir noch vor wenigen Jahren die Neigung, alle Interfacesteuerungen von der einen Zentraleinheit ausführen zu lassen, also z.B. Tastaturdekodierung oder Druckersteuerung nebst Zeichenformung (beim AIM 65 z.B.). Inzwischen sind Tastaturdecoder, CRT-Steuerbausteine und solche für die serielle Datenübertragung eine Selbstverständlichkeit geworden. Sie haben die Computer leistungsfähiger gemacht, und wir sollten in Geräten nicht mehr auf sie verzichten. Man sollte sich in diesem Trend auch nicht scheuen, für dauerhafte Lösungen die erwähnten 1-Chip-Prozessoren für weitere noch nicht abgedeckte Schnittstellenaufgaben einzusetzen.

Mit der Herausgabe dieses Heftes treten neue Abonnementspreise in Kraft, DM 54,- im Inland und DM 59,- bei Bezug aus dem Ausland für jeweils 6 Hefte. Bei den sonstigen Bestellungen (Bücher, Zubehör, Heftnachlieferungen) werden Versandkosten von DM 2,50 je Sendung erhoben. Der Herausgeber bittet um Verständnis für diese Anpassung, die nach der empfindlichen Erhöhung der Postgebühren um 30-40% per 1.7.82 und mit zahlreichen anderen Kostensteigerungen in den vergangenen eineihalb Jahren notwendig wurde. Das 65xx MICRO MAG wird sich in Zusammenarbeit mit den freien Autoren weiterhin um die Darstellung profunder Information für den Betrieb der Systeme bemühen! In diesem Sinne auch sind die Leser gebeten, die Zeitschrift im Bekanntenkreis weiterzuentpfehlen: Je größer der Kreis der engagierten Betreiber ist, je mehr darf man auf Dauer an viellfältiger Information erwarten.

Kleinanzeigen

Verkaufe Doppelkassettenlaufwerk in Gehäuse für AIM 65, PC 100 u.a.. VB DM 160,--. MICCON-B. Heckl, Tel. 0911 - 65 17 47, Alte Wallensteinstr. 146, 8500 Nürnberg 80.

"STARTING FORTH" von L. Brodie, die beste und umfassendste Einführung in FORTH für DM 52,80. Dr. J. Schrenk, Postfach 904, 7500 Karlsruhe 41.

8192-Punkte-Graphikerw. (128x64) für CRT-2-Videointerface von Graf Elektronik, Kempten DM 75,-. Info gegen Freiumschlag: Roland Mester, Kleiststraße 8, 6054 Rodgau 3.

CBM-2/3/4/8001 Speicherbelegung, 630 Adr. 20 DM * Beschreibung von ROM-Routinen für reelle Arithm., Tape- & IEEE-Bus-I/O, Parameterübergaben an Maschinenprogramme etc. 25 DM * Zusammen 88 Seiten A4 für 35 DM * Alle ROM-Adressen als Label-Files auf Diskette 40 DM * Katalog kostenlos: H. J. Koch, Liegnitzer Str. 8, 3008 Garbsen 8.

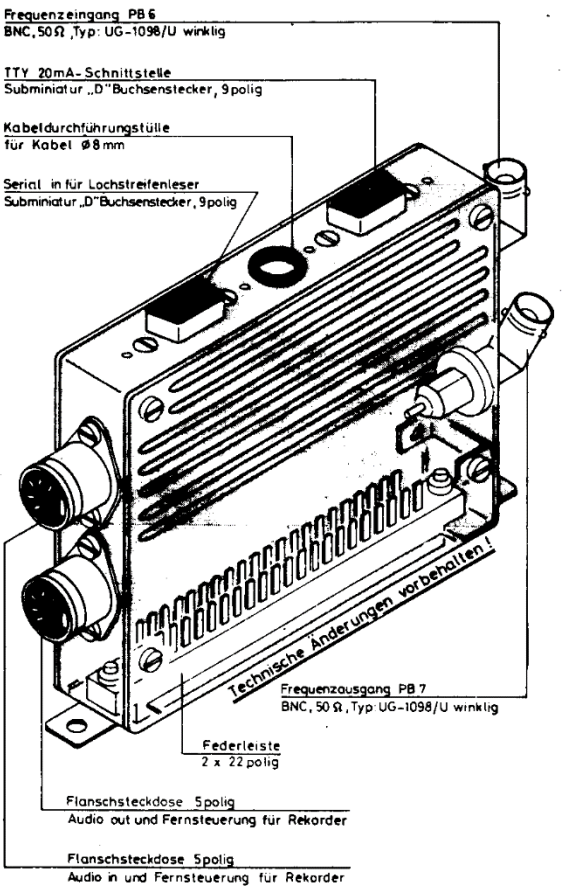
Universal-Steckverbinder

AIM 65 / Siemens PC 100

Bestückung nach Wunsch möglich

Lieferbar ab 45. KW. 81.

Preise auf Anfage



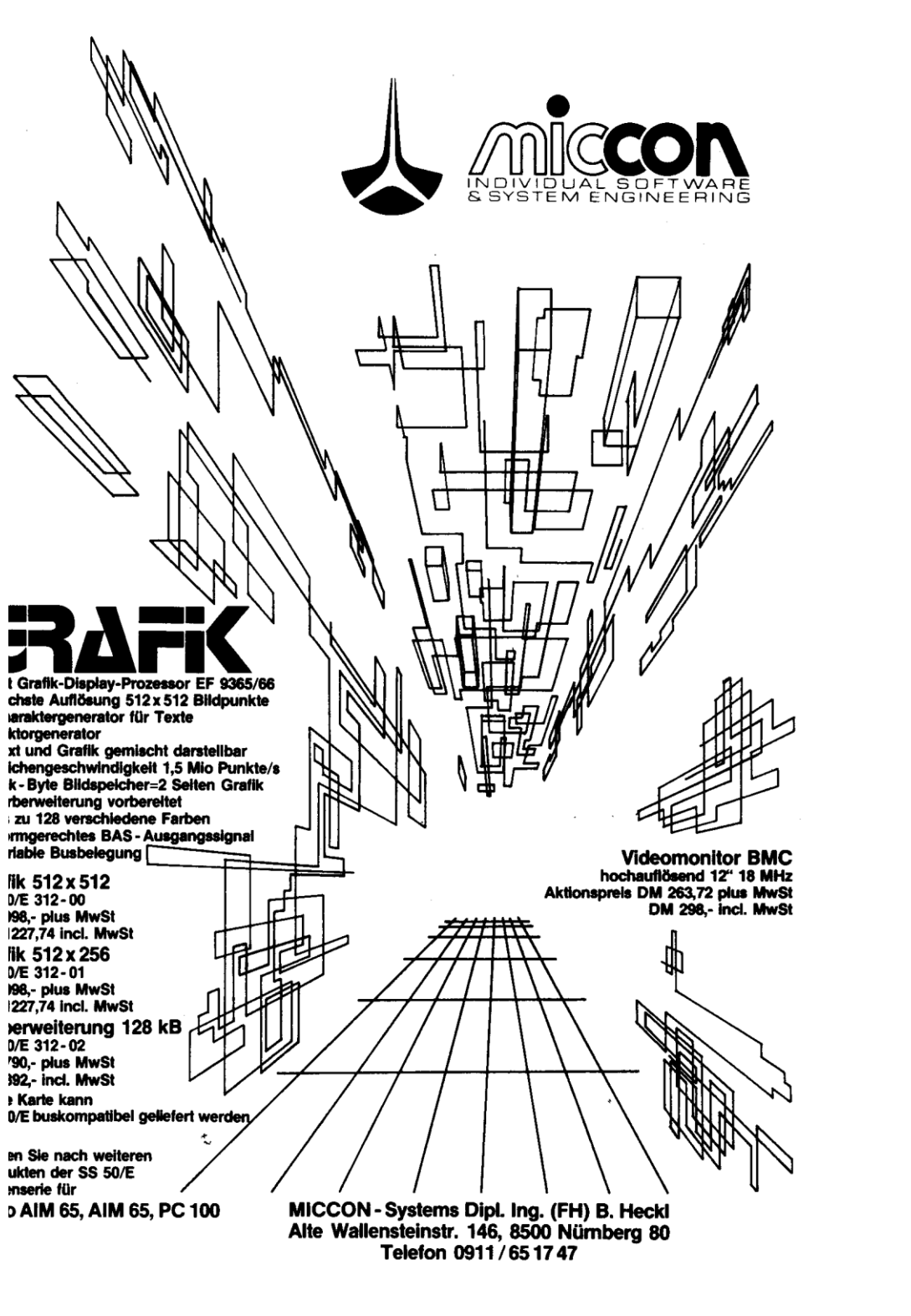
STECKER

INGENIEURBÜRO

Postfach 60 07 66

5000 Köln 60

Neue Telefon-Nummer!! Tel.: (0221) 7 12 40 18


miccon
INDIVIDUAL SOFTWARE
& SYSTEM ENGINEERING
RAFIK
t Grafik-Display-Prozessor EF 9365/66
chste Auflösung 512 x 512 Bildpunkte
araktergenerator für Texte
ktorgenerator
xt und Grafik gemischt darstellbar
chengeschwindigkeit 1,5 Mio Punkte/s
k - Byte Bildspeicher=2 Seiten Grafik
rberweiterung vorbereitet
zu 128 verschiedene Farben
rmgerechtes BAS - Ausgangssignal
riable Busbelegung
fik 512 x 512
0/E 312 - 00
98,- plus MwSt
227,74 incl. MwSt
fik 512 x 256
0/E 312 - 01
98,- plus MwSt
227,74 incl. MwSt
erweiterung 128 kB
0/E 312 - 02
90,- plus MwSt
92,- incl. MwSt
Karte kann
0/E buskompatibel geliefert werden
en Sie nach weiteren
ukten der SS 50/E
nserie für
AIM 65, AIM 65, PC 100
Videomonitor BMC
hochauflösend 12" 18 MHz
Aktionspreis DM 263,72 plus MwSt
DM 298,- incl. MwSt
MICCON - Systems Dipl. Ing. (FH) B. Heckl
Alte Wallensteinstr. 146, 8500 Nürnberg 80
Telefon 0911 / 65 17 47

DUO Plott Interface

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

Paralleles IEEE 488 - Interface, genormter IEEE-Stecker und Kabel
kompletter Zeichensatz des CBM-Computers
zwei Geräteadressen für Groß-/Grafikmodus und Textmodus
alle Funktionen der Drucker bleiben erhalten, Floppy-Kompatibilität
Deutsche Umlaute, ß und Paragraph
Problemloser Einbau, inklusive deutsches EPSON-Handbuch
Komplettpreis einschl. Kabel, CBM-Grafiksatz und Handbuch incl. MWSt
für EPSON DM 398,--
für ITOH DM 450,--

Umrüstsatz MX80 F/T auf MX82F/T

Aus Ihrem EPSON MX-80 F/T wird der MX-82 F/T
Einzelnadelansteuerung, erweiterter Befehlsatz, Elite-Schrift
Preis inkl. MWSt DM 250,--
Komplettpreis Interface, Kabel, Handbuch und Umrüstsatz incl. MWSt DM 600,--
Komplettpreis wie vor, einschl. neuem MX-80 F/T incl. MWST DM 1.998,--

Für COMMODORE

Deutsche Tastatur mit Original-Tastensätzen (keine Aufkleber)
inklusive deutschem Zeichengenerator
für 8032 DM 198,--
für 8032 und 8096 DM 298,--
nur 8096, ohne Tasten DM 98,--

Speichererweiterung auf 8096

LOS-Kompatibel, inkl. MWSt DM 898,--

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

Stellberg Computer-Systeme

COMMODORE EPSON C' ITOH SOFTWARE INTERFACE

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

CPU-Karte NICO 65

CPU 6502, 1 oder 2 MHz Takt
8 k Byte ROM, PROM, EPROM oder RAM
(Typ durch Lötbrücken festlegbar)
2 k Byte RAM als Arbeitsspeicher
2 VIA 6522 mit zusammen 40 Pin I/O
4 Zähler, Timer, Schieberegister u.a.m.
2 Steckerleisten dafür, je 22-polig
Bustreiber für alle Busse
1 Bus-Steckerleiste VG 64 zur Erweiterung
Decoder mit Adressenwahl durch Lötbrücke
Power-On-Reset, IRQ, NMI
5 V Betriebsspannung
Stromaufnahme je nach Bestückung ca. 350-510 mA ohne EPROMs

Adressenbelegung (hex):
2 k Byte RAM 0000 - 07FF
2 VIA 0800 - 0BFF, je 16 Byte
8 k Byte EPROM 6000 - 7FFF oder E000 - FFFF,
durch Lötbrücke einstellbar. Wenn beide fehlen, ist der
Bus in diesem Bereich frei.

Grundausstattung beinhaltet: 1 k RAM, 1 VIA
CPU-Karte komplett ist voll bestückt, jedoch ohne EPROMs

Preise:	**Grundausstattung**	**390,-- DM**
	Komplett	**450,-- DM**
		zzgl. MWSt

CPU-Karte NICO 69

Gleiche Konfiguration wie obige CPU-Karte NICO 65
jedoch mit CPU 6809

Preise:	**Grundausstattung**	**450,-- DM**
	Komplett	**510,-- DM**
		zzgl. MWSt

ELEKTRONISCHE BAUGRUPPEN
MICROPROZESSOR-TECHNIK
Ohmann

Im Wiehagen 15 - Tel. (05241) 67480
4830 Gütersloh 12

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

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

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

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

Nachliefermöglichkeiten: Hefte 1-13 sollten als Buch nachbezogen werden (s. Anzeige unten). Solange Vorrat reicht, können auch noch einzelne Hefte der Nos. 7-13 geliefert werden. Hefte 14-25 sind unbeschränkt nachlieferbar zu DM 7,80/Stück + DM 2,50 je Sendung.

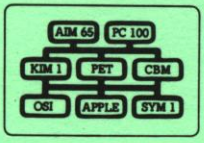
Private Besteller werden um Überweisung/Scheck (auch Auslandsschecks) zusammen mit der Bestellung gebeten. Konto Roland Löhr, Nr. 654 70-202 Postscheckamt Hamburg, BLZ 200 100 20.

Leser-Service des Herausgebers

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

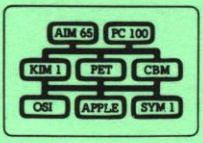
Thermokopf (Printerplatte) für AIM 65 und PC 100
mit Anleitung für den leichten Einbau. Auffrischung des Druckes **DM 28,--**

Das Buch 1-6 des 65.. MICRO MAG



230 Seiten, DM 26,--

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Programming & Interfacing the 6502 with Experiments
von Marvin L. de Jong, ca. 410 S., engl., viele Schaltungsvorschläge und die Programme für den Betrieb der Interfaces dazu, didaktisch gegliedert DM 65,--

Forth User's Guide
Handbuch der Firma Rockwell für ihr Fig-FORTH zum AIM 65, ca. 300 S., engl.. Erklärung des Befehlssatzes und der E/A, die im vorläufigen Handbuch noch fehlte. Geeignet auch für andere Betreiber eines Fig-FORTH DM 24,--

Vorstehende Preise sind inkl. MWSt, zuzüglich DM 2,50/Sendung + ggfs. NN + DM 1,70

FORTH-Programmierkurs am 8./9. Okt. 1982 in Ahrensburg bei Hamburg. Anmeldung und Prospekt beim Herausgeber.