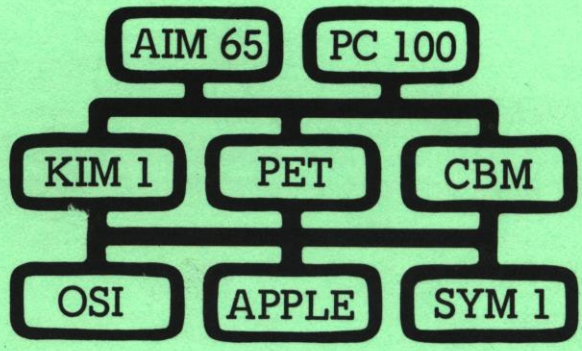
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 8,50 Nr. 25 Juni 1982



Inhaltsverzeichnis

FORTH !

Strings für FORTH 3
FORTH (2) 15
Mengen in FORTH 24
FORTH im Eigenbau 30
Der AIM 65 - PL/65-Compiler 41
BASIC - Compactor 48
Prozeßtechnik mit Microcomputern (4) 54
SWAP für BASIC 3 58
SHAKER 60
AIM Spezial (12) 62
Bücher 40, 53, 62
Aus der Branche - Produkte 62
Editorial 63

GWK

GESELLSCHAFT FÜR TECHNISCHE ELEKTRONIK mbH.
Asternstr. 2, D-5120 Herzogenrath, Tel. (0 24 06) 6 23 94 · Telex: 8 32 109 gwk d

SYSTEMEXPANSION FÜR AIM 65/ AIM 65/40

- **Floppy-Controller**
- **Video Interface**
- **A-D Converter**
- **D-A Converter**
- **Serielles und Parallel I/O**
- **Speichererweiterung RAM/EPROM**
- **Eprom-Programmer**
- **Prototyp Board**
- **Mother Board. Bus Buffer**
- **Power Supplies**
- **System Software**
  12K Extended BASIC

6809 COMPUTERSYSTEME AUF EUROPAKARTEN

- **CPU-Karte**
- **Floppy-Controller**
- **Winchester-Controller**
- **Schneller A-D Converter**
- **D-A Converter**
- **Serielles und Parallel I/O**
- **Grafik Controller**
- **Ram Board 32K**
- **Eprom Board 16/32K**
- **Bus Board**
- **Multiuser, Multitasking bei geeignetem Betriebssystem**

Strings für FORTH

Die nachfolgenden Vorschläge für die Einrichtung von Stringfunktionen dürften nicht nur für die Betreiber eines FORTH von Interesse sein, sondern auch für alle 6502-Assembler-Programmierer, die nach Lösungen für die eine oder andere Aufgabe suchen. Aus diesem Grunde wurde für solche Funktionen, die mit dem FORTH-Assembler formuliert wurden, auch der erzeugte Maschinencode gelistet. Im Vergleich mit dem Quelltext kann man hier nebenbei die Arbeitsweise eines strukturierten Assemblers studieren.

Stringfunktionen stellen für FORTH nach den Beobachtungen noch immer eine Option dar, die man entweder nachkaufen kann oder die man nach entsprechenden Vorschlägen selbst eintippt, um das Vokabular der Sprache zu erweitern. Vorschläge hierzu hat es u.a. in Dr. Dobb's Journal (1), im Forth Anwenderhandbuch für den AIM 65 (2) und in BYTE (3) gegeben. Dabei ist zu bemerken, daß die von Rockwell gemachten Vorschläge (2) weitgehend auf dem Aufsatz in Dr. Dobb's Jounal beruhen und daß beide unbefriedigend sind. Es geht ja z.B. nicht an, daß die VAL-Funktion den Programmablauf unterbricht, wenn sie nicht konvertiereren kann (in solchen Fällen muß sie eine 0 abliefern) oder daß die MID$ - Funktion zufälligen Müll abliefert, wenn sie versehentlich auf den hinteren Raum einer Stringvariablen angewendet wird, der für diese zwar reserviert ist aber z.Zt. nicht benutzt wird.

Die genannten Quellen konnten damit nur eine Grundlage für die nachfolgende Ausarbeitung sein, die ein Instrumentarium zur Verfügung stellt, das über die bequemen Möglichkeiten der meisten BASIC-Implementierungen auf Kleinrechnern hinausgehen dürfte. Der Autor ist dabei der Auffassung, daß eine Sprache, die nicht nur für das Steuern und Messen eingesetzt wird, Texte bequem beherrschen muß. Oder auch: Stringfunktionen erweitern die Einsatzmöglichkeiten einer Sprache erheblich.

Wir haben jetzt nicht nur String-Konstante, die kein überflüssiges Byte verbrauchen und die gegen ein unbeabsichtigtes Verändern geschützt sind, sondern auch eine ganze Reihe höherer Funktionen wie INSTRING (ermittelt ob ein Suchstring enthalten ist), Stringvergleich auf gleich, größer oder kleiner, sogar mit Berücksichtigung von Kleinbuchstaben ($COMP), Stringtausch ($SWAP), Ersetzen eines Instrings durch eine neue Zeichenkette (SEX), Beschneiden eines Strings vorne oder hinten, Absuchen eines Strings von vorne oder hinten auf das erste Vorkommen eines Suchzeichens (FLF bzw. FLB) usw.. Dazu die von BASIC her vertrauten Funktionen wie LEFT$, MID$, VAL, RIGHT$, STR$, ASC, CHR$ und LEN.

Mit den vorgeschlagenen Funktionen wird nicht nur eine bequeme Textbearbeitung ermöglicht, sondern z.B. auch ein schnelles Sortieren (Funktionen $COMP und $SWAP).

Im Prinzip hätte man die meisten Funktionen rein mit den Möglichkeiten der höheren Sprache FORTH definieren können. Der Autor ist jedoch der Ansicht, daß insbesondere Suchaufgaben wie in $COMP oder in INSTRING so schnell als irgend möglich ausführen sollten. Für diese wurde daher unter FORTH-Assembler programmiert.

Es war das Bemühen, die einzelnen Befehlszeilen möglichst weitgehend zu kommentieren, so daß an dieser Stelle nur noch allgemeinere Hinweise gegeben werden sollen:

Das zentrale Befehlswort für Strings ist die Definition STRING und die ihr ähnliche $CONST. Es sind sog. 'defining words', die einen neuen Datentyp schaffen. Sie werden z.B. in folgender Form gebraucht:

```
10 STRING NAME1
" DELTA" $CONST NAME2
```

Im ersten Fall wird eine Variable mit Namenskopf NAME1 angelegt, und zwar mit einer maximal nutzbaren Länge von 10 Bytes hex oder dezimal, je nachdem, welche Zahlenbasis gerade eingestellt ist.

Die Zuweisung einer Zeichenkette an eine solche Stringvariable geschieht z.B. durch:

" ADAM" NAME1 S! (s-store)

durch Übertragungen wie B$ NAME1 S! oder z.B. durch $SWAP.

Eine Stringvariable vom Typ STRING hat folgendes Speicherformat:

Namenskopf	Link-Adresse	Codefeld-Adresse	max. Länge	Speicherraum f.d. String
nfa	lfa	cfa	pfa	pfa+1

Mit dem Aufruf einer Stringvariablen gelangt nun a) eine Adresse auf den Datenstack, die der Beginnadresse des eigentlichen Strings entspricht (Parameterfeldadresse+1) und b) die zur Zeit benutzte aktuelle Länge LEN (diese wird von der Funktion SRCH ermittelt). Strings werden nach hinten immer mit einer 00 begrenzt, so daß SRCH die Bytes bis dahin zählt.

Die Ablage von Startadresse und Länge (ADDR LEN in den Listings) entspricht den Voraussetzungen der Ausgabefunktion TYPE, so daß wir z.B. einfach sagen können NAME1 TYPE. Adresse und Länge sind auch Berechnungsbasis für die meisten der folgenden Funktionen.

Insbesondere gibt ADDR-1 das Byte an, in dem die maximale Länge des Strings verankert ist. Alle in einen String schreibenden Funktionen müssen prüfen, daß die maximale Länge nicht überschritten wird, weil sonst Nachbarzellen beschädigt würden. Stringkonstanten ($CONST) enthalten nun das Attribut 0 für die maximale Länge. Sie sind damit gegen Überschreiben geschützt, können gleichwohl immer ihren Inhalt abgeben.

Die in den nachfolgenden Listings am häufigsten benutzten Befehlsworte betreffen den Datenstack, wie DUP, OVER, 3 PICK, SWAP, DROP, ROT usw.. Sie dürften den Lesern im wesentlichen vertraut sein. Seltener sind Transfers vom Datenstack zum Hardware und umgekehrt mit >R und R>. Diese Befehle müssen je Zeile oder neuem Befehlswort paarweise benutzt werden, weil die Maschine sonst abstürzt. - Um ein lästiges Umordnen der Operanden zu vermeiden, wurde in einigen Fällen auch das PAD benutzt, eine freie Speicherzone im RAM oberhalb des Vokabulars.

Zum Verständnis für die Assemblerprogrammierung: Auch der Assembler verlangt die Eingabe der Befehlsworte in umgekehrt polnischer Notation. In der zero page gibt es die sog. N–Area von 9 Bytes, sie erstreckt sich von symbolisch N-1 bis N+7. Beim AIM 65 liegt sie ab hex 97. Der Befehl SETUP JSR, befördert soviele Byte-Paare (16 Bit) in die N-Area, wie der derzeit im Akku übergebene Parameter fordert (also ein Wert zwischen 1 und 4 Paaren). Dabei landet das oberste Item vom Datenstack bei N, das nächste bei N+2 usw.. Der Datenstackpointer wird entsprechend mit den Entnahmen berichtigt. Umgekehrt kann man mit PUSH0A den Inhalt des Akkus als ein mit Null aufgefülltes Wort beim Verlassen der eigenen Routine auf den Datenstack transportieren. Auch mit TOP ist eine Adressierung des Datenstacks möglich. Solche Zellen in der zero page eignen für die Ablage von Pointeradressen und von Zählern, wie in den Listings ausgenutzt. Das Wort ')Y' im Assembler bewirkt die indirekte Pointeradressierung mit Indexregister Y in der zero page.

Literaturhinweise: (1) Ralph Deane, A Proposal on Strings for Forth, Dr. Dobb's Jounal Nov/Dec 1980, S. 40 ff., (2) Rockwell, Forth user's manual AIM 65, Document No. 29650N72, June 1981, Anhang I, (3) Kim Harris, FORTH Extensibility, BYTE, August 1980.

```
( FORTH STRINGFUNKTIONEN)
( COPYRIGHT BY ROLAND LOEHR 1982)
( V10.05.82)
( REC. FOR22)

FORGET TASK ! TASK !
HEX
DECIMAL
```

```
: SRCH
( ADDR -- ADDR LEN)
( BESTIMME AKTUELLE LAENGE DES STRINGS)
   DUP
   BEGIN
    DUP C$ SWAP 1+ SWAP 0=
   END
   SWAP - 1- ;

: STRING
( DEFINING WORD FOR STRING-TYPE)
( DEFINE: COUNT 'STRING' NAME)
( COUNT = MAXIMAL LENGTH OF STRING)
   <BUILDS
    ABS 255 MIN 1 MAX DUP
    ( FILTER FUER LAENGE)
    C, 0 DO 32 C,
        LOOP ( FUELLE MIT BLANKS)
    0 C, ( TERMINATOR ANHAENGEN)
   DOES>
   1+ DUP SRCH ; ( LAENGENPARAMETER UEBERGEBEN)

0 VARIABLE IB 255 ALLOT
( EINGABEPUFFER FUER STRINGS)

: (") ( USED BY ")
   R COUNT DUP 1+ R> + >R ;

: "
( WORD TO DEFINE A STRING IN IB)
   34 STATE $
   IF
    COMPILE (") WORD
    HERE C$ 1+ ALLOT
   ELSE WORD HERE COUNT
    IB SWAP ROT OVER IB
    SWAP 1+ CMOVE 2DUP
    + 0 SWAP C!
   THEN ;

IMMEDIATE
HEX

: VAL
( ADDR LEN -- D)
( RETURNS A DOUBLE PRECISION NUMBER)
( FROM STRING ON THE DATASTACK)
DROP 0 SWAP 0 SWAP
DUP C$ 2D = DUP >R ( SAVE SIGN )
IF
ELSE 1-
THEN (NUMBER) DUP DUP C$ 2E =

IF 1+ >R (NUMBER)
ELSE >R
THEN R> - DPL C! R>

IF DNEGATE THEN ;

DECIMAL
```

```
: STR$
( D -- STRING)
( CONVERTS DOUBLE PRECISION NUMBER TO STRING)
SWAP OVER DABS
<# DPL @ 0= NOT
IF
 BEGIN DPL @ 0= NOT
 WHILE # DPL @ 1- DPL !
 REPEAT
  46 HOLD
THEN
#S SIGN #> ;

: MLEN
( ADR LEN -- LEN)
( RETURNS MAX LENGTH OF A STRING)
   DROP 1- C@ ;

HEX

: $CONST
( DEFINING WORD TO CREATE A STRING-CONSTANT)
( OF FIXED LENGTH, THAT CANNOT BE DESTROYED)

<BUILDS 0 C, ( LEN=0 )
( STRING-KONSTANTE, GEGEN UEBERSCHREIBEN GESCHUETZT)
SWAP DROP ( COUNT LEFT )
0 DO IB I + C@ C, ( TRANSPORT TO HERE )
  LOOP 0 C, ( APPEND ZERO )
DOES>

1+ DUP SRCH ;

: S!
( ADDR LEN ADDR LEN -- )
( SCHREIBE IN EINEN STRING, AUSSER STRING-KONSTANTE)
DROP ( LEN ) DUP ( ADDR. OF DEST )
CFA 1- @ ( WHERE FROM ? )
HEX 22 - ' $CONST -
( FROM $CONST ? )
0= IF 2DROP DROP ." DONE NOTHING"
   ELSE DUP 1- C@
   ROT MIN 0 MAX 2DUP
   + 0 SWAP C! CMOVE THEN ;

: $SWAP
( ADDR LEN ADDR LEN -- )
( VERTAUSCHUNG ZWEIER STRINGVARIABLER)
2DUP >R >R
IB SWAP CMOVE ( TO IB )
2DUP R> R S! ( FILL SECOND STRING )
DROP DUP 1- C@
R> MIN IB ROT ROT 1+ CMOVE ( IB TO FIRST STRING ) ;

: LEN
( ADDR LEN -- LEN)
( LAENGENATTRIBUT DES STRINS UEBERGEBEN)
( AKTUELL BENUTZTE LAENGE)
   SWAP DROP ;
```

```
: MID$
( N M ADDR LEN -- ADDR LEN)
( STARTING AT POSITION N FOR M BYTES OF STRING)
DUP ( LEN) 5 PICK ( ABSTELLE)
- DUP ( LAENGENDIFFERENZ)
0=
IF 2DROP ( DIF, LEN)
 ROT + 1-
 SWAP 0 MAX 1 MIN
ELSE 0< NOT
 IF ( LAENGE GROESSER) DROP ( LEN)
  ROT + 1- DUP SRCH ( REST-LEN)
  ROT MIN
 ELSE ROT ROT 2DROP 0
 THEN THEN ;

: LEFT$
( N ADDR LEN -- ADDR LEN)
DROP SWAP ;

: RIGHT$
( N ADDR LEN -- ADDR LEN)
( RETURNS THE N RIGTHMOST CHARS OF A STRING)
DUP 4 PICK MIN - + SWAP ;

: S+
( ADDR LEN ADDR LEN -- )
( STRINGS VERBINDEN, WENN LAENGE UND TYP ES ZULASEN)
4 PICK 1- C$ ( MAX LEN ) DUP
0= IF 2DROP 2DROP DROP
   ELSE
    OVER - ( DO NOT OVERRIDE STRING )
    4 PICK -
    DUP 0<
         IF +
         ELSE MIN
         THEN
    >R ROT ROT + R>
    2DUP + >R CMOVE
    0 R> C!
THEN
;

: HACK
( N ADDR LEN -- )
( N BYTES AM ENDE DES STRINGS ABHACKEN)
3 PICK 1- C$ ( SEE MLEN )
0= IF 2DROP DROP
   ( WAR STRINGKONSTANTE)
   ELSE 2DUP -
0< IF 2DROP 0 0
THEN
- + 0 SWAP C!
THEN ;


: CUT ( ADR LEN AB --- )
( AB STELLE BESCHNEIDEN, NICHT JEDOCH KONSTANTE)
( VERKUERZT STRING)
```

```
SWAP DROP OVER 1- C$ ( MLEN )
DUP
0= IF 2DROP DROP
   ( WAR KONSTANTE)
   ELSE OVER -
    0< IF 2DROP
        ELSE + 0 SWAP C!
THEN THEN ;

: SUB
( ADDR LEN # M -- )
( SUBSTITUTE M BYTES STARTING AT BYTE #)
MID$
   ROT MIN 1 MAX CMOVE ;

: S=
( ADDR LEN ADDR LEN -- FLAG)
( STRINGVERGLEICH OB GLEICH)
( FLAG=1 ON EQUAL, =0 IF NOT EQUAL)
   ROT OVER =
   IF
    1 SWAP 0
    DO
    DROP OVER C$ OVER C$ =
    IF
     1+ SWAP 1+ SWAP 1
    ELSE
     0 LEAVE
    THEN
    LOOP
   ELSE
    DROP 0
   THEN
   SWAP DROP SWAP DROP ;

: FLF ( FIND LOCATION FORWARD OF CHARACTER IN STRING)
 ( ADR LEN CHAR -- ADDR 1 FLAG OR 0-FLAG ONLY)
 0 PAD C! ( FLAG=0)
 >R ( LEN)
 BEGIN DUP 0= NOT
 WHILE OVER C$ R =
  IF DROP 0 1 PAD C!
  ELSE 1- SWAP 1+ SWAP ( LEN-1, ADDR+1, CHAR FOUND)
  THEN
 REPEAT
R> 2DROP ( LEN)
PAD C$ ( FLAG) DUP 0=
 IF SWAP DROP ( FALSE-FLAG ONLY)
 THEN
;

: FLB ( FIND LOCATION BACKWARD IN STRING)
 ( ADDR LEN CHAR -- ADDR/FLAG=1 OR FLAG=0 ONLY)
 0 PAD C! ( FLAG) >R ( LEN)
 DUP >R + R> ( ENDADDRESS LEN)
```

```
 BEGIN DUP 0= NOT
  WHILE OVER C$ R = ( GET & COMPARE CHARS)
    IF DROP 0 1 PAD C! ( FLAG=1)
    ELSE 1- SWAP 1- SWAP ( LEN-1, ADDR-1)
    THEN
 REPEAT
 R> 2DROP PAD C$ DUP 0= ( GET FLAG)
  IF SWAP DROP ( FALSE FLAG ONLY)
  THEN
;

: NSERT ( XCHANGE CHARACTERS)
( ADR LEN OLDCHAR NEWCHAR -- ) ( NO FLAGS)
4 PICK 1- C$ 0=
IF 2DROP 2DROP
ELSE
 >R FLF ( SCAN)
  IF R> SWAP C!
  ELSE R> 2DROP
  THEN
THEN ;


CODE (MOVE)
( FROM-ADDR TO-ADDR LEN)
( CALCULATES POINTERS IN N-AREA FOR MOVE FROM BACK)
3 # LDA,
SETUP JSR, ( TRANSPORT OF PARMS TO ZEROPAGE)
CLC,
N 5 + LDA,
N 1 + ADC,
N 5 + STA, ( PAGE ADDRESSES ADDED TO POINTER FROM)
CLC, ( NOW POINTER TO)
N 3 + LDA,
N 1 + ADC,
N 3 + STA,
RTS,
END-CODE

CODE ZMOVE
 ( FROM-ADDR TO-ADDR COUNT -- )
 ( MOVE BYTES STARTING AT THE REAR END & APPEND A ZERO)
SEC, ( FLAG: APPEND A ZERO)
PHP,
' (MOVE) JSR,
PLP, ( GET CARRY STATUS BACK)
N LDA, ( CHECK ON LEN ZERO)
N 1 + ORA,
0= NOT
IF,
 N LDY,
 0 # LDA,
 CS
  IF, N 2 + )Y STA,
  THEN,
 N STA, ( LOWBYTE OF LEN =0 NOW)
```

Erzeugter Maschinen-Code

```
0916 38 SEC
0917 08 PHP
0918 20 JSR 08F8
091B 28 PLP
091C A5 LDA 97
091E 05 ORA 98
0920 F0 BEQ 094B
0922 A4 LDY 97
0924 A9 LDA #00
0926 90 BCC 092A
0928 91 STA (99),Y
092A 85 STA 97
```

```
 BEGIN,
  DEY,
  N 4 + )Y LDA,
  N 2 + )Y STA,
  TYA, ( CHECK Y)
 0= UNTIL,

 BEGIN,
  N LDA,
  N 1 + ORA,
  0= NOT
  WHILE,
   N 5 + DEC, ( PAGE-ADDR FROM)
   N 3 + DEC, ( PAGE-ADDR TO)
   N 1 + DEC, ( COUNT HIGH)
   BEGIN,
    DEY,
    N 4 + )Y LDA, ( MOVE OF FULL PAGES)
    N 2 + )Y STA,
    TYA, ( CHECK Y)
   0= UNTIL,
  REPEAT,
THEN,
NEXT JMP,
END-CODE

CODE <MOVE
 ( FROM-ADDR TO-ADDR COUNT -- )
 ( MOVE BYTES STARTING AT THE REAR END)
 ( NEEDED IN OVERLAPPING FIELDS)
 CLC, ( FLAG NO ZERO APPEND)
 ' ZMOVE 1 + JMP,
  ( JUMP JUST AFTER CARRY SET)
END-CODE



CODE OMOVE
( ADDR ADDR LEN -- )
( FROM TO MOVE RIGHT, APPEND A ZERO)
3 # LDA,
SETUP JSR, ( GET PARMS TO N-AREA)
0 # LDY,
BEGIN,
 N 4 + )Y LDA,
 N 2 + )Y STA,
 INY,
 N CPY, ( ARE WE AT LENGTH)
0= UNTIL,
 0 # LDA,
 N 2 + )Y STA,
 NEXT JMP,
END-CODE

: CLIP ( AT FRONT OF STRING)
( N ADDR LEN -- )
( CLIP N BYTES FROM START OF STRING, NOT IN CONSTANT)
3 PICK 1- C$
```

Erzeugter Maschinen-Code

```
092C 88 DEY
092D B1 LDA (9B),Y
092F 91 STA (99),Y
0931 98 TYA
0932 D0 BNE 092C
0934 A5 LDA 97
0936 05 ORA 98
0938 F0 BEQ 094B
093A C6 DEC 9C
093C C6 DEC 9A
093E C6 DEC 98
0940 88 DEY
0941 B1 LDA (9B),Y
0943 91 STA (99),Y
0945 98 TYA
0946 D0 BNE 0940
0948 4C JMP 0934
094B 4C JMP B05A

0958 18 CLC
0959 4C JMP 0917

0966 A9 LDA #03
0968 20 JSR B099
096B A0 LDY #00
096D B1 LDA (9B),Y
096F 91 STA (99),Y
0971 C8 INY
0972 C4 CPY 97
0974 D0 BNE 096D
0976 A9 LDA #00
0978 91 STA (99),Y
097A 4C JMP B05A
```

```
0= IF 2DROP DROP
   ( *CONSTANT)
    ELSE
     2DUP ( LEN & CLIP)
     <
     IF 2 DROP 0 SWAP C!
     ELSE SWAP DROP >R DUP R +
 SWAP
 R> 1+ OMOVE
THEN
THEN
;

CODE INSTR
( ADDR LEN ADDR LEN            -- ADDR 1 OR 0-FLAG)
( INSTRING FUNCTION)
( ADR & LEN OF STRING TO BE SEARCHED)
( ADR & LEN OF STRING TO BE MATCHED)
( RETURNS TRUE FLAG & LOCATION ON MATCH)
( ON NO MATCH ONLY RETURNS FALSE FLAG)
( QUICK EXECUTING IN MACHINE CODE)
4 # LDA,
SETUP JSR, ( TRANSPORT PARAMETERS FROM DATASTACK TO N-AREA)

BEGIN,                                                  | 09D2 A9 LDA #04
N 4 + LDY,                                              | 09D4 20 JSR B099
N CPY, ( LAENGENVERGLEICH DER STRINGS)                  | 09D7 A4 LDY 9B
CS WHILE,                                               | 09D9 C4 CPY 97
 0 # LDY,                                               | 09DB 90 BCC 0A0A
 BEGIN,                                                 | 09DD A0 LDY #00
  N 6 + )Y LDA,                                         | 09DF B1 LDA (9D),Y
  N 2 + )Y CMP,                                         | 09E1 D1 CMP (99),Y
  0= WHILE, ( VERGL. AUFSTEIGEND, SOLANGE GLEICHHEIT)
   INY,                                                 | 09E3 D0 BNE 09FF
   N CPY,                                               | 09E5 C8 INY
   0= NOT IF,                                           | 09E6 C4 CPY 97
   ELSE,                                                | 09E8 F0 BEQ 09ED
    ( GLEICHHEIT AUF GANZER LAENGE)                     | 09EA 4C JMP 09FC
    ( SCHREIBE TRUE-FLAG UND BEGINN DES INSTRINGS)
    ( AUF DEN DATENSTACK, ADRESSIERT MIT TOP)
    DEX,
    N 7 + LDA,                                          | 09ED CA DEX
    TOP STA,                                            | 09EE A5 LDA 9E
    N 6 + LDA,                                          | 09F0 95 STA 00,X
    DEX,                                                | 09F2 A5 LDA 9D
    TOP STA,                                            | 09F4 CA DEX
    1 # LDA,                                            | 09F5 95 STA 00,X
    PUSHOA JMP,                                         | 09F7 A9 LDA #01
    ( RETURN TO FORTH)                                  | 09F9 4C JMP B517
   THEN,                                                | 09FC 4C JMP 09DF
    REPEAT,
( LAENGE UND POINTER IN DER N-AREA VERMINDERN/ERHOEHEN)
 N 4 + DEC,                                             | 09FF C6 DEC 9B
 N 6 + INC,                                             | 0A01 E6 INC 9D
 0= IF, N 7 + INC,                                      | 0A03 D0 BNE 0A07
   THEN,                                                | 0A05 E6 INC 9E
REPEAT,                                                 | 0A07 4C JMP 09D7
( ERZEUGE FALSE-FLAG ON NO MATCH)                       | 0A0A A9 LDA #00
0 # LDA,                                                | 0A0C 4C JMP B517
PUSHOA JMP,
END-CODE
```

```
: SEX
( ADR LEN ADR LEN ADR LEN -- )
( STRING-EXCHANGE)
( ADR & LEN ZU BEARBEITENDER STRING)
( DITO SUCHSTRING)
( DITO REPLACEMENT-STRING ON TOP)

PAD 2+ ! ( LEN REPL)
PAD ! ( ADDR REPL)
DUP PAD 4 + ! ( LEN SUCHSTR)
4 PICK 1- DUP C$ + PAD 6 + ! ( LAST BYTE ZIELSTR)
INSTR ( SEE IF STRING CONTAINED WITHIN)
0= IF ( NOT FOUND)
   ELSE DUP PAD 4 + $ + DUP PAD 8 + !
   ( ADDR OF RESTSTRING)
   SRCH >R ( RESTLEN ZIELSTRIN )
   DUP PAD 2+ $ + ( DESTINATION ADDRESS)
   DUP R + ( NEW ENDADDR)
   1-
   PAD 6 + $ ( END OF STRING ADDR)
   > IF ( NO SPACE LEFT) 2DROP R> DROP ( DO NOTHING)
     ELSE DUP PAD 8 + $ ( ADDR OF RESTSTRING)
      < IF PAD 8 + $ SWAP R> 1+ CMOVE ( TO THE FRONT)
        ELSE PAD 8 + $ SWAP R> ZMOVE ( SHIFT RIGHT)
        THEN
     PAD $ SWAP PAD 2+ $ CMOVE ( INSERT REPLACEMENT)
  THEN
THEN
;

CODE FILTER
( FILTER FUER KLEINBUCHSTABEN EINSCHL. DEUTSCHEN ZEICHEN)
( WERDEN AUF GROSSBUCHSTABEN UMGESTELLT)
( 'SZ' = HEX 7E  WIRD ALS 'S' BEHANDELT)

7E # CMP,
0= IF, 53 # LDA, ( WAR SZ)
ELSE,
 61 # CMP,
 CS IF, ( KANN KLEINBUCHSTABE SEIN)
 7F # CMP,
  CS NOT
   IF, ( IST KLEINBUCHSTABE)
   SEC,
   20 # SBC, ( WIRD GROSSBUCHSTABE)
   THEN,
THEN,
THEN,
RTS,
END-CODE
```

```
0AE5 C9 CMP #7E
0AE7 D0 BNE 0AEE
0AE9 A9 LDA #53
0AEB 4C JMP 0AF9
0AEE C9 CMP #61
0AF0 90 BCC 0AF9
0AF2 C9 CMP #7F
0AF4 B0 BCS 0AF9
0AF6 38 SEC
0AF7 E9 SBC #20
0AF9 60 RTS
```

```
CODE $COMP
( ADDR LEN ADDR LEN -- FLAG)
( COMPARE 2 STRINGS ON = < OR > )
( FLAG=0 ON EQUAL, =1 ON STR1 > STR2, = FF ON STR1 < STR2)
4 # LDA,
SETUP JSR, ( PARMS FROM STACK TO N-AREA)
0 # LDY,
N 1 - STY, ( FLAG=0)
```

```
 BEGIN,
  N 4 + LDA,  ( BOTH LENGTHES MAY NOT BE ZERO)
  0= IF,
     ELSE, N LDA,
     THEN,
  XSAVE STA,  ( HAVE A FLAG)
  N 1 - LDA,  ( GET STATUS FLAG)
  0= NOT  ( DO NOT PROCEED IF WE HAVE A STATUS)
     IF, 0 # LDA,
     XSAVE STA,  ( OVERRIDE FLAG)
     THEN,
 XSAVE LDA,  ( GET FLAG)
 0= NOT

 WHILE,  ( COMPARE STRINGS)
  ( MAIN LOOP)
  N 2 + )Y LDA,  ( GET CHAR FRON 2ND STRING)
  ' FILTER JSR,  ( CONVT CHAR)
  N 1 + STA,  ( SAVE RESULT)
  N 6 + )Y LDA,  ( CHAR FROM 1RST STRING)
  ' FILTER JSR,  ( CONVT)
  SEC,
  N 1 + SBC,  ( COMPARE TO 2ND CHAR)
  0= IF,  ( THEY ARE EQUAL, WE MAY PROCEED)
       INY,
       N DEC,  ( COUNT)
       N 4 + DEC,  ( COUNT)
      ELSE,
       CS IF, N 1 - INC,  ( FLAG STR1 > STR2)
          ELSE, N 1 - DEC,  ( FLAG STR1 < STR2
          THEN,
      THEN,
 REPEAT,

N 1 - LDA,  ( GET FLAG)
0= IF,  ( NOW CHECK LENGTHS)
   N LDA,  ( LEN1)
   N 4 + ORA,  ( LEN2)
   0= IF,  ( EQUAL DON'T ALTER FLAG)
       ELSE,
        N LDA,  ( LEN1)
        0= IF, N 1 - INC,  ( STR2 IS LONGER)
           ELSE, N 4 + LDA,
            0= IF, N 1 - DEC,  ( STR1 IS LONGER)
           THEN,
       THEN,
     THEN,
THEN,
N 1 - LDA,  ( GET FINAL FLAG)
PUSHOA JMP,
END-CODE

: CHR$  ( PUT A HEXBYTE INTO STRING)
( BYTE ADDR LEN -- )
DROP DUP 1- C$
0= IF 2DROP ( $CONSTANT )
   ELSE DUP >R  ( SAVE ADDR)
    C!  ( BYTE) R> 1+ 0 C!  ( APPEND A ZERO)
  THEN
;
```

```
0B04 A9 LDA #04
0B06 20 JSR B099
0B09 A0 LDY #00
0B0B 84 STY 96
0B0D A5 LDA 9B
0B0F D0 BNE 0B14
0B11 4C JMP 0B16
0B14 A5 LDA 97
0B16 85 STA A6
0B18 A5 LDA 96
0B1A F0 BEQ 0B20
0B1C A9 LDA #00
0B1E 85 STA A6
0B20 A5 LDA A6
0B22 F0 BEQ 0B49
0B24 B1 LDA (99),Y
0B26 20 JSR 0AE5
0B29 85 STA 98
0B2B B1 LDA (9D),Y
0B2D 20 JSR 0AE5
0B30 38 SEC
0B31 E5 SBC 98
0B33 D0 BNE 0B3D
0B35 C8 INY
0B36 C6 DEC 97
0B38 C6 DEC 9B
0B3A 4C JMP 0B46
0B3D 90 BCC 0B44
0B3F E6 INC 96
0B41 4C JMP 0B46
0B44 C6 DEC 96
0B46 4C JMP 0B0D
0B49 A5 LDA 96
0B4B D0 BNE 0B65
0B4D A5 LDA 97
0B4F 05 ORA 9B
0B51 D0 BNE 0B56
0B53 4C JMP 0B65
0B56 A5 LDA 97
0B58 D0 BNE 0B5F
0B5A E6 INC 96
0B5C 4C JMP 0B65
0B5F A5 LDA 9B
0B61 D0 BNE 0B65
0B63 C6 DEC 96
0B65 A5 LDA 96
0B67 4C JMP B517
```

```
: ASC ( RETURNS NUMERICAL VALUE OF 1RST CHAR. IN)
( A STRING. ERROR INDICATED ON 00 VALUE)
( ADDR LEN -- N) ( OR) ( ADDR LEN --/ERROR)
 DROP C$ DUP
 0= IF DROP ." ILLEGAL QUANTITY" CR
    THEN
;

: ERA ( ERASE A STRING TO "")
( ADDR LEN -- )
DROP 0 SWAP C! ;

FINIS
```

```
VLIST
  BCC ERA
  B9F ASC
  B73 CHR$
  B04 $COMP
  AE5 FILTER
  A17 SEX
  9D2 INSTR
  986 CLIP
  966 OMOVE
  958 <MOVE
  916 ZMOVE
  8F8 (MOVE)
  8BE NSERT
  85E FLB
  808 FLF
  7B6 S=
  7A1 SUB
  765 CUT
  72B HACK
  6D2 S+
  6BA RIGHT$
  6A9 LEFT$
  652 MID$
  643 LEN
  60F $SWAP
  5B5 S!
  584 $CONST
  571 MLEN
  52B STR$
  4D2 VAL
  48B "
  475 (")
  36C IB
  339 STRING
  314 SRCH
  309 TASK
```

Für die Anwendung und Wirkung der Stringfunktionen hier einige Beispiele. Dabei wird meistens davon ausgegangen, daß die Variablen vor der Befehlsausführung folgenden (String-) Inhalt haben:

A$ = 12345 B$ = BERTA D$ = DELTA

Anweisung	Man erhält:	
10 STRING A$	VARIABLE A$ FÜR 10 BYTES	
" 12345" A$ S!	A$= 12345	
" DELTA" $CONST D$	KONSTANTE MIT INHALT DELTA	
A$ B$ S!	A$= 12345	B$= 12345
A$ B$ $SWAP	A$= BERTA	B$= 12345
A$ D$ S!	D$= DELTA	$KONSTANTE!
B$ " ERTA" A$ SEX	B$= B12345	
B$ ASC	42	
B$ ERA	B$=	(LEER)
A$ 3 CLIP	A$= 45	
A$ 4 HACK	A$= 1	
A$ " 34" INSTR	ADRESSE FLAG=1	
A$ " EMIL" INSTR	0	
A$ B$ $COMP	FF	(KLEINER)
B$ A$ $COMP	1	(GRÖSSER)
B$ " BERTA" $COMP	0	(GLEICH)
A$ 32 41 NSERT	A$= 1A345	
D$ A$ S! A$ D$ S=	1	
A$ 33 FLF	ADDR 1	
A$ 3 LEFT$ A$ TYPE	123	
A$ VAL D.	12345	
129.345 STR$ A$ S!	A$= 129.345	
333. STR$ A$ S!	A$= 333	
3 2 A$ MID$ TYPE	34	

R.L. #

FORTH (2)

3. Grundstrukturen

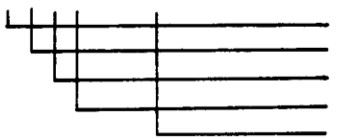
3.1 Einfache Befehle im Tischrechner-Modus

Nach den Initialisieren des FORTH (Einsetzen von ROMs und Drücken der Taste (5) beim AIM, bzw. Laden und Starten mit einem SYS-Befehl) meldet es sich z.B. mit

AIM 65 FORTH V1.3.

Das System ist jetzt im Tischrechner-Modus und wartet auf Befehle von der Tastatur. Nach einem solchen 'COLD-Start' werden Zahlen als Dezimalzahlen interpretiert. Wir können zum Beispiel eingeben:

155 6 + . (RETURN-Taste)



Die Eingaben sind dabei:

1. Operand
2. Operand
Additions-Befehl
Ausgabe-Befehl (Punkt)
Auslösung der Befehlszeile.

Das System antwortet 161 OK

Ausgabe des Ergebnisses
Prompt oder Cursor, daß alles gut ging, neue Eingabebereitschaft.

Hier haben wir bereits eine typische Befehlszeile in umgekehrter polnischer Notation:Die Operanden (die Zahlen 155 und 6) werden als erste genannt und mit ihrer Nennung auf den Stack gebracht. Erst danach darf die + -Operation aufgerufen werden, weil sie 2 bereits vorhandene Operanden voraussetzt. Mit Abarbeitung des Befehlswortes '+' werden diese beiden Operanden vom Datenstack entfernt, stattdessen wird deren Summe dort hinterlassen. In symbolischer Schreibweise sieht man links der Striche, was die + -Operation vor der Ausführung auf dem Stack erwartet (die Ganzzahlen n1 und n2) und rechts davon, was sie hinterläßt:

(n1 n2 — sum)

Der Punkt ist ebenfalls ein Befehlswort des FORTH: Drucke die zuoberst auf dem Stack befindliche Zahl aus, symbolisch: (n —). Auch dieser Befehl 'verbraucht' ein Element vom Datenstack.

Das Betätigen der Return-Taste löst die Abarbeitung der gesamten Befehlszeile aus. Das System addiert jetzt die beiden Zahlen zu 161 und druckt die Summe aus und meldet sich dann mit 'OK' oder blinkendem Cursor zurück.

Wir merken uns: Der Zwischenraum (SPACE) ist in FORTH ein lebenswichtiges Element, er trennt Operanden untereinander und gegen Befehlsworte. Auch Befehlsworte werden durch Space gegeneinander begrenzt.

Es hat also eine verschiedene Wirkung, ob man schreibt 155 oder 1 5 5. Im ersten Fall wird eine Zahl auf den Stack gebracht, im zweiten Fall drei Zahlen. Auch ein Befehlswort, das aus mehreren Zeichen besteht, darf nicht durch eingestreute Spaces verstümmelt werden. So gibt es z.B. den Befehl +! (plus-store) aber auch die Einzelbefehle + und ! mit einer ganz anderen Wirkung. Wichtig ist also die Reihenfolge der Operanden und Befehle und deren Trennung durch Space untereinander.

Jedes Abweichen von dieser Regel führt für die obige Rechenaufgabe zu unerwarteten Ergebnissen. Man probiere ruhig einmal ' 155 + 6 = .

Für die obige Berechnung erhalten wir das gleiche Ergebnis, wenn wir die Zeile in Einzelschritte/Einzelzeilen auflösen. Wir erhalten dann folgende Abflaufgrafik:

Befehl	symbolische Befehlswirkung	Parameter auf dem Datenstack
155 (RET)	(— n1)	n1
6 (RET)	(— n2)	n2 n1
+ (RET)	(n1 n2 — sum)	sum
. (RET)	(n —)	

Das Betätigen der RETURN-Taste haben wir in dieser Grafik mit (RET) symbolisiert. Künftig gehen wir davon aus, daß der Leser seine Zeilen auch ohne diesen besonderen Hinweis abschließt und zur Ausführung bringt. Auch das 'OK' des Systems werden wir nur noch dann abdrucken, wenn dies zur Verdeutlichung nötig ist.
Die Auflösung unserer ursprünglichen Befehlszeile ' 155 6 + . ' in vier Einzelzeilen zeigt: *Eine Befehlszeile darf beliebig viele aneinandergereihte und durch Space getrennte Operanden und Befehlsworte enthalten (bis zur maximalen Zeilenbreite). Erst das (RET) löst die Abarbeitung aus. In der noch nicht abgeschlossenen (offenen) Zeile können Eingabefehler noch mit der DELETE-Taste getilgt werden.*

Der Leser probiere jetzt bitte auch die weiteren Grundrechenarten aus:

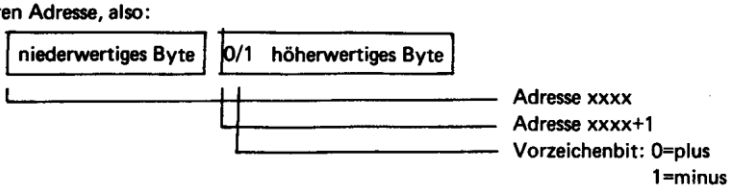
Befehlswort	symbol. Wirkung	Operation
–	(n1 n2 — diff)	Subtraktion n1 – n2
*	(n1 n2 — prod)	Multiplikation n1 * n2
/	(n1 n2 — quot)	Division n1 : n2

Beispiele:

```
9 4 - . 5 OK
3 7 * . 21 OK
36 9 / . 4 OK
37 9 / . 4 OK
```

An der letzten Division sehen wir, daß die bisher besprochenen Grundrechenarten auf Ganzzahlen im Wertebereich von –32768 bis +32767 (Integers) wirken und daß sie Ganzzahlen im Ergebnis abliefern. Weiterhin: Der Divisionsrest ging im letzten Beispiel verloren.

Maschinenintern sind solche Integers in 16 Bit (2 Byte) in einer sog. 'Zelle' gespeichert. 15 Bit repräsentieren dabei die Zahl in binärer Darstellung und das 16 Bit das Vorzeichen. Das Byte mit dem höherwertigen Zahlenanteil und mit dem Vorzeichen steht speicherintern an der höheren Adresse, also:



Mit Forth kann man sich in verschiedenen Zahlensystemen bewegen, z.B. zur Zahlenbasis 2, 8, 10, 16 oder auch z.B. 36. Die interne Zahlendarstellung und -verarbeitung bleibt dabei immer binär, die 'Kulisse' der eingestellten Zahlenbasis betrifft also nur die Interpretation der Eingaben und die Aufbereitung der Ausgaben.

DECIMAL und HEX sind Schlüsselworte, die diese Interpretation der Zahlen-E/A umschalten:

```
HEX A B + . 15 OK
HEX A B * DECIMAL . 110 OK
```

Das zweite Beispiel zeigt: man kann die Interpretationsbasis innerhalb einer Zeile beliebig umschalten. A und B werden als hexadezimale Eingaben betrachtet. Vor dem Drucken mit ' . ' wurde auf dezimale Ausgabe umgeschaltet. Eine einmal gewählte Zahlenbasis wirkt solange fort, bis eine andere gewählt wird. Wir verließen das letzte Beispiel im Dezimalmodus. Wenn wir jetzt anschließend eingeben

```
A B + .
```

so antwortet das System mit ' A ? ', weil 'A' kein Befehlswort ist und weil die Ziffer 'A' im Dezimalmodus verboten ist. Zahleneingaben werden also auf Zulässigkeit hinsichtlich der eingestellten Zahlenbasis überprüft. Ausnahme: Im binären Modus werden auch die häufig benutzten Zahlen 2 und 3 mit ihrem richtigen Wert akzeptiert:

```
2 BASE ! OK
1 1 + . 10 OK
2 1 + . 11 OK
3 1 + . 100 OK
```

Die Zeile ' 2 BASE ! ' ist mnemonisch zu lesen als ' 2 BASE STORE', speichere die 2 in jene symbolische Adresse, in der das System seine Interpretationsbasis speichert. Wir sind damit im binären Modus, den wir wahlfrei mit 'HEX' oder 'DECIMAL' wieder verlassen können, also zum Beispiel:

```
101 11 * DECIMAL . 15 OK
```

Dem Leser steht nach diesen ersten Befehlen eine Rechen- und Zahlenkonvertiermaschine für beliebige Zahlensysteme zur Verfügung, die er mit eigenen Beispielen zur Übung betreiben möge. Wenn er z.B. sagt

```
DECIMAL 36 BASE !
```

so steht ihm der Ziffernvorrat von 1 - 9 und von A - Z für Experimente zur Verfügung.

3.2 Vorschau auf weitere Befehlsgruppen

Als Sprache weist FORTH natürlich weitere Befehlsgruppen auf. Wir werden Rechen- und Formatierbefehle für doppelt genaue Zahlen (32 Bit) kennenlernen, Vergleichsbefehle, mit denen wir den Programmfluß in einer strukturierten Form z.B. mit BEGIN .. WHILE ... REPEAT oder IF .. ELSE .. THEN steuern können, Zählschleifen, ferner Befehle, die auf den Daten- und Hardwarestack wirken.

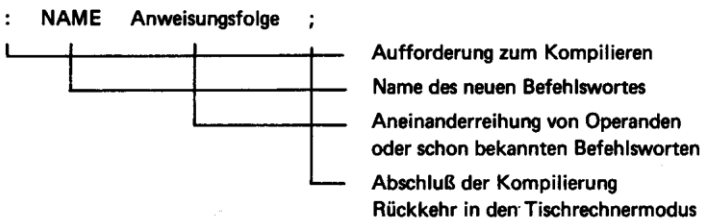
Für den Anfang sind Befehle wichtig, die die Formulierung eines Programmes gestatten und die die Kompilierfähigkeit des FORTH zeigen. Wir verbinden das zugleich mit den immer wichtigen Ein- und Ausgabebefehlen, denn eine Sprache, die keine Bedienerführungen schreiben könnte, wäre nicht viel mehr als ein kleiner Taschenrechner wert.

Nach dem ersten Kennenlernen werden die Befehlsgruppen systematisch vertieft und ergänzt um Fragen der Massenspeicherung, des Assembler-Gebrauches oder des Schaffens neuer Datentypen. Parallel laufend bringen wir Einzeldarstellungen z.B. für die Stringbearbeitung, für Datensets oder für das nützliche CASE-Statement.

3.3 Das Kompilieren neuer Befehlsworte - eines Programmes

Bei den ersten Versuchen im Tischrechner-Modus blieben die von uns eingetippten Befehle nicht in der Maschine gespeichert, sie wurden unmittelbar abgearbeitet und dann vergessen. Solche losen Befehlsketten kann man sicher auch im Text-Editor ablegen, von dort her einlesen und abarbeiten. Das ist jedoch noch nicht das eigentliche Programmieren. *FORTH gestattet es vielmehr, neue mit Namen versehene Befehlswortketten in die Liste der ausführbaren Befehle (dictionary) einzureihen. Mit dem Aufruf des Namens wird dann die Anweisungsfolge ausgeführt.*

Wir haben damit folgendes Format für die Kompilierung neuer FORTH-Befehle:



Dieses Format zeichnet sich durch den Doppelpunkt am Anfang (colon definition), der die Kompilierung einleitet und durch das Semikolon am Schluß aus, das die Komplierung beendet.

NAME ist die Zeichenkette, unter der das neue Befehlswort abgespeichert wird und mit dessen Aufruf es später zur Ausführung gebracht wird. Die bisher schon benutzten Befehlsworte wie z.B. + , − , * , / sind solche Namen im dictionary. Für den Namen dürfen beliebige Zeichen (Ziffern, Buchstaben, Sonderzeichen, nicht jedoch SPACE, RETURN) verwendet werden. Man vermeide jedoch nicht druckbare Zeichen und Überschneidungen mit schon vorhandenen Befehlsworten (Befehlswortvorrat lt. Handbuch).

Zulässig sind z.B also MAL2, 2MAL-2, P1, P*, SI, HOLE, SCHREIBE-PI usw..

Obwohl FORTH die persönliche Bequemlichkeit beim Eintippen sehr unterstützt, nehme man nicht zu kurze Bezeichner und wähle sie vor allem mnemonisch aussagefähig, also nicht einfach zu 'XYZ'.

Die Zeichenkette NAME darf bis zu 31 Zeichen enthalten. Das AIM-FORTH speichert sie alle im dictionary ab. Andere Implementierungen hacken den Namen durchaus nach dem dritten Zeichen ab, so daß durchaus die Frage der Verwechselbarkeit auftritt.

In der ANWEISUNGSFOLGE werden (getrennt durch Spaces) die benötigten und schon im dictionary enthaltenen Befehlsworte, Konstante, Variable und/oder Zahlen aufgeführt. die sog. forward references sind nicht zulässig. Sie werden vom System mit Ausgabe des beanstandeten Wortes und einem Fragezeichen quittiert. Man kann also nur auf bekannte Worte zurückgreifen.

Das Semikolon beendet die Kompilierung. Wenn alles gut ging, meldet sich das System mit 'OK' zurück. Das neue Befehlswort ist damit genauso Bestandteil des FORTH geworden, wie die schon früher im dictionary enthaltenen Befehle.

Programmieren mit FORTH heißt neue Befehlsworte zu schreiben, die in ihrer Summe das Programm bilden. Der Programmaufbau erfolgt dabei modular in möglichst kurzen und einfachen Anweisungsfolgen, die typisch etwa bis zu 10 Befehlsworte enthalten. Die nächste hierarchisch höhere Programmebene bedient sich dieser Module bis schließlich ein letztes Befehlswort das Hauptprogramm darstellt. Wie auch in anderen strukturierten Sprachen steht dieses Hauptprogramm ganz am Schluß des Quelltextes, nachdem alle in ihm benutzten Worte zuvor definiert worden sind.

Längere Programme werden im Text-Editor angelegt. Bei unseren ersten Versuchen können wir darauf noch verzichten. Wir schreiben z.B.

```
: DEC DECIMAL ;
```

Beginn und Ende der Kompilierung
Name
Anweisungsfolge

Das neue Wort DEC bewirkt das Gleiche wie DECIMAL, es ist nur schneller einzutippen. Und wir können gleich weiter definieren

```
: BINARY 2 BASE ! ;
```

Beginn und Ende der Kompilierung
Name
Anweisungsfolge

Die Anweisungsfolge entspricht den früheren Experimenten zur Zahlenwandlung: 2 BASE ! speichert die binäre Zahlenbasis 2 in jene symbolische Adresse BASE, die das System zur Zahleninterpretation unterhält. BINARY ist damit ein gleichwertiges 'Umschaltwort' wie HEX, DECIMAL oder DEC.

Der Leser probiere im Tischrechnermodus z.B. einmal aus:

```
DEC 5 BINARY . 101
```

Umschalten auf Basis 10
5 auf den Stack
Umschalten auf binär für Ausgabe
Druckanweisung für 16-Bit Zahlen
Ausgabe der 5 als Binärzahl

3.4 Das Dictionary - die Befehlsliste

Ehe wir weitere Befehlswort definieren, wollen wir einmal das 'dictionary' aufschlagen. Das geschieht mit dem Befehl VLIST, also

```
HEX VLIST
```

Auf dem AIM 65 finden wir dann folgende Ausgabe:

```
VLIST
  322 BINARY
  313 DEC
  309 TASK
 C9FA .S
 - - -
```

Die Auslistung kann jederzeit unterbrochen werden, indem man auf irgendeine Taste drückt. Vor den Befehlsworten finden wir in hexadezimaler Notierung die Speicheradresse des Befehlswortes, genauer gesagt: seine Parameterfeldadresse. Dazu später mehr. Natürlich hätten wir die Adressen auch in einem anderen Zahlensystem darstellen lassen können. Wir stellen

fest, daß neue Befehlsworte an jeweils höheren RAM-Adressen abgespeichert werden. Das Leerwort TASK (dummy word) residiert ebenfalls noch im RAM. Alle Worte ab ' S. ' bis ' LIT ' sind beim AIM unter ROM-Adressen zu finden. (Sofern das FORTH für andere Rechner zu laden ist, findet sich natürlich alles im RAM.)

Das Dictionary ist wie eine Perlenschnur, es reiht sich Befehlswort an Befehlswort. Für uns zunächst unsichtbar sind sie untereinander durch Zeiger ('links' in der lfa, Linkfeldadresse) verbunden. Das Absuchen des Vokabulars im Tischrechnermodus oder beim Kompilieren neuer Worte geschieht immer vom letzten Eintrag her im Rückwärtsgang. Die Abarbeitung eines bereits kompilierten Wortes erfordert dagegen keinerlei Sucharbeit. Kompilierte Worte enthalten in ihrem Parameterfeld die Adreßvektoren der in ihrer Definition enthaltenen/benutzten Worte. Die Aneinanderreihung (Auffädelung) solcher Adreßvektoren ist einer Kette von Unterprogrammaufrufen mit effektiver Adresse vergleichbar. Hier liegt das 'Geheimnis' des schnell ausführenden FORTH.

Den bisherigen Übungen ist weiterhin zu entnehmen: Die Programmierung erfolgt ohne Zeilennummern. Befehle kommen mit dem Aufruf ihres Namens zur Ausführung. Dienstleistungen, die wir in BASIC oder Assembler als Unterprogramm anlegen würden, werden in FORTH als nachgeordnete Befehlsworte definiert. Es kann, wie wir noch sehen, auch mit benamten Konstanten und Variablen gearbeitet werden. Das BASE der früheren Beispiele ist eine solche Systemvariable.

Im RAM befindliche Befehlsworte können mit FORGET wieder getilgt werden. FORGET DEC würde das Wort DEC und alle ihm folgenden Worte (BINARY etc.) tilgen. Das Arbeiten mit FORGET ermöglicht nicht nur eine Bereinigung, sondern auch ein Arbeiten mit Programm-Overlays. Hinweise dazu später.

In manchen Fällen wird man ein in der VLIST enthaltenes Wort nicht mit FORGET tilgen können, weil sich während seiner Kompilierung ein Unglück ereignete. FORTH kann dann dieses Wort nicht erkennen. Abhilfe: Zuerst das Wort SMUDGE geben, dann das FORGET.

3.5 Einfache Befehle zur Textein- und Ausgabe

In BASIC kennen wir den Ausdruck eines Strings mit der Anweisung PRINT "Textkette". Der entsprechende Befehl in FORTH ist

```
." Textkette"     (dot-quote)          Ausgabe eines Literals
```

Auf das hochgestellte Anführungszeichen muß ein Space als Begrenzer folgen, erst dann beginnt der auszugebende Text. Das Textende wird mit einem abschließenden Anführungszeichen gekennzeichnet, ein Space ist nicht mehr erforderlich, es sei denn, er soll zur Textkette gehören.

Der ." -Befehl hat also die Grundform ." ". Er wird zweckmäßigerweise mit dem weiteren Befehl 'CR' verbunden, der einen Vorschub auf eine neue Zeile bewirkt. Man kann den Ausgabebefehl für Textketten natürlich auch in die Definition von Befehlswörtern einbeziehen, z.B.

```
: M1 ." TEXT 1" CR ;          M1 und M2 geben Texte aus und
: M2 ." TEXT 2" CR ;          bewirken einen Zeilenvorschub
```

Diese Art der Textausgabe eignet sich für die Erzeugung von Prompts zur Bedienerführung. Weitere nützliche Worte sind

```
SPACE          erzeugt 1 Zwischenraum
n SPACES       erzeugt n Zwischenräume
```

In BASIC sind die Stringfunktionen wie LEFT$, MID$, RIGHT$, VAL, ASC, CHR$ usw. für die Textbearbeitung sehr nützlich. Dort können wir z.B. auch sagen A$ = A$ + B$

oder PRINT A$, B$. Das Gleiche hat FORTH von Haus aus meistens nicht. Man kann solche Funtionen aber nachimpementieren. Vorschläge dafür in diesem Heft.

Der sehr nützliche Befehl .S dient dem nicht zerstörenden Ausdruck des Datenstacks für Untersuchungszwecke. Das 'oberste' Item (Element) wird zuerst angezeigt (als Zahl), dann das nächste usw.. Wenn nichts auf dem Stack ist, erfolgt nur die einfache Rückmeldung EMPTY OK. Symbolisch

.S	(—)	Anzeige des Datenstacks

Der Befehl .S setzt die Zelleninhalte (16 Bit) auf dem Stack in Zahlen um, und zwar in solchen zur eingestellten BASE. Ein kleiner Versuch:

DECIMAL 33 44 55 .S 55 44 33 OK	Anzeige dezimal
HEX .S 37 2C 21	Anzeige hexadezimal

Wir erkennen sehr schön die zur Eingabe umgekehrte Reihenfolge der Ausgabe. - Eine auf dem Stack befindliche Zahl kann aber auch als die ASCII-Codierung eines Zeichens interpretiert und auf die aktive Ausgabe geschrieben werden, und zwar mit

EMIT	(n —)	Ausgabe Zahl als ASCII-Zeichen

Die Zahl wird dabei 'verbraucht'. In der Wirkung entspricht EMIT etwa dem PRINT CHR$ (Zahl) (1 Byte), also z.B.

HEX 37 EMIT 7 OK

Das System gibt eine 7 aus, denn hex 37 ist die ASCII-Codierung für 7. Das Gleiche ginge mit DECIMAL 55 EMIT 7 OK

Eingabebefehle

Die Eingabe eines einzelnen Zeichens von der Tastatur auf den Datenstack erfolgt mit

KEY	(— char)	Eingabe 1 Zeichen von der Tastatur

Nach dem Auslösen des Wortes KEY wartet die Eingabe auf 1 Zeichen. Das System meldet sich erst nach einem Tastendruck mit 'OK' zurück. Die Wirkung entspricht in BASIC der Anweisung 10 GET A$: IF A$ = "" THEN 10. Mit einer einfachen Definition wollen wir nun den mit KEY empfangenen Tastendruck wieder vom Stack her als ASCII-Zeichen in eine separate Zeile ausdrucken:

: ASCII KEY CR EMIT CR ;

Das neue Wort ASCII bewirkt also das Echo eines Tastendruckes. Wir lassen nun einmal die Phantasie etwas schweifen: Ein großes 'A' wird als Hexbyte 41 gespeichert, ein kleines 'a' als hex 61. Man könnte doch nun zu jedem mit KEY empfangenen Byte hex 20 hinzuaddieren und hätte damit für den Tastenbereich A – Z Kleinschreibung. Wer ein Video-Terminal mit Kleinschreibung hat, möge einmal versuchen

: KLEIN KEY CR HEX 20 + EMIT CR ;

Der Term ' 20 + ' addiert hex 20 zum Byte, das mit KEY empfangen wurde. EMIT besorgt die Ausgabe, die 'CR' bewirken jeweils einen Zeilenvorschub. Wer noch mehr probieren möchte, addiere oder subtrahiere einmal andere Zahlen als hex 20. Damit wären Ansätze für eine Verschlüsselung gegeben.

Eine Textkette wird mit EXPECT von der aktiven Eingabe empfangen. Man muß jedoch vor EXPECT angeben, an welche symbolische oder explizit genannte Adresse die Zeichen gespeichert werden sollen und wieviel Zeichen maximal ankommen dürfen. Die Eingabe bricht dann mit dem Erreichen dieser Zählung ab oder wenn vorher ein Carriage Return empfangen wurde. Ein sicherer Empfangsort hat den symbolischen Namen PAD. Er befindet sich immer

68 Bytes oberhalb des gegenwärtigen Vokabular-Endes. Damit haben wie folgendes Format:

EXPECT (adr count --) Eingabe einer Textkette

Man mache einmal einen Versuch mit

DECIMAL PAD 8 EXPECT

Das Wort PAD befördert automatisch die Anfangsadresse eines geeigneten Zeichpuffers auf den Datenstack. Die folgende 8 ist der count und EXPECT die Eingaberoutine, die auch den Empfangspointer weitersetzt. Wir können nach Auslösung dieser Befehlszeile nun bis zu 8 Zeichen eingeben oder vorher mit (RET) abbrechen. Das System meldet sich dann mit 'OK' zurück.

Die Zeichen befinden sich nun irgendwo im Speicher. Die Frage ist, wie man sie nun weiterverwerten kann. Mit PAD läßt sich das erste Byte des Eingabepuffers wiederum adressieren. Mit weiteren Befehlen des FORTH und ggfs. Programmschleifen ist ab dieser Adresse eine Bearbeitung der empfangenen Bytes möglich. Im Moment zeigen wir nur, wie die Ausgabe der empfangenen oder irgendeiner anderen bereits adressierten Textkette möglich ist:

TYPE (adr count --) Textausgabe ab Adresse

Es handelt sich um den Ausgabebefehl für ASCII-codierte Zeichen auf die aktive Ausgabeeinheit. Dabei ist 'adr' die Anfangsadresse und 'count' die Menge der auszugebenden Zeichen wobei das System nicht auf irgendwelche Begrenzer achtet. Der Leser mache einen Versuch:

PAD 8 CR TYPE CR Ausgabe von 8 Zeichen in eine besondere Zeile (wegen der 'CR')

Wenn die Eingabe auch genau 8 Zeichen betrug, so wird man den Eingabetext identisch wiederfinden. War der Eingabetext kürzer, so wird im Anschluß an ihn 'Schrott' ausgegeben, der zufällige Inhalt der anschließenden Zellen also. Natürlich gibt es Befehle, den Eingabepuffer im Vorwege z.B. mit Leerzeichen (BLANKS) oder mit Nullen aufzufüllen (ERASE), z.B.

PAD 20 ERASE Einschreiben von 20 Nullen in den Puffer

Eine Suchroutine kann dann den tatsächlich benutzten Byte-count ermitteln, ehe man TYPE anwendet. In den Stringroutinen dieses Heftes wird im Befehlswort SRCH dieser Dienst zur Verfügung gestellt.

3.6 Zählschleifen in FORTH

Das Absuchen eines Strings auf einen Begrenzer fordert geradezu die Einführung von Programmschleifen, die mit

BEGIN Anweisungskette Überprüfung der Endbedingung UNTIL
(Format einer Bedingungsschleife)

als Bedingungsschleife formuliert werden. Das Prinzip der strukturierten Programmierung wollen wollen wir zunächst jedoch an Zählschleifen verdeutlichen. Sie haben die Grundform

: NAME DO Anweisungen LOOP ; Format der Zählschleifen

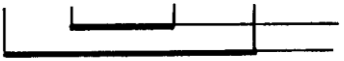
Alle strukturierenden Befehlsworte, wie DO .. LOOP, BEGIN .. UNTIL, IF .. ELSE .. THEN usw. können nicht im Tischrechnermodus benutzt werden, sondern nur innerhalb von Befehlswort-Definitionen zwischen Doppelpunkt und Semikolon, denn sie erfordern vom System die Errechnung und Kompilierung von Sprüngen innerhalb des Konstrukts.

Dabei kann vor dem DO eine Anweisungskette stehen, zwischen DO und LOOP müssen keine Anweisungen stehen (leere Schleife, Zeitverzögerung). Nach LOOP können bei Bedarf weitere Anweisungen folgen.

Schleifen und andere Ablaufstrukturen dürfen ineinander verschachtelt werden, wenn die

Regel beachtet wird, daß das innere Konstrukt immer als erstes aufgelöst wird, also z.B. nach der Form

```
: NAME ... DO ... DO ... LOOP ... LOOP ... ;
```



Schachtelung:
innere Schleife
äußere Schleife

Die Worte DO und LOOP sind dabei immer als ein Paar zu verwenden, andernfalls quittiert das System einen Fehler.

Wie auch die BASIC-Zählschleife FOR I = 1 TO 99 : Anweisungen : NEXT I, so erfordert auch die DO ... LOOP Begrenzungen für die Zählung der Laufvariblen I. Diese Begrenzungen müssen sich vor Ausführung der DO ... LOOP als 2 Integerzahlen auf dem Datenstack befinden und zwar als

(Endwert+1 Anfangswert —) Begrenzer der Laufvariablen I

Hinterlassen wird nichts, auch das I nicht, das man aber mit anderen Mitteln retten kann.

Es ist also wieder umgekehrt polnisch. Im Gegensatz zum BASIC (Ausführung bis einschließlich 99) wird in FORTH *ausschließlich* Endwert ausgeführt. Als Zählparameter notieren wir daher immer: Erwünschter Endwert+1, wir würden vergleichbar also ' 100 ' sagen müssen.

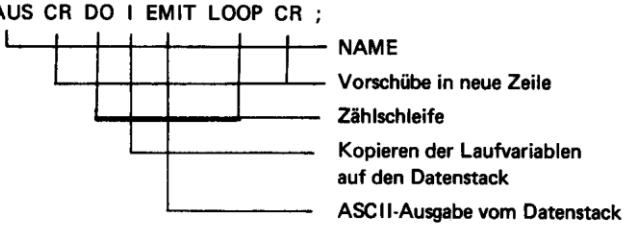
Zum Verhalten der DO ... LOOP während der Ausführung ist zu bemerken, daß DO die beiden begrenzenden Zählparameter vom Datenstack entfernt und auf den Hardwarestack bringt, so daß uns innerhalb der Schleife bei Bedarf evtl. zuvor bereitgestellte weitere Parameter zuoberst auf dem Datenstack zur Verfügung stehen, ohne daß wir dort umschaufeln müssen.

Für das nächste Beispiel wird noch ein Befehl erklärt:

I (— n) kopiere Laufvariable vom Hardwarestack auf den Datenstack

Dieser Befehl kann innerhalb von DO ... LOOPs benutzt werden. Wir stellen uns jetzt die Aufgabe, die Buchstaben von A - K mit Hilfe einer Zählschleife auf die Anzeige auszugeben. Zu den Begrenzern für die DO ... LOOP: ASCII 'A' wird hex als 41 dargestellt (unterer Begrenzer), das 'K' als hex 4B, der obere Begrenzer muß also 4C sein. Die eigentliche Ausgaberoutine definieren wir zu

```
: AUS CR DO I EMIT LOOP CR ;
```

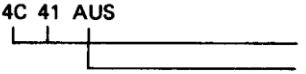


NAME
Vorschübe in neue Zeile
Zählschleife
Kopieren der Laufvariablen auf den Datenstack
ASCII-Ausgabe vom Datenstack

AUS (Endwert+1 Anfangswert —)

Wenn wir jetzt aufrufen

```
HEX 4C 41 AUS
```



Begrenzer auf Datenstack bringen
Zählschleife mit ASCII-Ausgabe der Laufvariablen

so erhalten wir

ABCDEFGHIJK

Der Leser möge mit weiteren Werten der Begrenzer arbeiten. Erwähnenswert mag sein, daß unser Befehlswort 'AUS' nicht nur zwischen DO und LOOP Anweisungen enthält, sondern auch vor DO und nach LOOP, nämlich die Zeilentransporte.

Es ist dem Leser unbenommen, in die Definition für 'AUS' vor dem DO die untere Grenze einzusetzen, so daß immer z.B. ab 'A' ausgegeben wird:

```
: AUS2 CR 41 DO I EMIT LOOP CR ;
AUS2              ( Endwert+1 --- )
```

Oder man will immer bis einschließlich 'Z' ausgeben, aber ab Zeichen, das per Tastendruck mit KEY übergeben wird, z.B.

```
: AUS3 CR 5B KEY CR DO I EMIT LOOP ;
                  |  |
                  |  |_____ obere Begrenzung
                  |________ ab Zeichen
```

Es ist der Vorteil der DO ... LOOP, daß FORTH selbst auf das Erreichen des Endbegrenzers achtet, den wir sonst mit einem Vergleich und UNTIL überprüfen müßten. Weitere Spielarten sind möglich, z.B. mit der +LOOP: Die Laufvariable I wird um die Schrittweite erhöht, die als fester oder berechneter Wert zuoberst auf dem Datenstack liegt (entspricht dem STEP in BASIC).

(Wird fortgesetzt) R. L.

Hans-Georg Lange, 1000 Berlin 30

Mengen in FORTH

Die Verwendung von Mengen (Sets) bietet effiziente Speicherausnutzung, da von jedem Element einer Menge nur 1 Bit belegt wird. Mengenoperationen erleichtern die Programmierung komplexer Entscheidungsstrategien.

Bekanntlich bietet PASCAL die Möglichkeit, Mengen zu definieren und mit diesen Mengen Operationen durchzuführen. Das hier wiedergegebene Programm erweitert das Vokabular (dictionary) der Sprache FORTH um den Datentyp 'SET'.

Eine durch SET SETNAME definierte Menge kann aus SETSIZE Elementen bestehen, wobei die Größe einer Menge nur durch den vorhandenen Speicherplatz begrenzt wird. Da alle Mengen zueinander kompatibel sind, sind auch alle Mengenoperationen möglich. Außerdem sind Befehle zum Setzen oder Löschen einzelner oder aller Elemente einer Menge, zum Test auf Vorhandensein eines Elementes sowie zum formatierten Ausdruck vorhanden.

Implementierungshinweise

Das Programm wurde auf einem CBM 3032 mit einem 16-Bit-Integer FORTH-System entwickelt, dessen Vokabular sich weitgehend an den FIG-Standard hält (FORTH Interest Group). DTOP zeigt auf das erste freie Byte nach Ende des FORTH-Vokabulars (Anm. d. Hrsg. DTOP entspricht dem HERE), 1 X +! addiert 1 zur Variablen X, A B /MOD teilt A durch B und legt Quotient und Rest auf dem Stack ab.

Eine einfache Anwendung von Mengen zeigt das Programm SETDEMO. Man führe mit den Mengen FRUECHTE und SUEDFRUECHTE einige Operationen durch und prüfe mit dem Wort TEST auf Vorhandensein eines Elementes.

```
 1 (
2    DEFINING AND USING SETS IN FORTH
3 .
4    HANS-GEORG LANGE
5    GOLTZSTR. 18
6    1000 BERLIN 30
7                    TEL 030 215 19 47
8 .
9                    25/04/82
10 )
11 FORGET SETSIZE
12 HEX
13 (
14 )
15 80                CONSTANT SETSIZE
16 SETSIZE 8 /       CONSTANT HLIM
17 SETSIZE 8 / 2 -   CONSTANT DLIM
18 (
19   SET SETNAME DEFINES A SET
20   NAMED SETNAME OF SETSIZE
21   MEMBERS. SETSIZE MUST BE A
22   MULTIPLE OF 16 DECIMAL AND
23   IS DEFINED AS A CONSTANT IN
24   LINE 15 HERE.
25 )
26 : SET
27    SETSIZE 8 / DTOP +!
28          ;:
29 ;
30 (
31   SETNAME SET$ LOADS THE CONTENTS
32   OF THE SET NAMED SETNAME
33   ON TOP OF STACK
34 )
35 : SET$
36    DUP HLIM + SWAP
37     DO
38      I $
39      2
40     +LOOP
41 ;
42 (
43   SET1 SET2 SET+ BUILDS THE UNION
44   OF BOTH SETS AND STORES RESULT
45   TOP OF STACK
46 )
47 : SET+
48    SET$
49     0 DLIM DO
50      DTOP $ I + !
51      -2
52     +LOOP
53    SET$
54     0 DLIM DO
55      DTOP $ I + $  OR
56      DTOP $ I + !
57      -2
58     +LOOP
```

```
59      DTOP $ SET$
60 ;
61 (
62    SET1 SET2 SETEOR BUILDS THE
63    DIFFERENCE OF BOTH SETS AND
64    STORES RESULT TOP OF STACK.
65    I.E. LET SET1= 1111 , SET2= 0101
66    THEN SETEOR GETS 1010
67 )
68 : SETEOR
69     SET$
70      0 DLIM DO
71        DTOP $ I + !
72        -2
73      +LOOP
74     SET$
75      0 DLIM DO
76        DTOP $ I + $ EOR
77        DTOP $ I + !
78        -2
79      +LOOP
80      DTOP $ SET$
81 ;
82 (
83    SET1 SET2 SET* CALCULATES THE
84    INTERSECTION OF BOTH SETS AND
85    LEAVES RESULT TOP OF STACK.
86 )
87  : SET*
88     SET$
89      0 DLIM DO
90        DTOP $ I + !
91        -2
92      +LOOP
93     SET$
94      0 DLIM DO
95        DTOP $ I + $ AND
96        DTOP $ I + !
97        -2
98      +LOOP
99      DTOP $ SET$
100 ;
101
102 (
103   SETNAME SET! STORES THE SET ON
104   TOP OF STACK / AS LEFT BY SET$
105   OR SET-OPERATIONS / IN THE SET
106   NAMED SETNAME. CARE MUST BE
107   TAKEN THAT A SET IS ACTUALLY
108   PRESENT ON TOP OF STACK
109 )
110 : SET!
111     DUP DLIM + DO
112        I   !
113        -2
114      +LOOP
115 ;
116 (
```

```
117   VARIABLES USED TO KEEP TRACK
118   OF POSITION OF A MEMBER IN A SET
119 )
120 0 VARIABLE SPOS
121 0 VARIABLE STEMP
122 (
123   GSPOS GETS THE POSITION OF
124   A SET'S MEMBER
125 )
126 : GSPOS
127    STEMP !
128    1 - DUP 8 /MOD STEMP +!
129    1 SPOS !
130    DUP  0=
131     IF DROP
132     ELSE 0 DO SPOS $ SPOS +! LOOP
133    THEN
134   DROP
135 ;
136 (
137   NUMBER SETNAME PUTMEM PLACES
138   NUMBER IN THE SET NAMED SETNAME.
139   NUMBER MAY BE AN INTEGER CONST
140   OR VARIABLE RANGING FROM
141   1 TO SETSIZE
142 )
143 : PUTMEM
144     GSPOS
145     STEMP $ $ SPOS $ OR STEMP $ !
146 ;
147 (
148   NUMBER SETNAME DELMEM
149   DELETES THE MEMBER NAMED NUMBER
150   FROM THE SET NAMED SETNAME.
151   SEE PUTMEM
152 )
153 : DELMEM
154     GSPOS
155     STEMP $ $ SPOS $ FFFF EOR
156     AND STEMP $ !
157 ;
158 (
159   NUMBER SETNAME TSTMEM
160   TESTS THE SET NAMED SETNAME FOR
161   MEMBERSHIP OF NUMBER. IF PRESENT,
162   LEAVE 1 ON TOP OF STACK, ELSE
163   LEAVE 0
164 )
165 : TSTMEM
166     GSPOS
167     STEMP $ $ SPOS $ AND 0=
168      IF 0 ELSE 1 THEN
169 ;
170 (
171   SET1 SET2 SET- BUILDS THE SET
172   OF MEMBERS OF SET1, THAT ARE NOT
173   MEMBERS OF SET2 ON TOP OF STACK.
174   I.E. LET S1= 1111 , S2= 0101
```

```
175    THEN S1 S2 SET- YIELDS 1010 ,
176         S2 S1 SET- YIELDS 0000
177 )
178 : SET-
179     DUP ROT
180     SET+ DTOP $ SET! SET$
181      0 DLIM DO
182        DTOP $ I + $ EOR
183        DTOP $ I + !
184        -2
185       +LOOP
186     DTOP $ SET$
187 ;
188 (
189   SETNAME SET. DISPLAYS THE SET
190   NAMED SETNAME ON VDU. PRINTOUT
191   IS A BINARY MAP, A 1 INDICATES
192   MEMBERSHIP.
193   HIGH ORDER MEMBER IS DISPLAYED
194   FIRST.
195 )
196 : SET.
197    CR CR
198   "    10 F E D C B A 9 8 7 6" S.
199    "   5 4 3 2 1" S.
200    0 SETSIZE 10 / 1 - DO
201      CR CR I F <=
202       IF 1 SPACES
203      THEN
204      I . 1 SPACES
205     0 F DO
206      DUP
207      J 10 * I + 1 + SWAP
208      TSTMEM
209      . 1 SPACES
210     -1 +LOOP
211    -1 +LOOP
212   DROP CR
213 ;
214 (
215   SETNAME SETCLR DELETES ALL
216   MEMBERS OF THE SET NAMED SETNAME
217   I.E. BUILDS AN EMPTY SET
218 )
219 : SETCLR
220     DUP DUP STEMP !
221     SET- STEMP $ SET!
222 ;
223 (
224   SETNAME SETINV INVERTS
225   A SET I.E LET S1= 0101 , THEN
226   S1 SETINV GETS 1010
227 )
228  : SETINV
229    DUP STEMP ! DUP HLIM + SWAP
230     DO
231      I $ FFFF EOR
232      2
```

```
233      +LOOP
234     STEMP $ SET!
235 ;
236
237 (
238   SETNAME SETSET SETS ALL
239   MEMBERS OF SETNAME TO A 1
240 )
241 : SETSET
242    DUP SETCLR SETINV
243 ;
244 (
245   SOME EXAMPLES
246 )
247 SET S1 SET S2 SET S3
248 S1 SETCLR S2 SETCLR
249 (
250   MEMBERS OF S3 REMAIN UNDEF-
251 )
252 1 S1 PUTMEM 2 S2 PUTMEM
253
```

SETDEMO

```
 1 FORGET APFEL
2 1 CONSTANT APFEL
3 2 CONSTANT BIRNE
4 3 CONSTANT BANANE
5 4 CONSTANT KIRSCHE
6 5 CONSTANT WEINTRAUBE
7 6 CONSTANT ROSINE
8 7 CONSTANT KIWI
9 8 CONSTANT GRAPEFRUIT
10 9 CONSTANT PFIRSICH
11 (
12 )
13 SET FRUECHTE
14 SET SUEDFRUECHTE
15 (
16 )
17 BANANE SUEDFRUECHTE PUTMEM
18 KIWI   SUEDFRUECHTE PUTMEM
19 GRAPEFRUIT SUEDFRUECHTE PUTMEM
20 PFIRSICH SUEDFRUECHTE PUTMEM
21 (
22 )
23 APFEL       FRUECHTE PUTMEM
24 BIRNE       FRUECHTE PUTMEM
25 KIRSCHE     FRUECHTE PUTMEM
26 WEINTRAUBE FRUECHTE PUTMEM
27 (
28 )
29 : TEST TSTMEM  CR
30    IF " IST ENHALTEN" S.
31    ELSE " IST NICHT ENTHALTEN" S.
32   THEN
33 ;
```

Michael Zimmermann, 6102 Pfungstadt

FORTH im Eigenbau

oder

Hinter den Kulissen von FORTH

In jüngster Zeit wächst das Interesse an FORTH. Rockwell kommt dem mit einem FORTH-Interpreter im ROM entgegen. Hier soll ein anderer Weg beschritten werden, der es dem Leser ermöglicht, sein eigenes FORTH zu definieren und es seinen besonderen Bedürfnissen anzupassen.

Was nun ist eigentlich FORTH, eine Programmiersprache, ein Compiler oder ein Interpreter? Diese Frage kann nicht so einfach beantwortet werden, da Elemente aller dieser Kategorien enthalten sind, ohne daß FORTH das eine oder andere voll und ganz wäre. Man kann sich die Antwort eigentlich erst geben, wenn man seine besondere Struktur begreift, die mit den vorgenannten Begriffen eigentlich nur lose Verbindungen hat.

Lange Jahre erfolgte der Einstieg in die Programmierung von Microcomputern entweder über die Maschinensprache bzw. den Assembler (wie bei KIM oder AIM) oder BASIC (PET/CBM). Beides sind Sprachen, die sich einer strukturierten Programmierung widersetzen. In dem selben Maße, wie die BASIC-Benutzer zu PASCAL aufsteigen, setzen sich die Assembler-Programmierer mehr und mehr mit FORTH auseinander. Dabei zeigen sich seine positiven Eigenschaften, die sein System für nahezu alle steuernden Prozesse prädestinieren. Besonders ist seine Anpassungsfähigkeit an die verschiedensten Umgebungen hervorzuheben. Das wird durch einen kleinen maschinenabhängigen 'Nukleus' und einen viel größeren in FORTH selbst definierten Teil erreicht. Durch diese Struktur ist in nahezu idealer Weise eine große Portabilität gewährleistet.

Besonders unter dem Gesichtspunkt der Anpassung ist es fast ein 'Sakrileg', FORTH in die Unabänderlichkeit eines ROM zu pressen. Man muß allerdings zugeben, daß dabei eine ganze Reihe von wünschenswerten Dienstleistungen mit einbezogen wurde. Die Anpassungsfähigkeit wird hier aber durch die Aufspaltung des Vokabulars in System- und Benutzerteil verkompliziert. Ebenso sind die auf dem AIM entwickelten FORTH-Programme nur auf diesem lauffähig, da sie immer die ROM's zur Ausführung benötigen.

Demgegenüber ermöglicht es eine Eigenentwicklung, daß über das systembedingte Minimum hinaus gerade soviel Routinen vorgegalten werden, wie es der Zweck erfordert. Solche Routinen sind darüber hinaus EPROM-fähig. Auch das wieder im erforderlichen Umfang und ohne den Overhead des Interpreters. Für ein solches lauffähiges Systemprogramm, das sowohl die FORTH-Systemteile als auch die benutzerspezifischen Routinen enthält, reicht der Assembler als Werkzeug voll aus.

Vor der Darstellung der eigentlichen Systemroutinen soll der Weg vom reinen Assembler zu FORTH beschrieben werden. Das zu beschreibende System stellt dabei keine vollständige Implementierung des FORTH dar und weicht zumindest in Details vom FORTH des AIM ab. Alle Grundfunktionen sind aber enthalten, so daß auch in diesem Falle von einem FORTH-System gesprochen werden kann.

Metamorphosen eines Programmes - Vom Assembler zu FORTH

Ausgangspunkt des Weges vom Assembler zum FORTH-Programm soll ein reines Maschinenprogramm sein, wie es in symbolischer Form im Beispiel 1 abgedruckt ist. In diesem Programm werden zwei Operanden, einer in $ 400/401, der andere in $ 480/481 in einen Arbeitspuffer an hex 00/01 und 02/03 geladen, in diesem Arbeitspuffer addiert und das Ergebnis aus 04/05 nach 400 und 401 zurückübertragen.

Diese Form des ausprogrammierten Codes kann in ein Hauptprogramm mit Unterprogrammaufrufen umgeformt werden und in eine Reihe von Unterprogrammen, die die eigentliche Verarbeitung

```
PROGRAMMBEISPIEL 1 - REINER ASSEMBLER-CODE
0000            0001            *=$300

LADEN OPERAND 1
0300 AD0004     0003            LDA $400
0303 8500       0004            STA $00
0305 AD0104     0005            LDA $401
0308 8501       0006            STA $01

LADEN OPERAND 2
030A AD8004     0008            LDA $480
030D 8502       0009            STA $02
030F AD8104     0010            LDA $481
0312 8503       0011            STA $03

ADDIEREN
0314 18         0013            CLC
0315 A500       0014            LDA $00
0317 6502       0015            ADC $02
0319 8504       0016            STA $04
031B A501       0017            LDA $01
031D 6903       0018            ADC #03
031F 8505       0019            STA $05

ZURUECKSPEICHERN
0321 A504       0021            LDA $04
0323 8D0004     0022            STA $400
0326 A505       0023            LDA $05
0328 8D0104     0024            STA $401

RUECKSPRUNG IN MONITOR
032B 4CBFE0     0026            JMP $E0BF
032E            0027            .END
```

vornehmen. Dies zeigt Beispiel 2, das in dieser Form noch ohne Unterstützung von irgendwelchen Systemprogrammen auf der 6502 lauffähig ist. Seine Form bringt noch keine direkten Vorteile, bei größeren Programmen nimmt aber durch die Aufteilung in ein einzelne Module die Übersichtlichkeit zu. Und auch jede Funktion, wie hier z.B. das Laden der Operanden, die Addition und das Rückübertragen des Resultates, ist als Code nur einmal erforderlich. Funktionen kommen damit durch einen JSR-Befehl zur Ausführung.

In den Fällen, wo von einem hinreichend umfangreichen Hauptprogramm nur eine relativ kleine Zahl von Unterprogrammen angesprochen wird, erscheint auch der JSR-Befehl noch zu platzaufwendig, da eine andere Operation im Hauptprogramm sowieso nicht auftreten kann. Es würde also für die Beschreibung des Programmablaufes ausreichen, lediglich die Startadressen der Unterprogramme in der Reihenfolge abzuspeichern, in der deren Durchlauf gewünscht wird.

Hierbei wird aus Gründen, die später erläutert werden, bei der Adreßfolge nicht direkt die Startadresse abgelegt, sondern eine Adresse, in der die eigentliche Anfangsadresse des Unterprogrammes abgespeichert ist. Bei Unterprogrammen in Maschinensprache soll dieses die Adresse direkt vor dem Maschinenprogramm sein, ihr Inhalt soll auf die Folgeadresse zeigen, eben auf den Anfang des Maschinencodes. Eine derartige Konstruktion soll mit 'CODE' beschrieben werden.

Ein Programm in dieser Form ist nicht mehr direkt mit dem 6502 ausfühbar, es ist vielmehr ein Systemprogramm erforderlich, das zum einen einen symbolischen Programmzähler führt und das zum anderen die Verbindung zwischen den Adressen und er Ausführung der dahinter stehenden Routinen vornimmt. Dieses Programm wird in FORTH-Tradition mit 'NEXT' angesprochen; alle Maschinenroutinen enden vergleichbar normalen Unterprogrammen, die mit 'RTS' enden, mit einem Sprung nach 'NEXT'.

```
PROGRAMMBEISPIEL 2
0000            0001 ;.ASSEMBLER-CODE ALS UNTERPROGRAMM
0000            0002         *=$300

HAUPTPROGRAMM
0300 200F03     0004         JSR LAD1
0303 201A03     0005         JSR LAD2
0306 202503     0006         JSR ADD
0309 203303     0007         JSR RUECK
030C 203E03     0008         JSR RETURN

LADEN OPERAND 1
030F AD0004     0010 LAD1    LDA $400
0312 8500       0011         STA $00
0314 AD0104     0012         LDA $401
0317 8501       0013         STA $01
0319 60         0014         RTS

LADEN OERAND 2
031A AD8004     0016 LAD2    LDA $480
031D 8502       0017         STA $02
031F AD8104     0018         LDA $481
0322 8503       0019         STA $03
0324 60         0020         RTS

ADDIEREN
0325 18         0022 ADD     CLC
0326 A500       0023         LDA $00
0328 6502       0024         ADC $02
032A 8504       0025         STA $04
032C A501       0026         LDA $01
032E 6503       0027         ADC $03
0330 8505       0028         STA $05
0332 60         0029         RTS

ZURUECKSPEICHERN
0333 A504       0031 RUECK   LDA $04
0335 8D0004     0032         STA $400
0338 A505       0033         LDA $05
033A 8D0104     0034         STA $401
033D 60         0035         RTS

RUECKSPRUNG IN MONITOR
033E 4CBFE0     0037 RETURN  JMP $E0BF
0341            0038         .END
```

Diese Routine 'NEXT' wird zusammen mit anderen Programmen, die dem Ablauf von FORTH-Programmen dienen, als interner Interpreter bezeichnet, im Gegensatz zu einem externen Interpreter, der Benutzereingaben in ein FORTH-Programm umsetzt.

Diese Form eines Programmes ist im Beipiel 3 verdeutlicht. Aus der beschriebenen Konstruktion von FORTH ergibt sich die größere Geschwindigkeit gegenüber BASIC: Während hier lediglich zu vorher festgelegten Adressen verzweigt wird, ist in BASIC die Interpretation von 'tokens' und sogar reinem Text erforderlich, obwohl eine gewisse Analogie zwischen einem 'token' und den Adressen der Routine in der FORTH-Form besteht.

Mit dem bis hierher erarbeiteten Rüstzeug ist es möglich, mit einem Hauptprogramm in FORTH-Form Maschinenunterprogramme zu verarbeiten. Ziel sollte es aber sein, auch Unterprogramme in FORTH-Struktur bearbeitbar zu machen. Da ein übergeordnetes Programm in FORTH generell

```
PROGRAMMBEISPIEL 3
0000            0001 ;ASSEMBLER-CODE ALS FOLGE
0000            0002 ;VON UNTERPROGRAMMADRESSEN

ADRESSE DER FORTH-SCHLEIFENROUTINE
0000            0004 NEXT    =$226
0000            0005         *=$300

HAUPTPROGRAMM
0300 0B03       0007         .WOR LAD1
0302 1A03       0008         .WOR LAD2
0304 2903       0009         .WOR ADD
0306 3B03       0010         .WOR RUECK
0308 4A03       0011         .WOR RETURN

LADEN OPERAND 1
030B 0D03       0013 LAD1    .WOR *+2          ;CODE
030D AD0004     0014         LDA $400
0310 8500       0015         STA $00
0312 AD0104     0016         LDA $401
0315 8501       0017         STA $01
0317 4C2602     0018         JMP NEXT

LADEN OERAND 2
031A 1C03       0020 LAD2    .WOR *+2          ;CODE
031C AD8004     0021         LDA $480
031F 8502       0022         STA $02
0321 AD8104     0023         LDA $481
0324 8503       0024         STA $03
0326 4C2602     0025         JMP NEXT

ADDIEREN
0329 2B03       0027 ADD     .WOR *+2          ;CODE
032B 18         0028         CLC
032C A500       0029         LDA $00
032E 6502       0030         ADC $02
0330 8504       0031         STA $04
0332 A501       0032         LDA $01
0334 6503       0033         ADC $03
0336 8505       0034         STA $05
0338 4C2602     0035         JMP NEXT

ZURUECKSPEICHERN
033B 3D03       0037 RUECK   .WOR *+2          ;CODE
033D A504       0038         LDA $04
033F 8D0004     0039         STA $400
0342 A505       0040         LDA $05
0344 8D0104     0041         STA $401
0347 4C2602     0042         JMP NEXT

RUECKSPRUNG IN MONITOR
034A 4C03       0044 RETURN  .WOR *+2          ;CODE
034C 4CBFE0     0045         JMP $E0BF
034F            0046         .END
```

nur aus den Aufrufen von Adreßpointern von Unterprogrammen besteht, ist die Aufbereitung der FORTH-Form nicht an der aufrufenden Stelle möglich. Das muß vielmehr am Anfang des Unterprogrammes selbst geschehen. - Bei Maschinenprogrammen hatten wir schon die Code-Verbindung, den Pointer auf die Folgeadresse, kennengelernt (.WOR *+2). FORTH-Unterprogramme ihrerseits

```
PROGRAMMBEISPIEL 4
0000            0001 ;DAS PROGRAMM IN FORTH-STRUKTUR
0000            0002 ;MIT UNTERROUTINEN

ADRESSE DER FORTH-SCHLEIFENROUTINE
0000            0004 NEXT   =$226

ADRESSEN DER FORTH-
0000            0006 ;UNTERPROGRAMM-VERWALTUNGSROUTINEN
0000            0007 TCALL  =$241
0000            0008 TRET   =$259
0000            0009        *=$300

HAUPTPROGRAMM
0300 4102       0011        .WOR TCALL
0302 0A03       0012        .WOR VORB
0304 3003       0013        .WOR ADD
0306 4203       0014        .WOR RUECK
0308 5902       0015        .WOR TRET

VORBEREITUNGSROUTINE
030A 4102       0017 VORB   .WOR TCALL
030C 1203       0018        .WOR LAD1
030E 2103       0019        .WOR LAD2
0310 5902       0020        .WOR TRET
0312            0021
0312            0022
0312            0023 ;PROGRAMMTEILE LAD1, LAD2, ADD, RUECK
0312            0024 ;WIE IN BEISPIEL 3
```

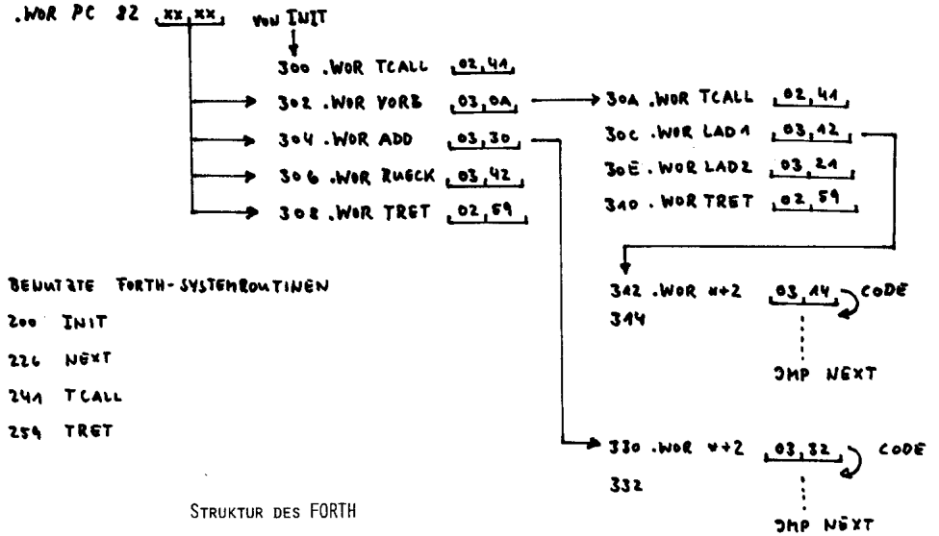
beginnen mit einem Pointer auf ein Maschinenprogramm 'TCALL', das die erforderlichen Aufgaben wahrnimmt, inbesondere das Sichern des Programmzählers und das Einstellen der neuen Sequenz auf die Folgeadresse ihres Aufrufes innerhalb des Unterprogrammes.

Jetzt wird auch die Berechtigung der Code-Konstruktion klar: Bei direktem Sprung auf Maschinen-Unterprogramme wären Unterroutinen in FORTH-Form nicht möglich. Diese Unterroutinen, die mit 'TCALL' angesteuert werden, schließen ab mit 'TRET', dem Rücksprungbefehl für FORTH Unterroutinen. - Von einem übergeordneten System aus gesehen, ist auch das FORTH-Hauptprogramm als Unterroutine zu begreifen. Aus diesem Grunde startet es auch mit TCALL und endet mit TRET. Hier muß lediglich in der Initialisierung der Rücksprung in den Monitor entsprechend den FORTH-Konventionen aufgebaut werden. - Nach diesen Vorarbeiten ist nun eine endgültige Beschreibung der Struktur eines FORTH-Programmes möglich, so wie es vom internen Interpreter decodiert und verarbeitet wird.

Struktur des FORTH

Die Struktur des FORTH wird mit Hilfe der beigefügten grafischen Darstellung beschrieben. Ausgangspunkt aller Betrachtungen ist der symbolische Programmzähler PC. Dieser zeigt auf eine Speicheradresse, unter der die Adresse eines Pointers auf eine Routine gespeichert ist. Beträgt der Inhalt von PC im Beispiel 304, so ist deren Inhalt die Startadresse einer Maschinenroutine ADD, die bei 330 beginnt. Da es sich um eine Maschinenroutine handeln soll, ist in Adresse 330 die Folgeadresse 332 gespeichert, ab 332 steht also der Code, der ADD zur Ausführung bringt.

So baut sich der Pfad vom symbolischen Programmzähler zum Maschinenprogramm auf. Ist der Inhalt des PC 302, so findet sich dort die Anfangsadresse einer anderen FORTH-Routine TRET ab Adresse 30A. Im Gegensatz nun zu maschinensprachlich formulierten Routinen hat diese in 30A/30B nicht den Verweis auf die Folgeadresse, sondern den Verweis auf die TCALL-Routine. TCALL sichert den den Wert des PC für den Rücksprung und lädt die Folgeadresse des TCALL-Aufrufes, nämlich 30C in den Programmzähler. Der Interpreter setzt dann die Ausführung an



STRUKTUR DES FORTH

dieser Stelle fort, indem er die vorgefundenen Adreßworte der Reihe nach in Aufrufe umsetzt, bis durch TRET ein Rücksprung erfolgt. – Die angegebenen Adressen sind nur Beispiele und stellen keine besondere Implementierung eines FORTH-Interpreters dar.

Daten und Unterprogramme

FORTH ist eine strukturierte Sprache. Unter diesem Gesichtspunkt ist auch die Darstellung von Unterprogrammen und Daten, Konstanten und Variablen, zu betrachten. Für die Speicherung des Rücksprungpunktes für Unterprogramme hat sich ein Stack-Konzept bewährt. In dieser Implementierung wird deswegen auch der Hardware-Stack der 6502 in Page 1 benutzt.

Für die Datenspeicherung soll hier ein 'Datenstack' in den Adressen von $ 00 bis 80 benutzt werden. Als Zeiger auf den Datenstack findet das X-Register Verwendung. Für den Hardwarestack wird der von der 6502 vorgehaltene Hardware-Stackpointer benutzt. Bei diesem findet eine Erhöhung im Zusammenhang mit dem Entnehmen und eine Verminderung mit dem Abspeichern von Werten statt. Beim Datenstack sind diese Bewegungen im X-Register nachzubilden; Da der Datenstack von den hohen Adressen her gefüllt wird, entspricht eine Erhöhung dem Entnehmen eines Wertes, eine Verminderung dem Einspeichern. Das X-Register weist als Stack;Zeiger immer auf die nächste freie Stelle. - Vor Verwendung des X-Registers für andere Zwecke ist dieses daher zunächst zu sichern und später wiederherzustellen.

Funktionen zur Verwaltung des Interpreters

Nach diesen Überlegungen ist es möglich, die zur Verwaltung des Interpreters erforderlichen Funktionen festzulegen:

Die Initialisierungsroutine muß
- einen indirekten Rücksprung auf den Monitor aufbauen,
- die Startadresse des FORTH-Programmes entgegennehmen,
- den Datenstack initialisieren und
- den Return-Stack initialisieren.

Die NEXT-Routine hat folgende Funktionen
- Sie muß den Inhalt der Speicherzelle holen, auf die der Programmzähler zeigt und für einen indirekten Sprung ablegen,
- den Programmzähler erhöhen,
- einen indirekten Sprung auf die Routine veranlassen.

Mit der TCALL-Routine wird
- der Inhalt des Programmzählers auf dem Return-Stack abgelegt,
- die Adresse aus dem indirekten Sprung entnommen und um 2 erhöht in den Programmzähler gebracht,
- zu NEXT verzweigt.

Die TRET-Routine
- führt einen Code-Befehl aus,
- entnimmt die Rücksprungadresse dem Return-Stack
- und speichert sie im symbolischen Programmzähler.

Auf eine genauere Beschreibung der Routinen kann hier verzichtet werden, kleine überschaubare Einheiten und eine ausreichende Dokumentation des Quelltextes zeigen hier schon die Einfachheit der FORTH-Struktur auf. Mit diesen Routinen ist das Gerüst des internen Interpreters aufgebaut, alle FORTH-Routinen werden mit dieser geringen Zahl von einfachen Dienstleistungen ausgeführt. Im Folgenden soll eine Reihe von weiteren Routinen dargestellt werden, die das eigentliche Arbeiten mit FORTH als Programmiersprache erst ermöglichen.

FORTH-Routinen

Nach den Routinen zur reinen Verwaltung eines FORTH-Programmes sollen nunmehr solche Routinen besprochen werden, die die vom Benutzer gewünschten Funktionen ausführen. Da diese im Quellentext ausführlich kommentiert sind, sollte an dieser Stelle eine Funktionsbeschreibung ausreichen.

Als Datenelemente werden hier nur Bytes und Worte (2 Bytes) zugelassen. Wenn beim Anwender weitergehende Anforderungen bestehen, so können diese, wie später zu zeigen sein wird, unschwer in das bestehende System implementiert werden. - Wie bereits erwähnt, erfolgen alle Datentransfers über den Datenstack. Aus diesem Grunde nehmen auch die Operationen, die den Datenstack bearbeiten, einen breiten Raum ein. Das oberste Element auf dem Stack, unabhängig ob in Byte- oder in Wortdarstellung, soll als Top Of Stack bezeichnet werden, das nächste als Second On Stack. Von der Speicheradressierung her befinden sich diese Elemente an der niedrigsten bisher belegten Adresse des Stack-Speichers und werden durch das X-Register indiziert angesprochen. Hierbei stellen in Byte-Darstellung

1,X TOP OF STACK und
2,X SECOND ON STACK dar.

In der WORD-Darstellung ergeben sich

1,X TOP OF STACK, niederes Byte
2,X TOP OF STACK, hohes Byte
3,X SECOND ON STACK, niederes Byte
4,X SECOND ON STACK, hohe Byte

In der Bezeichnung der Routinen wurde lose an die auch beim AIM-FORTH üblichen mnemonischen Begriffe angeknüpft, die den verbreiteten FORTH-Konventionen im wesentlichen entsprechen.

Die Routinen zur Stack-Manipulation

```
RECHNENDE ROUTINEN
------------------

INCR    INCREMENT
        ERHOEHT WORD AN TOP-OF-STACK UM 1

DECR    DECREMENT
        ERNEIDRIGT WORD AN TOP-OF-STACK UM 1
```

PLUS PLUS
ADDIERT WORDS IN TOP-OF-STACK UND SECOND-ON STACK, VERGISST ALTEN TOP UND SPEICHERT RESULTAT IM NEUEN TOP AB UND UEBERSCHREIBT DABEI ALTEN SECOND

MINUS MINUS
SUBTRAHIERT SECOND- VON TOP-OF-STACK, VERGISST ALTEN TOP,SPEICHERT RESULTAT IN NEUEN TOP UND UEBERSCHREIBT DABEI ALTEN SECOND

LADE- UND SPEICHEROPERATIONEN

PUSHB PUSH BYTE
BRINGT INHALT DES BYTE, DASS DEM OPERATIONSAUFRUF FOLGT IN DEN TOP-OF-STACK

PUSHW PUSH WORD
BRINGT INHALT DES WORDS, DASS DEM OPERATIONSAUFRUF FOLGT IN DEN TOP-OF-STACK

FETB FETCH BYTE
HOLT INHALT DES BYTES, DESSEN ADRESSE ALS WORD IM TOP-OF-STACK STEHT IN DEN TOP, UEBERSCHRIEBT DABEI DIE ADRESSE

FETW FETCH WORD
HOLT INHALT DES WORDS, DESSEN ADRESSE IM TOP-OF-STACK STEHT IN DEN TOP, UEBERSCHREIBT DABEI DIE ADRESSE

STB STORE BYTE
SPEICHERT INHALT DES BYTES, DASS ALS SECOND-ON-STACK STEHT AND DER ADRESSE AB, DIE ALS TOP STEHT, VERGISST DABEI ADRESSE UND BYTE

STW STORE WORD
SPEICHERT INHALT DES WORD, DASS ALS SECOND-ON-STACK STEHT AN DER ADRESSE AB, DIE ALS TOP STEHT, VERGISST DABEI ADRESSE UND WORD

LOGISCHE OPERATIONEN

SAND STACK AND
FUEHRT LOGISCHES AND ZWISCHEN TOP- UND SECOND-ON-STACK DURCH, VERGISST ALTEN TOP, SPEICHERT RESULTAT IN NEUEM TOP UND UEBERSCHREIBT DABEI ALTEN SECOND

SOR STACK OR
FUEHRT LOGISCHES OR ANALOG ZU SAND DURCH

SEOR STACK EOR
FUEHRT LOGISCHES EOR ANALOG ZU SAND DURCH

ROUTINEN DER EIN- UND AUSGABE

```
DOT     DOT
        GIBT WORD IM TOP-OF-STACK ALS HEXA-ZAHL AUS,
        VERBRAUCHT DABEI TOP

BNUM    BRACKET NUMBER
        LEGT EINGABE ALS HEXA-WORD IM TOP-OF-STACK AB

SPACE   SPACE
        GIBT SPACE AUS

CR      CARRIAGE RETURN
        GIBT CR AUS

KEY     KEY
        LIEST EIN ZEICHEN, LEGT ES ALS CHARACTER IM TOP AB

EMIT    EMIT
        GIBT ZEICHEN IM TOP AUS, VERBRAUCHT TOP DABEI
```

(Hinweis des Herausgebers: Die vorgenannten Routinen können aus Platzgründen erst im folgenden Heft ausgelistet werden.)

Das Arbeiten mit dem FORTH-Interpreter

Mit den bis jetzt beschriebenen Routinen ist es bereits möglich, lineare FORTH-Programme zu erstellen. Die weiterhin erforderlichen Kontrollbefehle werden später beschrieben. Da der ganze interne Interpreter als Assembler-Quelltext vorliegt, ist es durchaus möglich, das eigentliche FORTH-Programm ebenfalls als Assembler-Text als Folge von .WOR-Aufrufen zu erstellen. Hierbei ist es sinnvoll, nicht den ganzen internen Interpreter umzuwandeln, sondern diesen als Maschinencode zu laden und lediglich die Einsprungstellen der FORTH-Routinen für die Assemblierung zu definieren. Im vorliegenden Beispiel, als Run-Package bezeichnet, wird ein WORD als Adresse vom Benutzer angefordert und abhängig davon, ob das Bit 7 des Bytes an dieser Adresse gesetzt ist, eine Null oder Eins ausgegeben. - Sicher keine sehr umfangreiche Anwendung, sie zeigt aber hinreichend, wie nur unter Zurhilfenahme des Assemblers durchaus ein FORTH-ähnliches Programm erstellt werden kann. Auch eigene Assembler-Routinen sind hier unschwer und platzsparend hinzuzufügen, da eine Assemblierung in jedem Fall vorgenommen wird.

Auf den ersten Blick mag diese Form noch keine erheblichen Vorteile gegenüber einer reinen Assembler-Programmierung erkennen lassen, spätestens mit den im folgenden zu beschreibenden Kontrollstrukturen wird sich dies ändern, da mit diesem FORTH-System einerseits eine maschinennahe Programmierung möglich wird, andererseits aber das hierbei übliche Chaos in den Kontrollstrukturen vermieden wird.

(Wird fortgesetzt.)

```
0000        ;---------------------------------------------------------------
0000        FORTH-INTERPRETER
0000
0000        ;---------------------------------------------------------------
0000        PAGE ZERO
0000
0000        0000        0002        *0          ;128 BYTE PARAMETER-STACK
0000        0000        0003        *$80
0080        0080 00     0004 XSAV   .BYT 0      ;SAVE-SPEICHER FUER X-REGISTER
0081        0081 00     0005 OFLAG  .BYT 0
0082        0082 0000   0006 PC     .WOR 0      ;SYMBOLISCHER PROGRAMM-COUNTER
0084        0084 0000   0007 PTR    .WOR 0      ;POINTER FUER SPEICHERZUGRIFFE
```

```
0086          0086          0008        $$10C
010C          010C 4C0002   0009        JMP INIT
010F
010F          ---------------------------------------------------------
010F          AIM-RAM-ADRESSEN
010F
010F          010F          0011 ADDR   $A41C
010F
010F          ---------------------------------------------------------
010F          AIM-MONITOR-ADRESSEN
010F
010F          010F          0013 BLANK  $EB3E
010F          010F          0014 INALL  =$E993
010F          010F          0015 OUTALL =$E9BC
010F          010F          0016 CRLF   =$E9F0
010F          010F          0017 NUMA   =$EA46
010F          010F          0018 FROM   =$E7A3
010F          010F          0019 CRLF   =$E9F0
010F          010F          0020 ADDNE  =$EAB1
010F
010F          ---------------------------------------------------------
010F          INITIALISIER-PROGRAMM
010F
010F          010F          0022        $=$200
0200          0200          0023 INIT
0200          0200 A910     0024        LDA #$10        ;ZUERST RUECKSPRUNG AUF
0202          0202 8582     0025        STA PC          ;MONITOR IN $110 (F1)
0204          0204 A901     0026        LDA #$01        ;VORBEREITEN
0206          0206 8583     0027        STA PC+1
0208          0208 A9FC     0028        LDA #$FC
020A          020A 8D1001   0029        STA $0110
020D          020D A9FF     0030        LDA #$FF
020F          020F 8D1101   0031        STA $0111
0212          0212 20A3E7   0032        JSR FROM        ;ANFANGSADRESSE DES FORTH-PROGRAMMES
0215          0215 20F0E9   0033        JSR CRLF
0218          0218 AD1CA4   0034        LDA ADDR        ;HOLEN
021B          021B 8D4902   0035        STA IJMP+1      ;UND IM INDIREKTEN SPRUNG
021E          021E AD1DA4   0036        LDA ADDR+1      ;ABLEGEN
0221          0221 8D4A02   0037        STA IJMP+2
0224          0224 A2FF     0038        LDX #$FF        ;RETURN-STACK INITIALISIEREN
0226          0226 9A       0039        TXS
0227          0227 A27F     0040        LDX #$7F        ;DATA-STACK INITIALISIEREN
0229          0229 A900     0041        LDA #0          ;DEZIMAL-FLAG AUF BINAER SETZEN
022B          022B 8581     0042        STA DFLAG
022D          022D 4C4802   0043        JMP IJMP        ;VERZWEIGEN ZUM INDIREKTEN SPRUNG
0230          ---------------------------------------------------------
0230          NEXT-ROUTINE
0230
0230          0230          0045 NEXT
0230          0230 A000     0046        LDY #0          ;INHALT DER ADRESSE, AUF
0232          0232 B182     0047        LDA (PC),Y      ;DEN PC ZEIGT, HOLEN
0234          0234 8D4902   0048        STA IJMP+1      ;UND IM INDIREKTEN SPRUNG
0237          0237 C8       0049        INY             ;ABLEGEN
0238          0238 B182     0050        LDA (PC),Y
023A          023A 8D4A02   0051        STA IJMP+2
023D          023D 18       0052        CLC
023E          023E A582     0053        LDA PC          ;PROGRAMM-COUNTER UM 2 ERHOEHEN
0240          0240 6902     0054        ADC #2
0242          0242 8582     0055        STA PC
0244          0244 9002     0056        BCC IJMP
0246          0246 E683     0057        INC PC+1
```

```
0248    INDIREKTER SPRUNG
0248
0248    0248          0059 IJMP
0248    0248 6C0000   0060       JMP (0)
0248    ------------------------------------------------------------
0248    TCALL-ROUTINE
0248
0248    024B          0062 TCALL
024B    024B A583     0063       LDA PC+1      ;PROGRAMM-COUNTER HOLEN
024D    024D 48       0064       PHA           ;UND AUF DEN RETURN-STACK ABLEGEN
024E    024E A582     0065       LDA PC
0250    0250 48       0066       PHA
0251    0251 18       0067       CLC
0252    0252 AD4902   0068       LDA IJMP+1    ;INHALT DES INDIREKTEN SPRUNGES
0255    0255 6902     0069       ADC #2        ;HOLEN, UM 2 ERHOEHEN
0257    0257 8582     0070       STA PC        ;UND IM PROGRAMM-COUNTER ABLEGEN
0259    0259 AD4A02   0071       LDA IJMP+2
025C    025C 6900     0072       ADC #0
025E    025E 8583     0073       STA PC+1
0260    0260 4C3002   0074       JMP NEXT
0263    ------------------------------------------------------------
0263    TRET-ROUTINE
0263
0263    0263 6502     0076 TRET  .WOR *+2
0265    0265 68       0077       PLA           ;RUECKKEHRADRESSE VOM
0266    0266 8582     0078       STA PC        ;RETURN-STACK ENTNEHMEN UND
0268    0268 68       0079       PLA           ;IM PROGRAMM-COUNTER ABLEGEN
0269    0269 8583     0080       STA PC+1
026B    026B 4C3002   0081       JMP NEXT
```

#

Bücher

Brodie, Leo: Starting FORTH, An Introduction to the FORTH Language and Operating System for Beginners and Professionals. Verlag Prentice Hall Inc., Englewood Cliffs, NJ, 1981. ISBN 013-842922-7 (pbk.), ca. 350 Seiten, DM 48,70.

Leo Brodie hat mit diesem Buch die wohl bisher beste und umfassende Einführung in das Programmieren mit FORTH vorgelegt, und zwar für den verbreiteten Dialekt FIG FORTH (FORTH Interest Group). Er ist Mitarbeiter der FORTH Inc., und damit Mitarbeiter auch von Charles H. Moore, dem Schöpfer der Sprache. Das Buch ist so lebendig geschrieben, mit Zeichnungen und übersichtlichen Beispielen versehen, daß man als Leser bis zur letzten Seite gefesselt ist und dabei lernt, das Werkzeug FORTH zu gebrauchen. Die Darstellung ist in 13 Kapiteln angelegt und geht nach der notwendigen Einführung auch auf die strukturierenden Statements wie DO ... LOOP usw., Variable, Konstante und Arrays ein. Es wird der innere Aufbau der Sprache erklärt, die Ein- und Ausgabe, der Ausbau des Compilers u.a. mit 'defining words'.

Knecht, Ken: Introduction to FORTH. Verlag Howard W. Sams & Co, Inc., Indianapolis 1982, ISBN 0-672-21842-9, 142 Seiten, ca. DM 42,--.

Es handelt sich um ein Handbuch speziell für den Dialekt MMSFORTH, der auf den Computern Radio Shack TRS80 Modell I und III gefahren wird. In der Bezeichnung der Befehlsworte gibt es erhebliche Unterschiede zum FIG-FORTH. Das Buch bringt zwar etliche Beispiele, ist aber keine Einführung in das Programmieren, sondern vorwiegend eine Beschreibung, wie sich Befehle auswirken. Für den Benutzer eines FIG-FORTH ist es entbehrlich.

#

Ing. Othmar Kreil, A-5014 Salzburg und D-8228 Freilassing

Der AIM 65 PL/65-Compiler

Ein Anwenderbericht

Inbetriebnahme, Speicherbelegung

Der AIM-PL/65-Compiler wird in Form von zwei ROM's für die von B000 bis CFFF dekodierten Steckplätze und mit einem englischen Handbuch geliefert. Die Aufruftasten '5' und '6' haben jedoch eine andere Wirkung:

Mit '5' ruft man PL/65 auf, wenn man den TAB-Character (= '!') nur ein Space erzeugen lassen will, wie auf dem Thermodrucker.

Ruft man mit '6' auf, so werden für jeden TAB 5 Spaces abgesetzt, was bei mehr als 20 Zeichen pro Zeile den Ausdruck sehr übersichtlich gestaltet.

Während des Compilierens belegt PL/65 die Adressen bis herauf 049F für seinen Label-Stack. Somit ist 04A0 die erste freie RAM-Adresse für speicherresidente Verarbeitung. Im Betrieb belegt ein PL/65-generiertes Programm neben den vom Anwender her definierten Zero Page-Adressen von Vornherein 6 aufeinanderfolgende RAM-Adressen, vorzüglich von 0000 an. Das läßt sich jedoch mit der ENTRY-Anweisung ändern. Eine bare ENTRY-Anweisung legt den Verarbeitungsbeginn des Programmes auf hex 200 fest.

Der PL/65-Compiler arbeitet eng mit dem AIM-Monitor zusammen. Man benötigt ferner auch das auf D000 dekodierte Assembler-ROM oder einen anderen 6500-Assembler, da PL/65 einen Assembler Quellencode abgibt, den man noch durch den Assembler schicken muß, um ein ausführbares 6500-Object-Modul zu erhalten. Entsprechend aufwendig ist die Pflege der Programme und das Bereithalten der Listen für das Debuggen.

Hantierungsabläufe

Die Arbeitsgänge seien hiermit kurz umrissen:

1. Mittels des AIM-Editors formuliert man ein PL/65-Quellprogramm im Vokabular dieser Sprache. Dieses legt man möglichst auf einem externen Speichermedium (Cassette oder Diskette) ab.

2. In zwei Läufen erhält man PASS 1: die Liste des PL/65-Programmes mit Fehlerliste der Syntaxfehler und in PASS 2 den Quellencode für den 6500-Assembler. Diesen gibt man möglichst auch auf einen externen Datenträger aus.

3. Der Assembler generiert aus dem zuvor erzeugten Assembler-Quellentext das 6500-Object-Modul. Auch dieses speichert man möglichst auf externen Datenträger.

Somit erhält man zwei Listen, nämlich für das ursprüngliche PL/65-Programm und für das von ihm generierte Assembler-Quellenprogramm, und man hat drei Files, nämlich das für das PL6/65-Programm, das für die Assembler-Quelle und schließlich das Object-Modul.

Das in PL/65 formulierte Programm ist reichlich zu segmentieren, besonders im Falle einer Minimal-RAM-Konfiguration, da der erzeugte Assembler-Quellencode etwa drei- bis viermal so lang ist und dann möglicherweise nicht mehr in den Puffer des Editors paßt. - Beim Autor wird coresident mit dem Label-Stack des PL/65 ein Treiberprogramm für einen VIC-1-betriebenen Fernseher gespeichert. Ich kompiliere immer coresident, da ich jedes Modul sofort mit dem PL/65 nach Syntaxfehlern durchsuche und mit dem Editor verbessere. Wie das möglich ist, wird im nächsten Abschnitt beschrieben.

Die Syntaxprüfung des PL/65

PL/65 nimmt nur wenige Prüfungen vor: Ob alle Zeilen mit einem Semikolon abgeschlossen sind, ob Mehrfachanweisungen in einer Zeile durch Semikolon getrennt sind und ob die Paritäten zwischen BEGIN; und END; der Verarbeitungsblöcke aufgelöst werden können. Wenn das der Fall ist,

wird das PL/65-Quellprogramm anstandslos übersetzt. PL/65 macht keine Zuweisungen oder Bewertungen. Man muß sich dabei vorstellen, daß PL/65 ein Durchgang ist, mit dem aus einigen wenigen Statements eine ganze Prozedur in Assemblersprache erzeugt wird, die der Assembler dann noch in die Maschinensprache übersetzen muß.

Fehlende Semikolons am Zeilenende oder zwischen Mehrfachanweisungen in einer Zeile werden in Minderheiten als Fehler erkannt und auch als solche angezeigt. Bei deren Häufung jedoch wird der Kompilierungsvorgang sehr schnell eingestellt, ebenso bei Nichtparität zwischen BEGIN; und END;. Ein Ausdruck des zweiten Kompilier-Passes sorgt sehr schnell für Klarheit, warum der Vorgang abgebrochen wurde.

Ein anderer Grund für einen vollständigen Abbruch ist die Überschreitung der maximal möglichen Vorwärtsreferenzen. Diese sind mit etwa 10 bis 16 sehr eng begrenzt. Den Grund dafür habe ich nicht erforscht, aber ich habe einen Ausweg gefunden.

FOR ... END; - WHILE ... END; und IF ... THEN ...ELSE ...END; -Sequenzen verursachen diese Vorwärtsreferenzen, besonders im Falle einer Schachtelung. Das Maximum von etwa 3-4 Niveaus ist sehr schnell erreicht bzw. überschritten. Ich helfe mir da immer mit einem Subroutine-CALL, woselbst ich wiederum dieselbe Anzahl an Niveaus zur Verfügung habe. Bei dieser strukturierten Programmierung ist es bei geeigneter Planung durchaus möglich, mit guten Unterprogrammen ein Gesamtprogramm sehr effizient zu gestalten, bei zuträglicher Länge.

Für PL/65 gelten diselben Regeln wie für den 6500-Assembler, dem die Sprache ja nur vorgeschaltet ist:

1. Adreßmarken (Labels) dürfen bis zu 6 Zeichen lang sein, das erste muß ein Buchstabe sein.

2. A, P, S, X, Y dürfen nicht als Adreßmarken mit Länge eines Buchstabens benutzt werden, nicht einmal als einbuchstabige Laufvariable für FOR ... END; - WHILE ... END; oder IF ... THEN ... ELSE ... END; -Sequenzen. Und neu ist.

3. Auch 'ZZ' am Beginn von Adreßmarken ist nicht erlaubt. Diese behält sich der Compiler für von ihm generierte Zweizweigungsmarken vor.

Ganz anders als in BASIC, das Maschinenmodule gar nicht mag, ist es beim PL/65 oft unabdingbar notwendig, solche Module zu implementieren. Wie ich noch ausführen werde und wie man es im Manual feststellen kann, wird dies sehr erleichtert. Alles, was der Assembler kann, das kann auch PL/65, aber nicht alles was der Assembler nicht kann, kann das PL/65 auch nicht. Damit sind wir bei den

Einschränkungen für PL/65

Der Assembler kann nur addieren und subtrahieren, er führt aber keine Multiplikation und keine Division aus, auch keine Bool'sche Operation wie .AND, .OR und .EOR (den Punkt deshalb, weil er die Maschinenbefehle sehr wohl übersetzen kann) an Assembler-Variablen. Es ist deshalb müßig, solche Operationen, die während der Assemblierungsphase ausgeführt werden sollten, über PL/65 dem Assembler mitzuteilen, er kann es ja doch nicht! Das bedeutet: Assembler-Variable können auch über PL/65 nur addiert oder subtrahiert werden, wenn sichergestellt ist, daß alle dazu beitragenden Elemente während des zweiten Assemblerdurchlaufes schon Werte verkörpern. Zero Page-Variable müssen schon im ersten Assembler-PASS ihre Werte haben, deren spätere programmabhängige Inhalte unterliegen dieser Beschränkung nur hinsichtlich der Multiplikation und Division.

Indizierte Variablennamen sind auf beiden Seiten des Gleichheitszeichens möglich, während *indirekte Variablennamen* nur bei 1-Byte-Variablen auf beiden Seiten des Gleichheitszeichens möglich sind!

Die Operatoren für *indirekte Variablennamen* sind die jeweiligen Präfixe ' ' und '&', welche noch erklärt werden.

Die Operatoren für *indizierte Variablennamen* befinden sich ohne eingeschobenen Zwischenraum gleich nach dem Variablennamen zwischen einem Paar eckiger Klammern. Es ist ratsam, diesen

Klammerausdruck nicht allzusehr anwachsen zu lassen, da der daraus resultierende Maschinendode langwierig ist und weil sich seine Fehlerfreiheit nur sehr schwer überprüfen läßt.. Günstiger ist es, die Indizierung in einer vorher eingeschobenen Anweisungszeile aufzubereiten.

Bezüglich *indirekter Datenabarbeitung* gibt es noch zwei Feinheiten wie folgt:

Im Klammeraffen-Modus ( ) wird die Basisadresse in die Zero Page geholt und dann indiziert.

Im Ampersand-Modus (&) befindet sich die Basis-Adresse schon in er Zero Page, bereit zur Indizierung.

Zu diesem Zwecke legt PL/65 in der Zero Page 4 Register an, wovon zwei je 2 Bytes breit sind und zwei je 1 Byte. Es sind dies

```
R0 = 0      2-Byte-Register
R1 = R0+2   1-Byte-Register
R2 = R1+1   1-Byte-Register
R3 = R2+1   2-Byte-Register, was aus dieser Wertzuweisung kaum hervorgeht!
```

Unterprogramme (Subroutines), die im Akkumulator Daten hinterlassen oder dort vorzufinden hoffen, benötigen eine Dummy-Anweisung, damit die Daten für PL/65 greifbar werden. Vereinbarung: Assembler-Mnemonics zwischen Hochkommata (' ') bedeuten ein Literal, das dem Assembler direkt weitergegeben wird. Weiter:

1. Definition der Dummy-Location:
   DEF R4 = R3+2;
2. CALL hinterläßt Data im Akku:
   CALL READ; Zwei aufeinanderfolgende Tastatureingaben werden im Akkumulator
   CALL PACK; zu einer 2-stelligen Hex-Zahl gepackt und im Literal 'STA R4'
   CALL READ; dem PL/65 zur Weiterverarbeitung übergeben,
   CALL PACK; siehe Anweisung NMBR = R4.
   'STA R4';
   NMBR = R4;
3. CALL erwartet Data im Akku:
   R4 = NMBR; Der Inhalt von NMBR sollte als 2-stellige Hex-Zahl ausgegeben werden.
   CALL NUMA;

Interface-Probleme gibt es fast keine, solange man mit Registerbausteinen arbeitet, z.B. USART, ACIA, VIA, RIOT etc. ohne zusätzlichen Aufwand. Seitens des Assemblerprogrammes machte man es immer so, daß man deren Registern mnemonische Namen erteilte. Vereinbarungsgemäß stellen diese Register im 'memory mapped I/O' fixe Adressen dar, aus welchen ausgelesen oder in welche eingeschrieben wird.

Die Verarbeitungsbreite ist grundsätzlich 1 (!) Byte. Für Doppelbyte-Daten gibt es gewisse notwendige Erleichterungen im Zuge der Errechnung von Vektoren. Mehrfachbyte haben Kenner.

Es ist ratsam, besonders am Anfang, eine sorgfältige Analyse des generierten Assembler-Codes durchzuführen, um ein unverständliches Abstürzen des Programmes zu vermeiden. Ein besonders krasses Beispiel möge dies verdeutlichen, das auch die Notwendigkeit vor Augen führen sollte, Assembler-Module zu benutzen:

SHL SHIFT.4;
Dies bedeutet nicht: Shifte die nächsten 4 Bytes von SHIFT an um ein Byte nach links, nein, es bedeutet nur: shifte in der Location SHIFT um 4 Positionen nach links! Folgende Aufstellung möge dies verdeutlichen:

Beabsichtigte Operation	PL/65-codierte Operation	Die Ausführung entspricht
ASL SHIFT	LDA #4	ASL SHIFT
ROL SHIFT+1	TAY	ASL SHIFT
ROL SHIFT+2	ZZ0000 ASL SHIFT	ASL SHIFT
ROL SHIFT+3	DEY	ASL SHIFT
	BNE ZZ0000	

Wie man sieht, muß man diese und ähnliche Operationen von Hand codieren, da gibt es auch für Doppel-Byte-Daten keine Erleichterungen

Die Laufvariable der DO ... BY ... END; -Schleife (sie entspricht in BASIC dem FOR ... STEP ... NEXT) darf nur 1 Byte lang sein, ebenso wie die Werte der Anfangs- und Endvariablen!

Die Arbeitspraxis mit dem PL/65

Trotz aller zuvor geschilderten Einschränkungen ist mit der AIM 65 PL/65-Compiler ein liebes Werkzeug geworden, das mir sehr bei der schnellen Entwicklung von Maschinenprogrammen für die 6502 MPU hilft, da ich mich ausschließlich mit der Maschinenprogrammierung befasse. Mit seinen 8K Byte auf dem Steckplatz des BASIC leistet er mir große Dienste. Trotz meines nur 4K kleinen Arbeitsspeichers konnte ich Dank seiner Hilfe schon größere Programme durchführen. Wenn ich an andere MPL-Compiler denke, die für ihren Corpus 12-20K RAM und noch mehr (!) benötigen, kann ich meinen AIM-PL/65 nur noch loben, zumal er mir überdies Assembler-Quellencode liefert, auf den man noch platzsparender einwirken kann.

Wie schon zuvor beschrieben, übersetzt der PL/65 nur eine Kurzfassung des Problems in maschinengewohnte Umständlichkeit mit einem Minimum an Prüfungen, die sich nur an die Semantik des Ur-Programmes heranwagen, der Inhalt bleibt ungeprüft. Diese seine 'Untugend' nutze ich wie folgt voll aus:

Ich formuliere nur kleine, dafür übersichtliche Funktionsblöcke einer oder weniger Ebenen, nämlich wegen des in größerer Menge entstehenden Assembler-Quellencodes. Da PL/65 gar keine Notiz darüber führt, was gerade abgearbeitet wird, sondern nur Maschinenbefehls-Sequenzen zuzuordnen versucht und nur auf seine Blockgrenzen achtet, übersetzt er das Dargebotene anstandslos, eben nur auf seine BEGIN;-/END;-Paritäten bedacht. Diese Module sind kaum länger als 20-30 Programmzeilen und kaum größer als hex 10 Datablöcke nach dem AIM-Format auf Tonband. Der von PL/65 erzeugte Assembler-Quellencode paßt dann noch ziemlich genau in den Speicherbereich von 4A0 bis hex EFF. Von 200 bis 490 liegt ein noch zu beschreibendes s Dienstprogramm und von F00 bis FFF liegt mein VIC-1 Video-Monitor. Bei dieser Einteilung endet jedes in PL/65 geschriebene Modul mit der Paarung '.FIL Name'; EXIT;.

Die .FIL-Anweisung läßt den Assembler das nächste File assemblieren, während EXIT; den Kompilierungsvorgang beendet. Somit linke ich mittels des Assemblers die einzelnen PL/65-Ur-Module zu einem einzigen Assemblerprogramm zusammen. Zu diesem Zweck habe ich ein Dienstprogramm erstellt, das zwei Aufgaben löst und das im RAM-Bereich von hex 200 bis 490 liegt.

Dienstprogramm PLFOS

Dieses Dienstprogramm belegt neben den soeben erwähnten Zellen auch hex DF bis EE, die Adressen des Text-Editors. Es wird mit Taste F2 gestartet. Es hat zwei Teile: Der PL/65 Label Converter gestattet es, die mit ZZ beginnenen generierten Labels im Assembler-Quellcode auf Labels umzustellen, die mit 1-2 anderen Buchstaben beginnen. Der Zahlenteil des Labels wird nicht verändert.

Der Comments-Formatter sorgt für eine ungestörte Assembler-Liste. Alle Zeilen des Urtextes in PL/65 werden ja in die Assembler-Liste als Kommentar übernommen. Die im Urtext enthaltenen Semikolons bereiten normalerweise Fortmatierungsschwierigkeiten in dieser nächsten Phase. Es werden daher alle Semikolons, die sich innerhalb einer Zeile finden, durch 'zahme' Kommata substituiert, Semikolons am Zeilenende werden entfernt. Wichtig: Der Comments-Formatter verdoppelt das Anfangs-Semikolon einer jeden Kommentarzeile. Meine Formatierungsprogramme sind darauf abgestimmt, daß eine Kommentarzeile mit ' ;; ' beginnt, statt mit ' ;. '. Kommentare im Anschluß an Assembler-Instruktionen benötigen wie üblich nur 1 Semikolon. Bedienungsablauf:

1. Programm PLFOS laden
2. Editor-Bereich ab hex 4A0 definieren. Assembler-Quelltext im Editor bereitstellen.
3. Editor mit 'Q' oder 'ESC' verlassen. PLFOS mit Funtionstaste starten. Dabei sollten Drucker abgestellt sein. Man sieht die Abarbeitung des Quellentextes.
4. Den Prompt 'ZZ' mit Eingabe von 1-2 Buchstaben beantworten, die die Statthalter 'ZZ' ersetzen.
5. Das Programm gibt jede bearbeitete Zeile über OUTALL aus.
6. Das Programm startet selbständig die Kommentar-Formatierung und gibt wiederum alle bearbeiteten Zeilen über OUTALL aus.
7. 'EDITOR END' signalisiert das Ende der Textbearbeitung. Es tritt kein Mehrbedarf für Speicherplatz durch die o.a. Bearbeitung ein. Das bearbeitete Textfile nun auf externen Datenträger abspeichern.
8. Bei wiederholter Anwendung von PLFOS den Editor für die weiteren Textabschnitte jeneu initialisieren.

Beim Autor wurden weitere Programme für und in PL/65 entwickelt, so ein automatisches Assembler-Setup, Formatierer für die Assembler-Liste (für den Thermodrucker und auch für TTY), Ausgabe der Symboltafel sowie weitere Grunddienstleistungen, die man immer wieder benötigt und einfügen kann. Info unter Adresse Postfach 7 in A-5014 Salzburg oder Postfach 409 in D-8228 Freilassing.

```
) V.1 R.0 82.01.14 (
                0003          0003
0003            0004 ;;-------------------
0003            0005 ;;
0003            0006 ;; $$$$$$$$$$$$$$$$$$
0003            0007 ;; $                 $
0003            0008 ;; $     PL/65        $
0003            0009 ;; $     LABEL       $
0003            0010 ;; $   CONVERTER      $
0003            0011 ;; $      AND        $
0003            0012 ;; $   COMMENTS-      $
0003            0013 ;; $   FORMATTER      $
0003            0014 ;; $                 $
0003            0015 ;; $$$$$$$$$$$$$$$$$$
0003            0016 ;;
0003            0017 ;;-------------------
0003            0018 ;;
0003            0019 ;;  HOW TO USE IT:
0003            0020 ;;
0003            0021 ;;  1. INITIALIZE
0003            0022 ;;     EDITOR ABOVE
0003            0023 ;;     $4A0 = PL/65
0003            0024 ;;     TOP-ADDRESS
0003            0025 ;;
0003            0026 ;;  2. LOAD PL/65
0003            0027 ;;     ASSEMBLER-
0003            0028 ;;     FILE AS USUAL
0003            0029 ;;     BY EDITOR
0003            0030 ;;
0003            0031 ;;  3. EXIT EDITOR
0003            0032 ;;     BY <Q> OR BY
0003            0033 ;;     ESCAPE
0003            0034 ;;
0003            0035 ;;  4. START BY F2-
0003            0036 ;;     KEY, AFTER
0003            0037 ;;     END-MESSAGE
0003            0038 ;;     RE-ENTER BY
0003            0039 ;;     TYPING <T>
0003            0040 ;;
0003            0041 ;;-------------------
0003            0042
-------------------------------------------
0003            0044
0003            0045 CR     = 13
0003            0046 SPC    = 32
0003            0047
0003            0048        * = $DF
00DF            0049
00DF            0050 NOWLN  *=*+2
00E1            0051 BOTLN  *=*+2
00E3            0052 LOTXT  *=*+2
00E5            0053 TXTOP  *=*+2
00E7            0054 CURLN  *=*+2
00E9            0055 SRC    *=*+2
00EB            0056 DST    *=*+2
00ED            0057 NOSPC  *=*+1
00EE            0058 TEMP   *=*+1
00EF            0059
00EF            0060 COMIN  = $E1A1
00EF            0061 INALL  = $E993
00EF            0062 OUTALL = $E9BC
00EF            0063 STRING = $E7AF
00EF            0064
00EF            0065 M1     = $E000
00EF            0066 EDM1   = $E06C
00EF            0067 EDM2   = $E072
00EF            0068
00EF            0069 F2     = $10F
00EF            0070 START  = $300
00EF            0071
) PROGRAMMES ENTRY (
                00EF          0073
00EF            0074        * = F2
010F            0075
010F 4C0003     0076        JMP ENTRY
0112            0077
0112            0078        * = START
0300            0079
0300 78         0080 ENTRY  SEI
0301            0081
0301 18         0082        CLC
0302 D8         0083        CLD
0303 B8         0084        CLV
0304            0085
0304 A2FF       0086        LDX #255
0306 9A         0087        TXS
0307            0088
0307 A5E3       0089        LDA LOTXT+0
0309 85DF       0090        STA NOWLN+0
030B A5E4       0091        LDA LOTXT+1
030D 85E0       0092        STA NOWLN+1
030F            0093
030F A95A       0094        LDA #'Z'
0311 85E9       0095        STA SRC+0
0313 85EA       0096        STA SRC+1
```

```
0398 A5EC    0174        LDA DST+1
039A 91DF    0175        STA (NOWLN),Y
039C         0176
039C C8      0177        INY
039D C8      0178        INY
039E C8      0179        INY
039F C8      0180        INY
03A0         0181
03A0 C8      0182 INCR   INY
03A1 D0D0    0183        BNE LBLOOP
03A3         0184
03A3 205B04  0185 LINEND JSR BISPLY
03A6 206B04  0186        JSR NEWLIN
03A9 4C7103  0187        JMP LABLER
03AC         0188
03AC         0189
```

COMMENTS FORMATTER

```
             03AC        0191
03AC         0192 ;;
03AC         0193 ;;   COMMENTS-
03AC         0194 ;;   FORMATTER
03AC         0195 ;;
03AC A203    0196 SEMI   LDX #3
03AE 208704  0197        JSR CRLF
03B1         0198
03B1 A5E3    0199        LDA LOTXT+0
03B3 85DF    0200        STA NOWLN+0
03B5 A5E4    0201        LDA LOTXT+1
03B7 85E0    0202        STA NOWLN+1
03B9         0203
03B9 A000    0204 SEARCH LDY #00
03BB A200    0205        LDX #00
03BD         0206
03BD B1DF    0207 SRX    LDA (NOWLN),Y
03BF F076    0208        BEQ DONE
03C1         0209
03C1 C90D    0210        CMP #CR
03C3 F012    0211        BEQ LINDON
03C5         0212
03C5 C93B    0213        CMP #';'
03C7 D004    0214        BNE OTHER
03C9         0215
03C9 E8      0216        INX
03CA C8      0217        INY
03CB D0F0    0218        BNE SRX
03CD         0219
03CD C000    0220 OTHER  CPY #0
03CF D003    0221        BNE INCY
03D1 4C7604  0222        JMP NXTLIN
03D4         0223
03D4 C8      0224 INCY   INY
03D5 D0E6    0225        BNE SRX
03D7         0226
03D7 A900    0227 LINDON LDA #00
03D9 85ED    0228        STA NOSPC
03DB         0229
03DB 88      0230 DELPR  DEY
03DC D00B    0231        BNE TSTSPC
03DE         0232
03DE A5ED    0233        LDA NOSPC
03E0 D013    0234        BNE CHANGE
03E2         0235
03E2 A920    0236 SETSPC LDA #SPC
03E4 91DF    0237        STA (NOWLN),Y
03E6 4C2E04  0238        JMP GODIS
03E9         0239
03E9 B1DF    0240 TSTSPC LDA (NOWLN),Y
03EB C920    0241        CMP #SPC
03ED F010    0242        BEQ ISPC
03EF         0243
03EF C93B    0244        CMP #';'
03F1 F01D    0245        BEQ ISCL
03F3         0246
03F3 E6ED    0247        INC NOSPC
03F5         0248
03F5 B1DF    0249 CHANGE LDA (NOWLN),Y
03F7 C8      0250        INY
0315 85EB    0097        STA DST+0
0317 85EC    0098        STA DST+1
0319 A203    0099        LDX #3
031B 208704  0100        JSR CRLF
031E A5E9    0101        LDA SRC+0
0320 20BCE9  0102        JSR OUTALL
0323 A5EA    0103        LDA SRC+1
0325 20BCE9  0104        JSR OUTALL
0328 A920    0105        LDA #SPC
032A 20BCE9  0106        JSR OUTALL
032D A93D    0107        LDA #'='
032F 20BCE9  0108        JSR OUTALL
0332 A920    0109        LDA #SPC
0334 20BCE9  0110        JSR OUTALL
0337 A93F    0111        LDA #'?'
0339 20BCE9  0112        JSR OUTALL
033C A920    0113        LDA #SPC
033E 20BCE9  0114        JSR OUTALL
0341         0115
0341 2093E9  0116 GET    JSR INALL
0344 C90D    0117        CMP #CR
0346 F00B    0118        BEQ SET
0348         0119
0348 48      0120        PHA
0349 A5EC    0121        LDA DST+1
034B 85EB    0122        STA DST+0
034D 68      0123        PLA
034E 85EC    0124        STA DST+1
0350 4C4103  0125        JMP GET
0353         0126
0353 A920    0127 SET    LDA #SPC
0355 20BCE9  0128        JSR OUTALL
0358 A93D    0129        LDA #'='
035A 20BCE9  0130        JSR OUTALL
035D A920    0131        LDA #SPC
035F 20BCE9  0132        JSR OUTALL
0362 A5EB    0133        LDA DST+0
0364 20BCE9  0134        JSR OUTALL
0367 A5EC    0135        LDA DST+1
0369 20BCE9  0136        JSR OUTALL
036C A203    0137        LDX #3
036E 208704  0138        JSR CRLF
```

LABLE CHANGER

```
             0371        0140
0371         0141 ;;
0371         0142 ;;   LABLE-CHANGER:
0371         0143 ;;
0371 A000    0144 LABLER LDY #00
0373         0145
0373 B1DF    0146 LBLOOP LDA (NOWLN),Y
0375 F035    0147        BEQ SEMI
0377         0148
0377 C90D    0149        CMP #CR
0379 F028    0150        BEQ LINEND
037B         0151
037B C5E9    0152        CMP SRC+0
037D D021    0153        BNE INCR
037F         0154
037F C8      0155        INY
0380 B1DF    0156        LDA (NOWLN),Y
0382 C5EA    0157        CMP SRC+1
0384 D01A    0158        BNE INCR
0386         0159
0386 C8      0160        INY
0387 B1DF    0161        LDA (NOWLN),Y
0389 C92F    0162        CMP #47
038B 9013    0163        BCC INCR
038D         0164
038D C939    0165        CMP #57
038F B00F    0166        BCS INCR
0391         0167
0391 88      0168        DEY
0392 88      0169        DEY
0393         0170
0393 A5EB    0171        LDA DST+0
0395 91DF    0172        STA (NOWLN),Y
0397 C8      0173        INY
```

```
03F8 91DF    0251         STA (NOWLN),Y
03FA 88      0252         DEY
03FB D0DE    0253         BNE DELPR
03FD F02F    0254         BEQ GODIS
03FF         0255
03FF C8      0256 ISPC    INY
0400 B1DF    0257         LDA (NOWLN),Y
0402 C90D    0258         CMP #CR
0404 F005    0259         BEQ ISEND
0406         0260
0406 88      0261         DEY
0407 D0EC    0262         BNE CHANGE
0409 F0D7    0263         BEQ SETSPC
040B         0264
040B 88      0265 ISEND   DEY
040C D0CD    0266         BNE DELPR
040E F0D2    0267         BEQ SETSPC
0410         0268
0410 CA      0269 ISCL    DEX
0411 D010    0270         BNE SUBST
0413         0271
0413 A5ED    0272         LDA NOSPC
0415 F0CB    0273         BEQ SETSPC
0417         0274
0417 A000    0275         LDY #00
0419 A93B    0276         LDA #';'
041B 91DF    0277         STA (NOWLN),Y
041D C8      0278         INY
041E 91DF    0279         STA (NOWLN),Y
0420 4C2E04  0280         JMP GODIS
0423         0281
0423 A5ED    0282 SUBST   LDA NOSPC
0425 F0B4    0283         BEQ DELPR
0427         0284
0427 A92C    0285         LDA #','
0429 91DF    0286         STA (NOWLN),Y
042B 4CF503  0287         JMP CHANGE
042E         0288
042E 205B04  0289 GODIS   JSR DISPLY
0431 206B04  0290         JSR NEWLIN
0434 4CB903  0291         JMP SEARCH
0437         0292
0437 A90D    0293 DONE    LDA #CR
0439 20BCE9  0294         JSR OUTALL
043C A920    0295         LDA #32
043E 20BCE9  0296         JSR OUTALL
0441 20BCE9  0297         JSR OUTALL
0444 A06C    0298         LDY #EDM1-M1
0446 20AFE7  0299         JSR STRING
0449 A920    0300         LDA #SPC
044B 20BCE9  0301         JSR OUTALL
044E A072    0302         LDY #EDM2-M1
0450 20AFE7  0303         JSR STRING
0453 A203    0304         LDX #3
0455 208704  0305         JSR CRLF
0458 4CA1E1  0306         JMP COMIN
045B         0307

}}---UTILITIES---{{------------------------------------------

             045B         0309
045B         0310 ;;
045B         0311 ;;   SUBROUTINES:
045B         0312 ;;
045B A000    0313 DISPLY LDY #00
045D         0314
045D B1DF    0315 DISLPR LDA (NOWLN),Y
045F 48      0316         PHA
0460 20BCE9  0317         JSR OUTALL
0463 C8      0318         INY
0464 68      0319         PLA
0465 C90D    0320         CMP #CR
0467 D0F4    0321         BNE DISLPR
0469         0322
0469 88      0323         DEY
046A 60      0324         RTS
046B         0325
046B 38      0326 NEWLIN SEC
046C 98      0327         TYA
046D 65DF    0328         ADC NOWLN+0
046F 85DF    0329         STA NOWLN+0
0471 9002    0330         BCC LINOK
0473         0331
0473 E6E0    0332         INC NOWLN+1
0475         0333
0475 60      0334 LINOK   RTS
0476         0335
0476 A000    0336 NXTLIN LDY #00
0478 A90D    0337         LDA #CR
047A         0338
047A D1DF    0339 NXTLPR CMP (NOWLN),Y
047C F003    0340         BEQ CRFND
047E         0341
047E C8      0342         INY
047F D0F9    0343         BNE NXTLPR
0481         0344
0481 206B04  0345 CRFND  JSR NEWLIN
0484 4CB903  0346         JMP SEARCH
0487         0347
0487 A90D    0348 CRLF    LDA #CR
0489 20BCE9  0349         JSR OUTALL
048C CA      0350         DEX
048D D0FB    0351         BNE CRLF
048F         0352
048F 60      0353         RTS
0490         0354
0490         0355 FIN     .END
  ERRORS= 000
```

#

Ing. grad. Horst Steder, 6000 Frankfurt

BASIC-Compactor

Im MICRO MAG und anderen Zeitschriften wurden schon verschiedentlich kleine BASIC-Routinen veröffentlicht, die eine Komprimierung von BASIC-Programmen durchführen, die also alle Zwischenräume und REMARK-Statements entfernen. Ein sehr ausführlich kommentierter Artikel (1) brachte mich auf die Idee, eine solche Routine in Assembler zu schreiben. Dies allein würde nun, außer der durch die Maschinensprache bedingten Erhöhung der Geschwindigkeit, keinen neuen Beitrag ergeben. Daher besteht dieses Programm aus zwei Teilen: Im ersten Teil werden alle Zwischenräume und REM-Statements entfernt und die die Programmzeilen verbindenden 'Links' entsprechend korrigiert.

Der zweite Teil, der wahlweise aufgerufen werden kann, geht nun erheblich weiter, indem er hier alle nicht benötigten Zeilennummern entfernt und die entsprechenden Zeilen zusammenfügt. Wo immer möglich, werden also zur Trennung der einzelnen Anweisungen nicht mehr Zeilennummern, sondern Doppelpunkte verwendet, die ja der Trennung bei Mehrfachanweisungen in einer Zeile dienen. Es muß natürlich geprüft werden, ob Zeilen durch GOTO oder GOSUB usw. angesprungen werden. In diesen Fällen dürfen die Zeilennummern nicht entfernt werden. Außerdem müssen die Zeilen nach IF und ON ... ebenfalls erhalten bleiben, weil der Interpreter in der Folgezeile weiterfährt, wenn die Bedingung nicht eintrat.

Die vorgenannten Verfahren führen normalerweise zu einer drastischen Einsparung des benötigten Speicherraumes und als Zugabe auch zu einer kleinen Geschwindigkeitserhöhung. Das ist interessant, wenn z.B. ein BASIC-Programm in ein EPROM gebrannt werden soll. Allerdings ergeben sich durch die Zusammenfügung der Zeilen einige Besonderheiten, auf die später eingegangen wird.

Das nachstehende Programm wurde für den AIM 65 geschrieben. Es läßt sich aber sehr leicht für andere Rechner mit Microsoft-BASIC anpassen. Zur Anwendung des Programmes folgende Hinweise:

1) Das zu komprimierende BASIC-Programm muß syntaktisch fehlerfrei sein.
2) Es dürfen keine Sprünge auf REMARK-Zeilen existieren.
3) Strings, die mit einem Anführungszeichen beginnen, sollten auch mit einem solchen abgeschlossen sein. Das ist zwar nicht unbedingt notwendig, wenn nur der erste Teil des Programmes angewandt wird (hier werden schlimmstenfalls dann nur einige Spaces nicht entfernt). Bei der Anwendung des zweiten Teiles kann das Fehlen des abschließenden Anführungszeichens jedoch zur fehlerhaften Entfernung von Zeilennummern und Links führen.
4) Selbstverständlich (aber so etwas ist nach den Gesetzen von Murphy schon oft vergessen worden !): Vor dem Start des Compactors sollte man das Original-Programm mit den Kommentaren auf Tape oder Floppy retten!

Der Compactor wird mit der Funktionstaste (F3) gestartet und beendet den ersten Durchlauf mit dem Prompt 'PASS2 ?'. An dieser Stelle kann mit jeder Taste außer 'Y' abgebrochen werden. Jetzt kann man auch noch das BASIC-Programm testen und editieren, nachdem es keine REMARKs und Spaces mehr enthält. - Wird der Prompt jedoch mit 'Y' beantwortet, so erfolgt auch die Entfernung der nicht benötigten Zeilennummern.

Folgende Besonderheiten sind zu beachten: Die zusammengefügten Zeilen können, je nach Programm, länger als 72 oder als sogar 255 Zeichen werden. Das Programm kann dann nicht mehr mit dem SAVE-Befehl abgespeichert, bzw. nicht mehr mit LOAD eingelesen werden. Im zweiten Fall ist auch ein Auslisten mit LIST nicht mehr möglich. Man speichert daher ein so komprimiertes Programm mit dem DUMP-Befehl des AIM-Monitors wie folgt ab:

(D)
FROM=73 TO=80 (BASIC-Pointer!)
MORE?Y
FROM=xxxx TO=yyyy (xxxx ist dabei die Startadresse des Programmes wie im Pointer $ 73/74 hinterlegt. yyyy ist die Endadresse in $ 75/76.

Zum Laden wird zuerst das BASIC mit (5) initialisiert. Man springt dann in den Monitor zurück und liest das Programm mit (L) (LOAD des Monitors) ein. Das hat auch den Vorteil, daß es viel schneller als mit dem LOAD unter BASIC geht.

Einzelheiten über die Arbeitsweise des Compactors können den ausführlichen Kommentaren entnommen werden.

Bei dieser Gelegenheit sei noch ein Tip angebracht: Es wurde schon mehrfach vorgeschlagen, den AIM-Editor zum Ändern von BASIC-Zeilen zu verwenden. Da aber der BASIC-Buffer eine Zeilenlänge von 72 Zeichen, der Editor demgegenüber nur 60 Zeichen je Zeile akzeptiert, können beim Transfer Zeichen verschwinden, ohne daß eine Fehlermeldung entsteht. Beim Betrieb eines solchen Programmes ist es dann manchmal mühsam herauszufinden, daß aus einem GOTO 2080 ein GOTO 20 oder daß aus einem PRINT A*B+C ein PRINT A*B wurde.

Literatur: (1) Edward H. Carlson 'Program Compactor', COMPUTE, April 1981.

```
0000            0000 ; ***************************************************
0000            0001 ; *       BASIC  COMPACTOR       VERS.  3.3       *
0000            0002 ; *           FOR AIM-BASIC VERS. 1.1              *
0000            0003 ; *           COPYRIGHT BY H.STEDER  11/81         *
0000            0004 ; ***************************************************
0000            0005
0000            0006 ; THIS ROUTINE COMPACTS A BASIC PROGRAM BY RE-
0000            0007 ; MOVING ALL SPACES AND 'REM' LINES IN THE FIRST
0000            0008 ; PASS. IN THE SECOND PASS IT REMOVES ALL UNNECES-
0000            0009 ; SARY LINE NUMBERS AND CONCATENATES THESE LINES.
0000            0010
0000            0011 ; CHARACTERS AND TOKENS TO BE SEARCHED FOR
0000            0012 SPACE   =32
0000            0013 QUOTE   =34
0000            0014 COLON   =58
0000            0015 GOTO    =$88
0000            0016 IF      =$8A
0000            0017 GOSUB   =$8C
0000            0018 REM     =$8E
0000            0019 ON      =$90
0000            0020 TO      =$9E
0000            0021 THEN    =$A1
0000            0022 GO      =$C5
0000            0023 ; POINTERS FOR COMPACTING LOOP
0000            0024 PNTR1   =$C6            ;ALSO POINTER FOR 'CHRGET'
0000            0025         *=*+$20
0020            0026 PNTR2   *=*+2
0022            0027 PNTR3   *=*+2
0024            0028 PNTR4   *=*+2
0026            0029 PSFLG   *=*+1
0027            0030 ; BASIC POINTERS
0027            0031 HXNUM   =$14            ;HEX VALUE RETURNED BY CONVRT
0027            0032 BASTRT  =$73            ;START ADDR OF BASIC PROGRAM
0027            0033 VARST   =$75            ;START ADDR OF VARIABLE-SPACE
0027            0034 ; BASIC AND MONITOR ROUTINES
0027            0035 CHRGET  =$00BF          ;CHAR-GET ROUTINE
0027            0036 CLRVAL  =$B47C          ;CLEAR VARIABLES + POINTERS
0027            0037 CONVRT  =$B7DA          ;CONVERTS NUMBER STRING TO HEX
0027            0038 AIM65   =$E1A1
0027            0039 PROMPT  =$E7CA
0027            0040 RDRUB   =$E95F
0027            0041 CRLF    =$E9F0
0027            0042 OUTDIS  =$EF02
0027            0043
0027            0044         *=$112          ;FUNCTION KEY <F3>
0112 4C 00 50   0045         JMP PASS1
0115            0046
```

```
0115                    0047          *=$5000          ;**START OF ROUTINE
5000                    0048
5000 A9 00              0049 PASS1    LDA #0           ;CLEAR PASS FLAG
5002 85 26              0050          STA PSFLG
5004 20 6A 51           0051          JSR SETBEG       ;BASIC START --> POINTERS
5007 4C 44 50           0052          JMP LP3+6        ;FIRST START WITH 'REM'-TEST
500A                    0053
500A                    0054 ; HERE THE WALK-THROUGH LOOP STARTS. PNTR1 IS
500A                    0055 ; THE POINTER TO THE CHARACTERS IN THE ORIGINAL
500A                    0056 ; PROGRAM, PNTR2 IS THE POINTER TO THE NEW LO-
500A                    0057 ; CATIONS OF THE COMPACTED PROGRAM.
500A                    0058
500A B1 C6              0059 LOOP1    LDA (PNTR1),Y
500C F0 32              0060          BEQ LP3+2        ;IS TERMINATION OF LINE
500E C9 22              0061          CMP #QUOTE       ;A QUOTED STRING ?
5010 D0 0F              0062          BNE TST1
5012 91 20              0063          STA (PNTR2),Y
5014 20 13 51           0064 QUOT     JSR SHIFT        ;THIS LOOP TRANSFERS THE
5017 B1 C6              0065          LDA (PNTR1),Y    ;STRING IN QUOTES WITHOUT
5019 91 20              0066          STA (PNTR2),Y    ;CHANGING ANYTHING, UNTIL
501B F0 27              0067          BEQ LP3+6        ;EITHER THE LINE END (=00)
501D C9 22              0068          CMP #QUOTE       ;OR A SECOND QUOTE
501F D0 F3              0069          BNE QUOT         ;HAS BEEN FOUND
5021 C9 8E              0070 TST1     CMP #REM
5023 D0 09              0071          BNE TST2
5025 20 19 51           0072 RLP      JSR SHFT1        ;FOUND 'REM' IN LINE,
5028 B1 C6              0073          LDA (PNTR1),Y    ;SKIP IT AND REST OF LINE
502A D0 F9              0074          BNE RLP
502C F0 12              0075          BEQ LP3+2
502E C9 20              0076 TST2     CMP #SPACE
5030 F0 07              0077          BEQ LP2
5032 20 13 51           0078          JSR SHIFT        ;NO SPACE OR 'REM', JUST
5035 90 07              0079          BCC LP3          ;INCREMENT BOTH POINTERS
5037 B0 2B              0080          BCS END          ;FOUND END, TERMINATE PASS 1

5039 20 19 51           0081 LP2      JSR SHFT1        ;WAS SPACE, INCR PNTR1 ONLY
503C B0 26              0082          BCS END
503E B1 C6              0083 LP3      LDA (PNTR1),Y    ;TRANSFER A CHARACTER FROM
5040 91 20              0084          STA (PNTR2),Y    ;OLD TO NEW LOCATION
5042 D0 CA              0085          BNE LOOP1+4      ;NOT YET AT LINE END
5044 A0 05              0086          LDY #5           ;REACHED LINE END, NOW
5046 B1 C6              0087          LDA (PNTR1),Y    ;CHECK WHETHER NEXT LINE
5048 C9 8E              0088          CMP #REM         ;STARTS WITH 'REM'
504A D0 08              0089          BNE NXTL
504C 20 19 51           0090 RLP1     JSR SHFT1        ;IS 'REM', INCR PNTR1 ONLY
504F 88                 0091          DEY              ;TO DELETE COMPLETE LINE
5050 D0 FA              0092          BNE RLP1         ;INCLUDING LINK AND LINE-#
5052 F0 B6              0093          BEQ LOOP1
5054 A2 05              0094 NXTL     LDX #5           ;TRANSFER LINK AND LINE-#
5056 A0 00              0095          LDY #0           ;TO NEW LOCATION
5058 20 13 51           0096          JSR SHIFT
505B B1 C6              0097          LDA (PNTR1),Y
505D 91 20              0098          STA (PNTR2),Y
505F CA                 0099          DEX
5060 D0 F6              0100          BNE NXTL+4
5062 F0 A6              0101          BEQ LOOP1        ;BACK TO LOOP
5064                    0102
5064                    0103 ; WALKED THROUGH COMPLETE PROGRAM, NOW CORRECT
5064                    0104 ; ALL NECESSARY LINKS AND POINTERS
5064                    0105
5064 20 99 51           0106 END      JSR LNKADJ       ;CORRECT ALL LINKS
5067 20 19 51           0107          JSR SHFT1        ;INCREMENT VALUE OF LAST
506A 20 19 51           0108          JSR SHFT1        ;LINK BY 2 TO GET NEW VALUE
506D A5 C6              0109          LDA PNTR1        ;FOR THE START OF VARIABLES
506F 85 75              0110          STA VARST
5071 A5 C7              0111          LDA PNTR1+1
5073 85 76              0112          STA VARST+1
5075 20 7C B4           0113          JSR CLRVAL       ;RESET ALL VARIABLE POINTERS
5078 A5 26              0114 END1     LDA PSFLG        ;IN WHICH PASS ?
507A F0 03              0115          BEQ PASS2        ;WAS FIRST PASS, GO TO PASS2
507C 4C A1 E1           0116          JMP AIM65        ;ALL OVER, BACK TO MONITOR
507F                    0117
```

```
507F                0118 ; ------------------------------------------
507F                0119 ; IN THE SECOND PASS REMOVAL OF ALL UNNECESSARY
507F                0120 ; LINE NUMBERS AND CONCATENATION IS PERFORMED.
507F                0121 ; ------------------------------------------
507F                0122
507F A0 08          0123 PASS2  LDY #8
5081 B9 E6 51       0124        LDA TXTB-1,Y      ;DISPLAY 'PASS2 ?'
5084 20 02 EF       0125        JSR OUTDIS
5087 88             0126        DEY
5088 D0 F7          0127        BNE PASS2+2
508A 20 5F E9       0128        JSR RDRUB         ;ASK FOR "Y"
508D C9 59          0129        CMP #'Y'
508F D0 EB          0130        BNE END1+4        ;NO 'Y', TERMINATE
5091 20 CA E7       0131        JSR PROMPT        ;SHOW '*' AND START PASS2
5094 E6 26          0132        INC PSFLG         ;SET PASS FLAG
5096 20 6A 51       0133        JSR SETBEG        ;BASIC START --> LOOP POINTERS
5099 20 83 51       0134        JSR SETTAB        ;SET POINTERS OF NUMBER TABLE
509C A0 03          0135        LDY #3            ;START LINE-# IS ALWAYS
509E 20 C6 51       0136        JSR SVNUM         ;STORED AS FIRST IN TABLE
50A1 20 90 51       0137 LOOP2  JSR XSHIFT        ;SKIP OVER LINK & LINE-#
50A4 B0 45          0138        BCS PASS21        ;AT END OF PROGRAM
50A6 20 BF 00       0139        JSR CHRGET        ;GET CHARACTER IN LINE
50A9 C9 00          0140        CMP #0            ;END OF LINE ?
50AB F0 F4          0141        BEQ LOOP2         ;YES, GET NEXT LINE
50AD C9 88          0142        CMP #GOTO         ;NOW CHECK FOR LINE-#
50AF F0 17          0143        BEQ GETNUM        ;GET NUMBER AND STORE IN TABLE
50B1 C9 8C          0144        CMP #GOSUB
50B3 F0 13          0145        BEQ GETNUM
50B5 C9 A1          0146        CMP #THEN
50B7 F0 0F          0147        BEQ GETNUM
50B9 C9 8A          0148        CMP #IF           ;AT 'IF' STORE # OF NEXT LINE
50BB F0 20          0149        BEQ IFLP
50BD C9 90          0150        CMP #ON           ;DO THE SAME FOR 'ON'
50BF F0 1C          0151        BEQ IFLP
50C1 C9 C5          0152        CMP #GO           ;'GO TO' ALSO ALLOWED
50C3 D0 E1          0153        BNE LOOP2+5       ;NONE OF THESE, GET NEXT
50C5 20 BF 00       0154        JSR CHRGET        ;NEXT CHAR MUST BE 'TO'-TOKEN
50C8                0155
50C8 20 BF 00       0156 GETNUM JSR CHRGET
50CB B0 DC          0157        BCS LOOP2+8       ;IS NOT A NUMBER
50CD 20 DA B7       0158        JSR CONVRT        ;GET AND CONVERT LINE-#
50D0 48             0159        PHA               ;SAVE NEXT CHARACTER
50D1 20 D1 51       0160        JSR INTAB         ;CONVERTED LINE-# --> TABLE
50D4 68             0161        PLA
50D5 F0 CA          0162        BEQ LOOP2         ;END OF LINE
50D7 C9 2C          0163        CMP #','          ;WAS NEXT CHAR A COMMA?
50D9 F0 ED          0164        BEQ GETNUM        ;YES, GET NEXT NUMBER
50DB D0 C9          0165        BNE LOOP2+5       ;JMP ALWAYS
50DD                0166
50DD C8             0167 IFLP   INY               ;FOUND AN "IF" OR "ON", MOVE
50DE B1 C6          0168        LDA (PNTR1),Y     ;TO END OF CURRENT LINE
50E0 D0 FB          0169        BNE IFLP
50E2 C8             0170        INY
50E3 C8             0171        INY
50E4 C8             0172        INY               ;NOW GET FOLLOWING LINE-#
50E5 20 C6 51       0173        JSR SVNUM         ;AND STORE IT IN TABLE
50E8 4C A6 50       0174        JMP LOOP2+5
50EB                0175
50EB                0176 ; REPLACE ALL LINE NUMBERS NOT FOUND IN THE
50EB                0177 ; TABLE (PLUS THEIR LINKS) BY SPACES
50EB 20 6A 51       0178 PASS21 JSR SETBEG        ;GO TO START OF BASIC
50EE 20 90 51       0179        JSR XSHIFT        ;SKIP OVER LINK & LINE-#
50F1 20 19 51       0180 PS21   JSR SHFT1         ;MOVE TO NEXT LOCATION
50F4 90 03          0181        BCC *+5
50F6 4C 04 50       0182        JMP PASS1+4       ;AT PROGRAM END JUMP BACK TO
50F9                0183                          ;REMOVE ALL GENERATED SPACES
50F9 B1 C6          0184        LDA (PNTR1),Y     ;LOOP UNTIL END OF LINE
50FB D0 F4          0185        BNE PS21
50FD A5 75          0186        LDA VARST         ;SET POINTER TO TABLE START
50FF 85 22          0187        STA PNTR3
5101 A5 76          0188        LDA VARST+1
5103 85 23          0189        STA PNTR3+1
5105 20 2A 51       0190        JSR COMPAR        ;SEARCH FOR LINE-# IN TABLE
5108 90 E4          0191        BCC PASS21+3      ;FOUND IT, SKIP OVER
510A A9 3A          0192        LDA #COLON        ;NOT FOUND, REPLACE '00' BY A
510C 91 C6          0193        STA (PNTR1),Y     ;COLON AS INSTRUCTION DELIMITER
```

```
510E 20 5D 51  0194          JSR NSPACE        ;REPL. LINK+LINE-# BY SPACES
5111 F0 DE     0195          BEQ PS21          ;JMP ALWAYS
5113           0196
5113           0197 ; INCREMENT BOTH POINTERS AND CHECK FOR END
5113 E6 20     0198 SHIFT    INC PNTR2
5115 D0 02     0199          BNE *+4
5117 E6 21     0200          INC PNTR2+1
5119 E6 C6     0201 SHFT1    INC PNTR1         ;ENTRY, IF ONLY ORIGIN-
511B D0 02     0202          BNE *+4           ;POINTER MUST BE INCREMENTED
511D E6 C7     0203          INC PNTR1+1
511F A5 C7     0204          LDA PNTR1+1       ;CHECK, WHETHER END HAS
5121 C5 76     0205          CMP VARST+1       ;BEEN REACHED
5123 90 04     0206          BCC *+6
5125 A5 C6     0207          LDA PNTR1         ;IF YES, CARRY IS SET
5127 C5 75     0208          CMP VARST
5129 60        0209          RTS
512A           0210
512A           0211 ; SEARCH FOR MATCH WITH LINE-# IN TABLE
512A A0 03     0212 COMPAR   LDY #3            ;(IF MATCH --> CARRY CLEAR)
512C A2 00     0213          LDX #0
512E B1 C6     0214          LDA (PNTR1),Y     ;GET LINE-# LOW IN BASIC
5130 C1 22     0215          CMP (PNTR3,X)     ;COMPARE WITH TABLE VALUE
5132 F0 0A     0216          BEQ CMP1
5134 20 4C 51  0217          JSR INCP          ;INCREMENT TABLE POINTER
5137 20 4C 51  0218 CMP0     JSR INCP
513A 90 EE     0219          BCC COMPAR
513C B0 08     0220          BCS CRET          ;REACHED END OF TABLE
513E 20 4C 51  0221 CMP1     JSR INCP
5141 C8        0222          INY
5142 B1 C6     0223          LDA (PNTR1),Y     ;NOW COMPARE HI-BYTES
5144 C1 22     0224          CMP (PNTR3,X)
5146 D0 EF     0225          BNE CMP0
5148 18        0226          CLC               ;MATCH, RETURN WITH CARRY CLEAR
5149 A0 00     0227 CRET     LDY #0            ;RESET Y-REG
514B 60        0228          RTS
514C           0229
514C           0230 ; INCREMENT TABLE POINTER AND CHECK FOR TABLE END
514C E6 22     0231 INCP     INC PNTR3
514E D0 02     0232          BNE *+4
5150 E6 23     0233          INC PNTR3+1
5152 A5 23     0234          LDA PNTR3+1
5154 C5 25     0235          CMP PNTR4+1       ;TABLE END IS IN PNTR4
5156 90 04     0236          BCC *+6
5158 A5 22     0237          LDA PNTR3
515A C5 24     0238          CMP PNTR4
515C 60        0239          RTS
515D           0240
515D           0241 ; REPLACE LINK & LINE-# WITH SPACES
515D A2 04     0242 NSPACE   LDX #4
515F 20 19 51  0243          JSR SHFT1
5162 A9 20     0244          LDA #' '
5164 91 C6     0245          STA (PNTR1),Y
5166 CA        0246          DEX
5167 D0 F6     0247          BNE NSPACE+2
5169 60        0248          RTS
516A           0249
516A A5 73     0250 SETBEG   LDA BASTRT        ;START OF BASIC PROGRAM
516C 85 C6     0251          STA PNTR1         ;INTO THE LOOP POINTERS
516E 85 20     0252          STA PNTR2
5170 A5 74     0253          LDA BASTRT+1
5172 85 C7     0254          STA PNTR1+1
5174 85 21     0255          STA PNTR2+1
5176 A5 C6     0256          LDA PNTR1         ;DECREMENT POINTERS TO CATCH
5178 D0 04     0257          BNE *+6           ;LEADING 'REM'-ONLY LINES
517A C6 C7     0258          DEC PNTR1+1
517C C6 21     0259          DEC PNTR2+1
517E C6 C6     0260          DEC PNTR1
5180 C6 20     0261          DEC PNTR2
5182 60        0262          RTS
5183           0263
5183 A5 75     0264 SETTAB   LDA VARST         ;TABLE FOR LINE NUMBERS
5185 85 22     0265          STA PNTR3         ;STARTS IN VARIABLE SPACE
5187 85 24     0266          STA PNTR4
5189 A5 76     0267          LDA VARST+1
518B 85 23     0268          STA PNTR3+1
518D 85 25     0269          STA PNTR4+1
518F 60        0270          RTS
5190           0271
```

```
5190 A2 04     0272 XSHIFT LDX #4          ;SKIP OVER NEXT 4 LOCS
5192 20 19 51  0273        JSR SHFT1
5195 CA        0274        DEX
5196 D0 FA     0275        BNE XSHIFT+2
5198 60        0276        RTS
5199           0277
5199           0278 ; ADJUST THE LINKS OF THE CONCATENATED LINES.
5199           0279 ; NOTE: CANNOT BE DONE WITH A RUNNING INDEX, SINCE
5199           0280 ; THE NEW LINES MAY BE LONGER THAN 255 CHARACTERS.
5199           0281
5199 20 6A 51  0282 LNKADJ JSR SETBEG
519C 20 13 51  0283        JSR SHIFT
519F A0 01     0284 LNK1   LDY #1
51A1 B1 C6     0285        LDA (PNTR1),Y
51A3 F0 41     0286        BEQ LNK3        ;IS END OF BASIC PROGRAM
51A5 A5 C6     0287        LDA PNTR1       ;STORE LINK FOR NEXT LINE
51A7 85 20     0288        STA PNTR2       ;IN CURRENT POINTER
51A9 A5 C7     0289        LDA PNTR1+1
51AB 85 21     0290        STA PNTR2+1
51AD 20 90 51  0291        JSR XSHIFT      ;SKIP OVER LINE-# & LINK
51B0 88        0292        DEY
51B1 20 19 51  0293 LNK2   JSR SHFT1       ;SEARCH FOR END OF LINE
51B4 B1 C6     0294        LDA (PNTR1),Y
51B6 D0 F9     0295        BNE LNK2
51B8 20 19 51  0296        JSR SHFT1       ;INCR. TO NEXT LOCATION
51BB A5 C6     0297        LDA PNTR1
51BD 91 20     0298        STA (PNTR2),Y   ;SET POINTER VALUE INTO LINK
51BF C8        0299        INY
51C0 A5 C7     0300        LDA PNTR1+1
51C2 91 20     0301        STA (PNTR2),Y
51C4 D0 D9     0302        BNE LNK1        ;BACK TO LOOP UNTIL END
51C6           0303
51C6           0304 ; STORE ALL LINE-NUMBERS BEING REFERRED TO
51C6           0305 ; IN THE BASIC PROGRAM. NO CHECK IS MADE FOR
51C6           0306 ; AVAILABLE RAM-SPACE  (TOP OF USED MEM).
51C6           0307
51C6 B1 C6     0308 SVNUM  LDA (PNTR1),Y   ;INTERMEDIATE STORAGE
51C8 85 14     0309        STA HXNUM       ;OF (HEX-) LINE-#
51CA C8        0310        INY
51CB B1 C6     0311        LDA (PNTR1),Y
51CD 85 15     0312        STA HXNUM+1
51CF A0 00     0313        LDY #0          ;RESET Y-REGISTER
51D1 A5 14     0314 INTAB  LDA HXNUM       ;STORE LINE NUMBER IN
51D3 91 24     0315        STA (PNTR4),Y   ;HEX FORMAT INTO TABLE
51D5 A5 15     0316        LDA HXNUM+1
51D7 C8        0317        INY
51D8 91 24     0318        STA (PNTR4),Y
51DA 88        0319        DEY
51DB 18        0320 INTB   CLC
51DC A9 02     0321        LDA #2          ;INCREMENT TABLE POINTER
51DE 65 24     0322        ADC PNTR4       ;BY TWO LOCATIONS
51E0 85 24     0323        STA PNTR4
51E2 90 02     0324        BCC *+4
51E4 E6 25     0325        INC PNTR4+1
51E6 60        0326 LNK3   RTS
51E7           0327
51E7 20 3F     0328 TXTB   .BYT ' ? 2SSAP'
51EF           0329
```

#

Bücher

Schneider, Wolfgang: 'Einführung in PASCAL' in der Serie 'Programmierung von Mikrocomputern' (4), Verlag Vieweg, Braunschweig/Wiesbaden 1982, ISBN 3-528-04200-1, 155 Seiten, DM 24,--.

Nach seinen Büchern 'Einführung in BASIC' und 'BASIC für Fortgeschrittene' legt der Autor nunmehr eine bewußt knapp gehaltene (den Anfänger nicht durch Stoff-Fülle verwirrende) Einführung in PASCAL vor. Er vermeidet dabei den Gebrauch englischer Fachausdrücke und formuliert präzise deutsch. Wiederum ist die Darbietung des Stoffes klar und auch optisch gegliedert, am Schluß jeden Kapitels findet man graphisch hervorgehobene Zusammenfassungen, die ein Nachschlagen erleichtern. Im Anschluß an die einführenden Abschnitte findet man folgende Kapitel: Problemaufbereitung und Darstellung von Programmabläufen, Schreiben von PASCAL-Primärprogrammen, PASCAL-Sprachelemente, Der PASCAL-Vereinbarungsteil, Die arithmetische Zuordnungsanweisung, Ein- und Ausgabeanweisung, Steuerungsanweisungen, Kommentare im Programm, Fehlerbehandlung, 10 vollständig programmierte Beispiele. - In der Summe ist das Buch wegen seiner didaktischen Stoffaufbereitung empfehlenswert.

#

Bernhard Kokula, 6700 Ludwigshafen

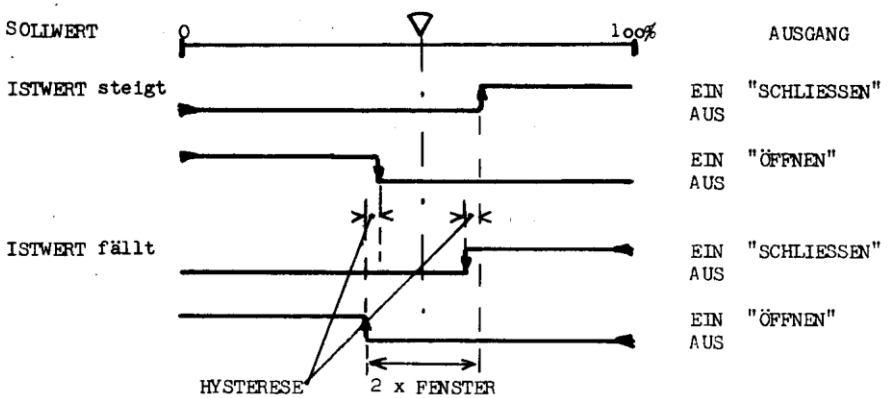
Prozeßtechnik mit Mikrocomputern (4)

Der Dreipunktregler (DPR)

Konnte der Zweipunktregler nur ein Stellglied in einer Wirkrichtung EIN-/AUS-schalten, so kann der Dreipunktregler dies in 2 Wirkrichtungen. Wie sein Name sagt, hat er drei Betriebspunkte:

Stellbefehl	*WENIGER*	*oder*	*SCHLIESSEN*	*oder*	*KÜHLEN* *usw.*
Nichts tun					
Stellbefehl	*MEHR*	*oder*	*ÖFFNEN*	*oder*	*HEIZEN*

Er benötigt somit zwei Ausgänge eines Ports. Hier seine Arbeitskennlinien:



Der Dreipunktregler besteht aus einem Analogteil und den Ausgangs-Schaltstufen. Der Analogteil führt zwei Berechnungen aus:

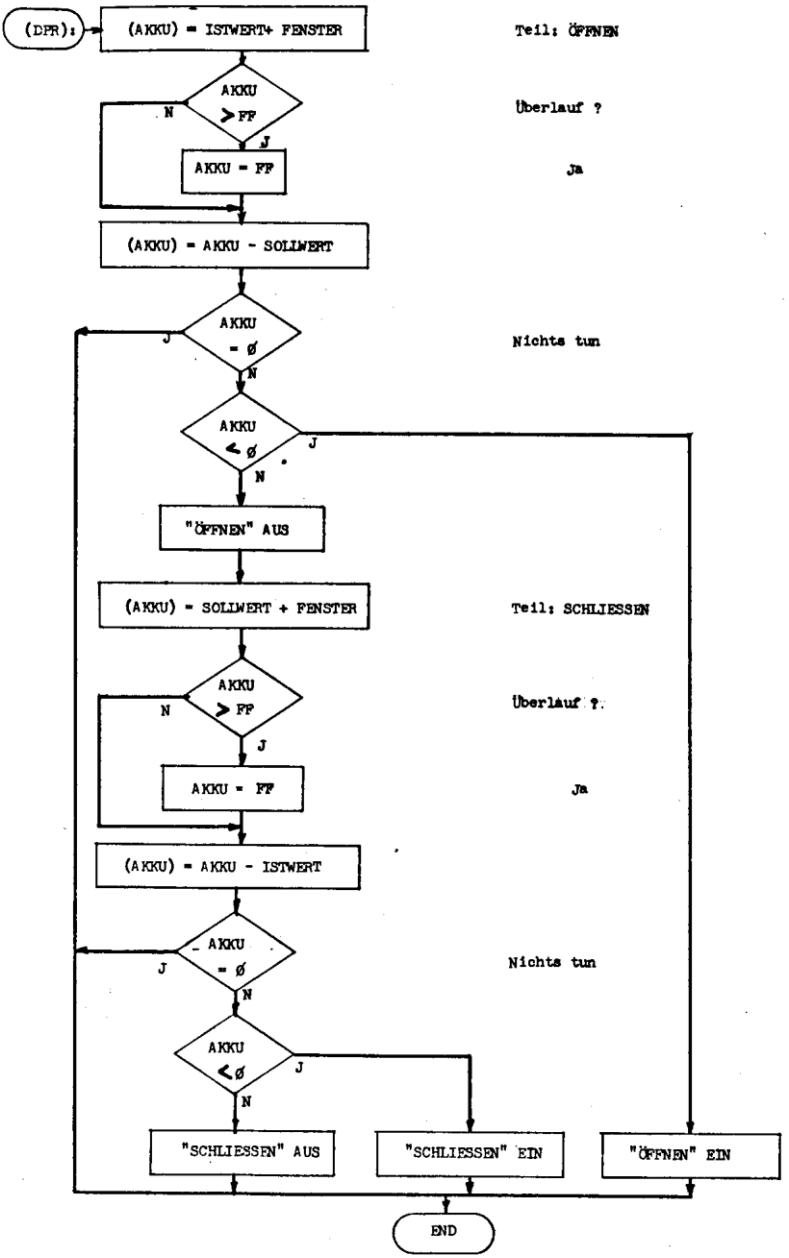
Istwert < Sollwert und
Istwert > Sollwert.

Die möglichen Ergebnisse sind:
"ÖFFNEN" EIN
"ÖFFNEN" AUS
Alles so lassen, wie es ist
"SCHLIESSEN" AUS
"SCHLIESSEN" EIN.

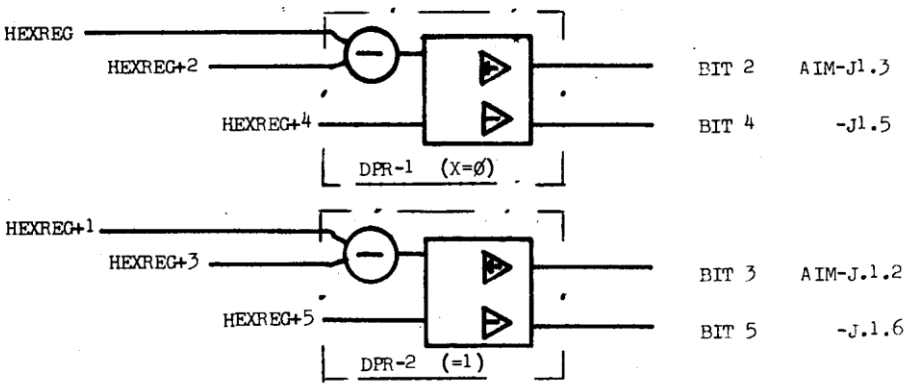
Diese Schalthandlungen werden von den Ausgangs-Schaltstufen ausgeführt. Das Flußdiagramm zeigt den Funktionsablauf.

Der Dreipunktregler hat noch kein Zeitverhalten, um begrenzte Stellimpulse abzugeben, z.B. auf Stellmotore von Mischventilen. Dazu brauchen wir einen Propotionalregler mit Pulsausgang, der in 3. Teil der Regler vorgestellt wird. - Man kann sich den Dreipunktregler vorstellen als zwei einzelne Zweipunktregler, deren Sollwert um jeweils das 'Fenster' nach oben und unten verschoben ist. Außer den Eingangsgrößen Istwert, Sollwert und Fenster benötigen wir eine Hyterese, damit der Regler gegen Schwankungen der Eingangsgrößen stabil ist. Für das Programm-Modul DREIPUNKTREGLER gilt das auf der übernächsten Seite abgedruckte Blockschaltbild.

FLUSSDIAGRAMM (DPR) 3-Punkt-Regler



Blockschaltbild für das Programm-Modul DREIPUNKTREGLER



Programmerklärung zum Modul DREIPUNKTREGLER (DPR)

Damit die einzelnen Module leicht zusammenzufügen sind, enthält jedes Modul alle erforderlichen Variablen, Konstanten und die nötigen EQUATES. Für die Trennung der Variablen vom Programm sorgen die 'LINK'. Sie weisen den Assembler an, die richtigen Adressen zuzuordnen. Ein 'LINK' besteht jeweils aus einem Paar gleichen Namens. Der erste Teil des Namens ist dabei 'PRO' für 'Programm' bzw. 'VAR' für 'Variable'. Der zweite Teil hat den Namen des Moduls. Beispiel:

```
VARADW = *          ;Variablen-Link von ADW
* = VARADW          ;Beginn der 'DPR'-Variablen.
```

Beim Zusammenstellen eines Programmes aus verschiedenen Modulen sind jeweils die Eingangs-Links auf den Namen des vorstehenden Modules auszurichten. Die Ausgangs-Links sind immer richtig, denn sie tragen ja den Namen des Moduls.

Die Initialisierungen der benötigten Ports und Timer sind so geschrieben, daß sie sich im Hauptprogramm 'in Serie schalten' lassen. Jeder bedient nur seine Bit ohne die anderen dabei zu verändern. Zusammengefaßt wird der Code etwas kürzer, aber so 'RAM-arm' ist ja man wohl meistens nicht mehr.

Das Modul 'DPR' kann bis zu 4 unabhängige Regler bedienen, vier deshalb, weil je Regler zwei Ausgangsbit benötigt werden und weil diese im selben Port liegen müssen. Jeder Regler hat aber seine eigenen Eingangsgrößen Istwert, Sollwert und Fenster. Sie müssen allerdings in der richtigen Reihenfolge vorliegen:

```
Zuerst alle   Istwert
dann die      Sollwert
zuletzt       Fenster       Man kann es aber auch anders ordnen.
```

Zum Aufruf eines Regler ist seine Adresse (0 bis 3) im 'XPOINT' zu hinterlegen. Das Testprogramm zeigt die serielle Abarbeitung von 2 Reglern. In früheren Modulen wurde die Adresse im X-Register übergeben, damit fiel dieses Register für Operationen aus. So ist es sicherer.

Die EIN/AUS-Schaltbefehle werden über X-indizierte Masken ausgeführt. Dies erlaubt die freie Bitbelegung. Durch Invertierung mit EOR #$FF können die EIN- zu AUS-Masken umgewandelt werden. Da andere Module Label mit gleichen Bedeutungen haben und weil dies zu Protesten des Assemblers führt, tragen die meisten Label die ersten beiden Buchstaben des Moduls.

Im nächsten Heft wird eine verbesserte Version des in Heft 20 veröffentlichten 'ADW', Analog/Digital-Wandlers abgedruckt.

```
MODUL: <DPR> 3-PUNKT-REGLER

025D          0107 ;ZUM REGELN VON 1 BIS 4 PROZESSGROESSEN
025D          0108 ;MIT DAUERBEFEHLS-AUSGABE 'SCHLIESSEN-STOP-OEFFNEN'
025D          0109 ;JEDER REGLER WIRD MIT <XPOINT> ANGEWAEHLT
025D          0110 ;ER HAT EIGENE IST- UND SOLLWERTE, SOWIE FENSTERBREITE
025D          0111 ;DIE AUSGAENGE FUER 'SCHLIESSEN U. OEFFNEN' MUESSEN IM
025D          0112 ;SELBEN PORT LIEGEN, DARUM MAX.4 REGLER.
025D          0113 ;JEDER AUSGANG HAT SEINE 'EIN'-MASKE
025D          0114 ;HIER SIND DIE 'EIN'-MASKEN FUER 2 REGLER VORGESEHEN.
025D          0115 ;'AUS'-MASKEN ENTSTEHEN DURCH INVERTIEREN DER 'EIN'-MASKE
025D          0116
025D          0117
025D          0118 ;+++++ VARIABLEN & KONSTANTEN <DPR>
025D          0119
025D          0120 DPZAHL =2              ;1-4 DREIPUNKT REGLER, HIER =2
025D          0121 DPPORT =UDRA           ;USER-6522-PORT A
025D          0122
025D          0123        *=VARADW        ;VARIABLEN-LINK VON <ADW>
00B7          0124 ISTREG =HEXREG         ;ZUORDNUNG DER ARBEITS-REGISTER
00B7          0125 SOLREG =ISTREG+DPZAHL
00B7          0126 DPWIND =SOLREG+DPZAHL
00B7          0127 VARDPR =*              ;VARIABLEN-LINK ZU <ERR>
00B7          0128
00B7          0129        *=PRGADW        ;PROGRAMM-LINK VON <ADW>
025D          0130 ;+++++ INITIALISIERE PORT FUER <DPR>
025D AD03A0   0131 INIDPR LDA DPPORT+2    ;PORT-DIR.REG.
0260 093C     0132        ORA #%00111100  ;BIT 2-5= OUT
0262 8D03A0   0133        STA DPPORT+2
0265 60       0134        RTS
0266          0135
0266          0136
0266          0137 ;      ++++++++++ PROGRAMM <3-PUNKT-REGLER>
0266          0138
0266 A6B6     0139 DPR    LDX XPOINT      ;REGLER-INDEX
0268          0140
0268          0141 ;----- IST ISTWERT + FENSTER < SOLLWERT ?
0268 18       0142        CLC
0269 B5B0     0143        LDA ISTREG,X    ;ISTWERT
026B 75B4     0144        ADC DPWIND,X    ; PLUS FENSTER
026D 9002     0145        BCC DP1         ;UEBERLAUF?
026F A9FF     0146        LDA #$FF        ; -JA, BEGRENZE
0271 D5B2     0147 DP1    CMP SOLREG,X    ; MINUS SOLLWERT
0273 F022     0148        BEQ LETSO       ;NICHTS TUN
0275 9021     0149        BCC MORE        ;OEFFNEN EIN
0277 BDAE02   0150        LDA DPMORE,X    ;SCHALT-MASKE
027A 20BF02   0151        JSR DP3         ;SCHALTSTUFE 'AUS'
027D          0152
027D          0153 ;----- IST SOLLWERT + FENSTER < ISTWERT ?
027D 18       0154        CLC
027E B5B2     0155        LDA SOLREG,X    ;SOLLWERT
0280 75B4     0156        ADC DPWIND,X    ; PLUS FENSTER
0282 9002     0157        BCC DP2         ;UEBERLAUF?
0284 A9FF     0158        LDA #$FF        ; -JA, BREGRENZE
0286 D5B0     0159 DP2    CMP ISTREG,X    ; MINUS ISTWERT
0288 F00D     0160        BEQ LETSO       ;NICHTS TUN
028A 9017     0161        BCC LESS        ;SCHLIESSEN EIN
```

```
028C BDB002  0162        LDA DPLESS,X    ;MASKE 'SCHLIESSEN'
028F 49FF    0163 DP3    EOR #$FF        ; INVERTIEREN: AUS
0291 2D01A0  0164        AND DPPORT      ; UEBERTRAGEN
0294 8D01A0  0165 STORE  STA DPPORT      ; AUSGEBEN
0297 60      0166 LETSO  RTS
0298         0167
0298         0168 ;STEUERBEFEHLE AUSFUEHREN
0298         0169 ;----- 'OEFFNEN EIN' DA ISTWERT < SOLLWERT
0298 BDB202  0170 MORE   LDA DPOFF,X     ;ERST BEIDE AUSSCHALTEN
029B 2D01A0  0171        AND DPPORT
029E 1DAE02  0172        ORA DPMORE,X    ; DANN BIT SETZEN
02A1 D0F1    0173        BNE STORE       ;IMMER
02A3         0174
02A3         0175 ;----- 'SCHLIESSEN EIN' DA ISTWERT > SOLLWERT
02A3 BDB202  0176 LESS   LDA DPOFF,X
02A6 2D01A0  0177        AND DPPORT
02A9 1DB002  0178        ORA DPLESS,X
02AC D0E6    0179        BNE STORE       ;IMMER
02AE         0180
02AE         0181 ;----- MASKEN ZUM OEFFNEN- UND SCHLIESSEN-'EIN'
02AE 04      0182 DPMORE .BYT %00000100  ;BIT 2, 'OEFFNEN EIN'-REGLER 1
02AF 08      0183        .BYT %00001000  ;BIT 3, REGLER 2
02B0 10      0184 DPLESS .BYT %00010000  ;BIT 4, 'SCHLIESSEN EIN'-REGLER 1
02B1 20      0185        .BYT %00100000  ;BIT 5, REGLER 2
02B2 EB      0186 DPOFF  .BYT %11101011  ;'AUS'-REGLER 1
02B3 D7      0187        .BYT %11010111  ;'AUS'-REGLER 2
02B4         0188 PRODPR =*              ;PROGRAMM-LINK ZU <ERR>
02B4         0189
```

#

Horst Brettin, 1000 Berlin 44

SWAP für BASIC 3

Dieses Programm wurde ursprünglich als SWAP$ (nur für String-Variable) entwickelt, um die Garbage-Collection-Routine des CBM zu überlisten. Wegen seiner Nützlichkeit wurde es dann aber zu SWAP erweitert. Es erlaubt den Austausch von Variablen gleichen Typs untereinander. Es hat die Aufruf-Syntax

SYS Adr,Var1,Var2

Beispiele: SYS 826,A%(I),A%(J) oder
SWAP=826
SYS SWAP,A(I),A(I+1)

Var1 und Var2 müssen also vom selben Typ sein. SYS 826,X,X% hat z.B. eine Fehlermeldung zur Folge! Die Anwendung von SWAP dürfte aus dem Demo-Programm SHELL-SORT gut ersichtlich sein.

SWAP läßt sich aber auch beim QUICK-SORT sowie bei Variablentauschaktionen jeder Art anwenden. Die hier vorliegende Version wurde für den CBM 3032 geschrieben. Es müßte aber möglich sein, sie auch auf anderen BASIC 3-Rechnern zu implementieren, z.B. auch beim AIM mit BASIC-ROMs, ebenfalls wohl beim PET 2001. Die Implementierung in BASIC 4 stößt auf zusätzliche Schwierigkeiten, und zwar wegen der String-Trailer (Vektoren am String, die auf den Descriptor zurückweisen), die für die schnelle Garbage Collection gebraucht werden.

SHELL-SORT ist übrigens ein guter Kompromiß, wenn einfache Sortierroutinen zu langsam sind und der Aufwand für QUICK-SORT sich nicht lohnt. Die eigentliche SHELL-SORT-Routine ist den in Zeilen 140 bis 200. Zeile 120 füllt das Feld mit Zufallszahlen, Zeile 220 prüft auf Sortierfehler. Um die mittlere Sortierzeit zu ermitteln, wird zehnmal sortiert (Zeilen 110 und 250). Na-

türlich wird jedesmal neu mit Zufallszahlen gefüllt. Zum Vergleich kann man SYS SWAP,A$(I), A$(I+B) durch folgenden Ausdruck in Zeile 170 ersetzen

```
T$ = A$(I):A$(I)=A$(I+B);A$(I+B)=T$
```

SWAP

```
                STRFLG   EQU $7              (C) 1982 BY
                INTFLG   EQU $8             HORST BRETTIN
                INDX     EQU $1F
                VARADR   EQU $44

                MISMATCH EQU $CC9A          TYPE MISMATCH ERROR
                CHKKOM   EQU $CDF8          PRUEFT OB ',' FOLGT
                GETVAR   EQU $CF6D          VARIABLEN-NAMEN LESEN U.
                                                        SUCHEN
033A  206403 SWAP       JSR VARGET
033D  48                PHA                 ;VEKTOR RETTEN
033E  A545              LDA VARADR+1
0340  48                PHA
0341  98                TYA                 ;TYPEINDEX RETTEN
0342  48                PHA
0343  206403            JSR VARGET          ;2.VARIABLE
0346  68                PLA                 ;1.TYPEINDEX
0347  8507              STA STRFLG
0349  68                PLA                 ;1.VEKTOR
034A  8520              STA INDX+1
034C  68                PLA
034D  851F              STA INDX
034F  C407              CPY STRFLG          ;TYPEN GLEICH?
0351  D00E              BNE ERR
0353  B144    CHANGE    LDA(VARADR),Y
0355  AA                TAX
0356  B11F              LDA(INDX),Y
0358  9144              STA(VARADR),Y
035A  8A                TXA
035B  911F              STA(INDX),Y
035D  88                DEY
035E  10F3              BPL CHANGE
0360  60                RTS

0361  4C9ACC ERR        JMP MISMATCH

0364  20F8CD VARGET     JSR CHKKOM
0367  206DCF            JSR GETVAR
036A  A002              LDY#2               ;3 BYTE DISCRIPTOR
036C  2407              BIT STRFLG
036E  3007              BMI IF
0370  88                DEY                 ;2 BYTE INTEGER
0371  2408              BIT INTFLG
0373  3002              BMI IF
0375  A004              LDY#4               ;5 BYTE REAL
0377  60      IF        RTS
```

SHELL-SORT

```
 100 INPUT"MENGE";M:DIM A$(M):SWAP=826
 110 PRINT"□":T0=0:FOR N=1 TO 10
 120 FOR I=0 TO M:A$(I)=STR$(RND(1)):NEXT
 130 T0=TI-T0
 140 B=M
 150 B=INT(B/2+.5):T=M-B
 160 R=T:T=0:FOR I=0 TO R
 170 IF A$(I)>A$(I+B) THEN SYS SWAP,A$(I),A$(I+B):T=I
 180 NEXT
 190 IF T GOTO 160
 200 IF B>1 GOTO 150
 210 T0=TI-T0
 220 FOR I=0 TO M-1:IF A$(I)<A$(I+1) THEN NEXT:GOTO 250
 230 PRINT"SORTIERFEHLER IN"I
 240 END
 250 NEXT N:PRINTM"ELEMENTE  "INT(T0/6+.5)/100"SEC.
 260 RUN
READY.                                                    #
```

Johann Leitner, 6000 Frankfurt 90

SHAKER (6502)

SHAKER erzeugt alle möglichen Kombinationen einer vorgegebenen Zahl von Bytes. Das Programm entstand in Anlehnung an ein kurzes BASIC-Programm (1) und verdankt seinen Namen einem Vorgänger in MICRO MAG (2). Es ist auf jedem 6502-System einsetzbar, wobei lediglich die Adresse von BSOUT geändert werden muß. BSOUT (3) ist eine ROM-Routine im CBM/PET, die das Zeichen im Akkumulator auf dem aktiven Peripheriegerät ausgibt und die Register A, X, Y unverändert läßt.

Die Größe von STR und ZWI muß jeweils LEN+1 sein, das Beispielprogramm kann also maximal 9 Bytes permutieren. LEN ist aber vom Prinzip her nur durch die Zahl der möglichen JSR-Aufrufe des Systems (und die verfügbare Rechenzeit) begrenzt. Bei Bedarf können die erzeugten Permutationen auch für eine eventuelle Weiterverarbeitung zwischengespeichert werden. Dies kann innerhalb von AUSGAB erfolgen.

Vor dem Aufruf müssen die Bytes ab STR+1 und deren Anzahl in LEN übergeben werden. Beim CBM/PET kann man durch vorheriges OPEN und CMD die Ausgabe auch auf Drucker oder Floppy leiten. Nach dem Programmablauf stehen die Bytes wieder in der ursprünglichen Reihenfolge, auch LEN ist unverändert.

Eine mögliche Anwendung ist die Erzeugung von (Farb-) Mustern aus vorgegebenen Einzelteilen. Auch bei manchen Optimierungsaufgaben benötigt man alle Permutationen einer Grundmenge. Oder man sieht das Programm einfach als Beispiel für eine Problemlösung mittels Rekursion.

Literatur: (1) Stryker, T., The Great Scramble Debate, Microcomputing 3 1982, S. 169. Peterborough, NH, USA, (2) Müller, W. E., SHAKE, 65xx MICRO MAG Nr. 23, S. 30, (3) Commodore, cbM 3016 und cbm 3032 Assembler Listing, Commodore Neu Isenburg, Juni 1979.

```
line# loc   code        line

00001  0000             ;--- shaker 290382 jl --- 6502 code ---

00003  0000             ;alle permutationen des strings ab str+1
00004  0000             ;mit der länge len werden ausgegeben.
```

```
00006  0000                 space=$20
00007  0000                 bsout  = $ffd2          ;(cbm/pet) gibt zeichen in a aus
00008  0000                        *=650
00009  028a                 len                     ;stringlänge
00010  028a                 str    *=*+10           ;string steht ab str+1!
00011  0294                 zwi    *=*+10           ;zwischenspeicher

00013  029e  ad 8a 02       start  lda len          ;sys 670 (cbm/pet)
00014  02a1  aa                    tax
00015  02a2  9d 94 02              sta zwi,x
00016  02a5  8a             permut txa
00017  02a6  d0 03                 bne per2
00018  02a8  20 d8 02              jsr ausgab
00019  02ab  de 94 02       per2   dec zwi,x
00020  02ae  10 02                 bpl per3
00021  02b0  e8                    inx
00022  02b1  60                    rts
00023  02b2  ca             per3   dex
00024  02b3  8a                    txa
00025  02b4  9d 94 02              sta zwi,x
00026  02b7  20 a5 02              jsr permut      ;rekursiver aufruf
00027  02ba  8a             neustr txa             ;x-tes zeichen an stringanfang
00028  02bb  48                    pha
00029  02bc  f0 15                 beq neux
00030  02be  a8                    tay
00031  02bf  88             neu2   dey
00032  02c0  f0 11                 beq neux
00033  02c2  bd 8a 02              lda str,x
00034  02c5  48                    pha
00035  02c6  b9 8a 02              lda str,y
00036  02c9  9d 8a 02              sta str,x
00037  02cc  68                    pla
00038  02cd  99 8a 02              sta str,y
00039  02d0  ca                    dex
00040  02d1  d0 ec                 bne neu2        ;entspricht jmp
00041  02d3  68             neux   pla
00042  02d4  aa                    tax
00043  02d5  4c a5 02              jmp permut

00045  02d8  ae 8a 02       ausgab ldx len
00046  02db  f0 13                 beq ausx        ;string ist leer
00047  02dd  a2 00                 ldx #0
00048  02df  bd 8b 02       aus2   lda str+1,x
00049  02e2  20 d2 ff              jsr bsout
00050  02e5  e8                    inx
00051  02e6  ec 8a 02              cpx len
00052  02e9  d0 f4                 bne aus2
00053  02eb  a9 20                 lda #space
00054  02ed  20 d2 ff              jsr bsout
00055  02f0  a2 00          ausx   ldx #0
00056  02f2  60                    rts

00058  02f3                       .end
```

AIM Spezial (12)

Die DAIM-Floppy wird gelegentlich am AIM 65 eingesetzt und hat dann Schwierigkeiten. Herr B. Kokula teilt mit, daß hier, wie auch an der Floppy des Systems 65 ein Beschaltungsfehler vorliegt: Die Pins 8 und 14 am Controller NEC 372 wurden einfach offen gelassen. Sie müssen auf 0 Volt (GND) gezogen werden, wenn man vorsichtig sein will, auch über 10 kOhm.

Thermopapier haftet oft im Printer des AIM 65, weil die Walze zu trocken geworden ist. Bekanntlich fehlt es an einer Gegenwalze für die exakte Papierführung. Abhilfe: Gummiwalze bei Bedarf mit Spiritus reinigen, dann eine Fingerspitze voll Glycerin auftragen, Überschuß entfernen. Das Gummi wird wieder elastisch und feuchtigkeitsanziehend und befördert das Papier durch bessere Reibung.

Bücher

Leventhal, Lance A.: Z80 Assembly Language Programming. Verlag Osborne/McGraw-Hill, Berkely, California 1979, ISBN 0-931988-21-7, ca. 500 Seiten.

Der Autor dieses Buches hat verschiedene einführende Bücher für die Programmierung von Mikroprozessoren geschrieben, so für den 6502 und den 6809. Sie haben jeweils etwa den gleichen didaktischen und schematischen Aufbau, so daß man bei der Durcharbeitung des dritten Buches schon etwa weiß, was zunächst kommen wird, nämlich Hausmannskost mit sehr einfachen Programmbeispielen, die besonders für den 6809 und für den Z80 nicht ausreichend auf die besonderen Techniken eingeht, die die Befehlsstruktur dieser Prozessoren zuläßt.

Die CPU Z80 hat einen umfangreichen Befehlssatz, der auf den ersten Blick nicht so regelmäßig aussieht. Nach Ansicht des Rezensenten hätte der Autor gerade deswegen seine Seiten nicht mit Hexcode der Befehle füllen sollen, sondern mit zusammenfassenden Übersichten, z.B. mit Gruppierungen wie: welche Befehle und Adressierungsarten gibt es für Load Register from Memory, Store Register to Memory, Transfer Register to Register, Store Immediate to Memory usw.. Gruppierung dient der Übersicht des Lesers. Wenn man sich die trotz dieses Buches notwendige Mühe macht, nur mit dem 'Z80 CPU Technical Manual' eine eigene Ordnung zu schaffen, dann hat man auf 2 Seiten ein Arbeitspapier auf dem Tisch, das die Regelmäßigkeiten des Befehlssatzes und der Adressierungsmöglichkeiten für den täglichen Gebrauch aufweist.

Man kann auch keine Bücher schreiben, die in Übersichten nicht auf die Veränderung der Status-Flags hinweisen. Wo findet man bei Leventahl die zusammenfassende Aussage in einem umrandeten Kästchen, daß beim 6502 fast alle ladenden Befehle Flags setzen, daß beim 6809 auch die abspeichernden Befehle Flags setzen oder daß fast alle den Z80 ladenden Befehle keine Flags setzen? Wie kann der Programmierer bei diesen Eigenheiten einem Verlust der Status-Information vorbeugen? Programmierung heißt Status-Information für Entscheidungen zu erzeugen und zu verwerten. Man kann auf über 500 vom Verleger zur Verfügung gestellten Seiten mehr Aussagen machen.

Aus der Branche/Produkte

System 7000/7100 nennt sich das neue Komplettsystem von ELTEC, Mainz, basierend auf dem neuen EUROCOM EII/V7 mit CPU 6809. Der Lieferumfang umfaßt Voll-Graphik-Video, 64 KB RAM, zwei Floppy Disk-Laufwerke, Gehäuse, frei programmierbare ASCII-Tastatur, FLEX-Betriebssystem, bildschirm-orientierter Editor, Textprozessor. Die beiden Systeme unterscheiden sich durch die Speichergröße der Floppies von 2x200 KB bis 2x640 KB. Die Software-Optionen umfassen ein Datenbanksystem, BASIC und einen PASCAL-Compiler. - Die enthaltene Tastatur ist auch unabhängig lieferbar (Bezeichnung PAT09), mit Tastenblöcken wie folgt: 19 Funktionstasten (z.B. für den Abruf von Textketten), 11 Cursor-Tasten, 11 numerische Tasten und 65 ASCII-Tasten. - Diese Tastatur enthält offensichtlich den Mini-Singleboard-Computer PATTY von ELTEC mit CPU 6809, 2 VIAs 6522 und 1 ACIA 6850 für das Interfacing, letzterer für serielle Datenübertragung. Info: ELTEC Electronic GmbH, Postfach 1847, 6500 Mainz 1

PETAL-Procompiler für Commodore-Computer. Dieser Precompiler arbeitet ohne irgendwelche Hardwarezusätze und stellt ein erweitertes BASIC mit hervorragenden Strukturierungsmöglichkeiten zur Verfügung, nämlich mit IF/THEN/ELSE/ENDIF, CASE/(ELSE)/ENDCASE, DO/WHILE/ENDWHILE, DO/UNTIL/ENDUNTIL, DO/EXIT IF//THEN/EXIT/ENDEXIT. Das letztgenannte Konstrukt erlaubt die Bedingungsabfrage und das Verlassen an beliebiger Stelle. Unterprogramme können entweder explizit mitgeteilt werden oder der Precomiler sucht sie sich auto-

matisch auf den Disketten, so daß sich umfangreiche Programmbibliotheken flexibel aufbauen lassen. PETAL ist für alle BASIC-Versionen erhältlich unf für alle CBM-Floppies und enthält eine ausführliche Dokumentation. Preis DM 395, inkl. MWSt. Info/Bezug: Laurin Digital-Systeme, Gerhard von Blücher, Postfach 50 02 07, 2000 Hamburg 50.

ROCKWELL International: Zu den RM65-Modulen gesellt sich jetzt das intelligente CRT-Controller-Modul RM65-5102 mit Betriebsprogramm. Die Karte im Europaformat erzeugt Groß und Kleinschrift in 6 verschiedenen Formaten von 40 Zeichen x 16 Zeilen bis 80 x 25. Es kann nicht nur an Board-Computersystemen, sondern auch am AIM 65 und am AIM 65/40 betrieben werden. - Das MPI RM65-5252 ist eine Interfacekarte mit 2 VIAS 6522 und für die Ausgabe mit TTL gepufferten Ports. Die Puffer können für die Eingabe überbrückt werden.

Neue Interfacebausteine 10937/10938/10939 für Vakuum-Fluoreszenz-Displays von Rockwell. Sie werden über einen seriellen Port angesprochen und können damit an jedem Computer betrieben werden. Der erstgenannte hat den vollen ASCII-Satz für Anzeigen mit 14-18 Segmenten und ist bereits lieferbar. Die beiden letzeren arbeiten zusammen, kommen gegen Jahresende, und steuern Punkt-Matrixanzeigen an.

Für den AIM 65 wird ein komfortables RENUMBER angeboten, als Maschinenprogramm in mehr als 1 KB, das auch die Möglichkeiten des 'von ...bis' und er Umordnung von Blöcken enthält. Info: Rainer Wittulski, Fritz-Pütter-Str. 14, 44 Münster.

CRT-Controller R6545 von Rockwell: Herrn H.-J. Regge in Bremen fiel ein Spezifikationsunterschied auf, der zu Schwierigkeiten bei der Programmierung führt, wenn man ihn nicht kennt: Nach dem Datenblatt Doc. No. 29000 D 55, Rev. 1, Jun. 79 (das kein vorläufiges war!) enthält der Baustein neben den Registern R0 bis R17 noch zwei weitere (R18 und R19 sowie die 'dummy location' R31), die zum Betrieb ohne die beim 6845 von Motorola erforderlichen Multiplexer gedacht sind. Bei den bis zum heutigen Tage ausgelieferten und in absehbarer Zeit verfügbaren Chips fehlen diese Register, auch das Statusregister ist gegenüber dem Datenblatt abgemagert, wie dem Datenblatt Doc. 29 000 D 67 von Dec. 1980 zu entnehmen ist. Man beachte also die Hinweise 'specifications subject to change without notice'! - Die 2 MHz-Version des Chips ist jetzt ab München verfügbar, auch den 6845 gibt es nun mit 2 MHz.

FIG-FORTH für CBM 3001/4001/8001. Der Computer-Service Lowinski bietet jetzt FORTH mit einem zeilenorientierten Editor, mit Macro-Assembler und Diskettenhandler für die CBM-Floppies an. Damit ist auch die virtuelle Speicherung in 'Screens' möglich. Als Erweiterung zum Grundpaket (DM 250) gibt es ein Floating Point Package (DM 150) und ein Stringpaket (DM 150). Das Floating Point Package erlaubt die Verwendung aller vom CBM her gewohnten Real-Arithmetik inklusiver der Funktionen, indem es ROM-Routinen mitbenutzt. Es erweitert damit die normal nur implementierte Integer-Rechenfähigkeit. Im Stringpaket stehen die auch vom BASIC her bekannten Stringfunktionen einschl. INSTRING zur Verfügung. Info/Bezug: Wolfgang Lowinski Computerservice, Gallwitzstr. 10, 7800 Freiburg.

Die leistungsfähigen Systeme Corvus Concept mit CPU 68000 werden seit kurzem von der GWK, Gesellschaft für technische Elektronik mbH in Herzogenrath vertrieben.

Editorial

Dieses Heft hat das Schwerpunktthema FORTH. Es soll damit die noch immer bestehende Informationslücke hinsichtlich des Gebrauches dieser Sprache möglichst schließen und zu eigenen Arbeiten ermuntern. FORTH wird in etwa drei Nachmittags- und Abendsitzungen geübt und erlernt, wenn man die Fülle seiner Befehlsworte zur eigenen Übersicht schon etwas vorsortiert hat. Bei einer solchen intensiven Einarbeitung erkennt man bald die Vorzüge dieser Sprache: Sie generiert einen knappen und schnell ausführenden Code, ist beliebig erweiterungsfähig und erlaubt den Einschluß von Assemblerprogrammen in die Liste der ausführbaren Befehlsworte.

Bei den Übungen an der Tastatur hat man ohne umständliche Hantierungen eine unmittelbare Erfolgskontrolle für die einzelnen Module. Undeutlich formulierte Kontrollstrukturen werden zurückgewiesen. Das Hauptaugenmerk ist auf die eigene richtige Verwaltung des Datenstacks zu legen, damit das zuoberst liegt, was als nächstes bearbeitet werden soll.

FORTH übt im Laufe der Zeit sicher einen erzieherischen Einfluß auf den Programmierer aus, er wird veranlaßt, in mit Namen versehenen Tätigkeitsmodulen zu denken und zu formulieren. Auch die Kontrollstrukturen, wie IF ... THEN ... ELSE, BEGIN ... WHILE ... REPEAT usw., sollte man mit ihren Regeln nicht als einen Zwang ansehen, sondern als Aufforderung und Hilfe, die für eine Programmierung immer notwendigen Entscheidungen im Vorwege zu überdenken, um dann um so effektiver programmieren zu können.

Mit der zunehmenden Information wird FORTH bei den Lesern dieser Zeitschrift mit der besonderen Lage ihrer Interessen immer mehr Zuspruch finden. Das Prinzip dieser Sprache wird man nach der Serie 'FORTH im Eigenbau' von Michael Zimmermann gewinnbringend auch für Anwendungscomputer verwenden können, ohne daß man sie mit FORTH-ROMs bestücken muß.

Aus der Sicht des Herausgebers hat FORTH gegenüber dem in diesem Heft auch dargestellten PL/65 einige Vorzüge: Man gelangt ohne umständliche Hantierungen (PL/65-Quelltext, Assemblerlauf, Objectcode) zum sofort ausführbaren Modul und man hat tiefere Verschachtelungsmöglichkeiten in den Kontrollstrukturen, wie z.B. ein Blick auf die Seite 12 mit der Routine $COMP zeigt.

Programmteile, die man mit dem Assembler des FORTH formuliert und deren Maschinencode man wegen der Kontrollstrukturen disassembliert neben den Quelltext stellt, können für alle Leser, die vielleicht nicht mit FORTH programmieren, von Nutzen sein, wie vielleicht einige der veröffentlichten Stringroutinen oder künftige dezimale Gleitkomma-Pakete oder Module zur Prozeßsteuerung, unter FORTH oder ähnlich laufende komplette Betriebssysteme, Monitore oder leistungsfähige Text-Editore für Anwendungscomputer. Wer als Assembler-Programmierer solche dann ja auch strukturiert-übersichtlichen Module für seine Zwecke benutzen will, braucht im wesentlichen nur das abschließende 'JMP NEXT' durch ein 'RTS' zu ersetzen.

Der Herausgeber ist oft gebeten worden, eine vergleichbar nützliche und sehr professionelle Informationsquelle wie für den 6502 auch für den Z80 und auch für sein vorwiegendes Betriebssystem CP/M zu schaffen. Bemerkenswert ist ja, daß auch immer mehr 6502-Computer die Option erhalten, mit einer Z80 betrieben werden zu können, um speziell die unter CP/M verfügbaren Sprachen und auch Dienstprogramme aufzuschließen. Es wird also ab etwa Mitte September 1982 eine Schwesterzeitschrift **8xxx MICRO MAG** *geben, die die Erwartungen einer engagierten Leserschaft in gleicher Weise erfüllen soll.*

#

Programmverbesserung zu LMR – LOAD, MOVE, RELOCATE in Heft 17, Seite41: Herr Klaus-Gerd Meyer aus Oldenburg schlägt vor, die Ziele dreier relativer Sprünge zu optimieren, es ergibt sich eine wesentlich schnellere Programmausführung bei MOVE. Auch die MOVE-Routine in Heft 7 sollte man sinngemäß abändern.

Adresse	alt	neu
0372	BNE NEIN	BNE COUNT
0386	BNE ONE	BNE MOV
038A	BNE LOOP	BNE MOV

Kleinanzeigen

AIM 65: Verkaufe Matrixdrucker (Dohmann) s. MICRO MAG 11, kompl. anschlußfertig 750,--; Epromprogrammer 2716 (Dohmann) 80,-; 32 kB RAM-Karte (Neudecker) 520,-. P. Bieberich, Nürnberg, Tel. 0911 - 22 13 83 ab 18 Uhr.

Quellentext, assemblierfähig zu AIM 65-FORTH V1.3 erhältlich bei Bernd Hilbert, DB5JN, Egellstr. 34, 6700 Ludwigshafen, Tel. 0621-68 24 76. Lieferformen: Paperlist (78 S. A4) DM 50, als 5''-Diskette Rockwell oder MCS DM 100, AIM-Tape DM 100.

AIM 65-Erweiterung, anschlußfertig, m. RAM, Eprom, Eprom-burner, zwei 5 1/4'' Laufwerken, Mutterplatine, Netzteil, Disk-orientierter Assembler, kompatibel zu System 65, BASIC, FORTH, PL/65, PASCAL und div. Software z.verk.: H. J. Regge, Fesenfeld 57, 2800 Bremen 1, T. 0421-71 114, abends

Für AIM 65: Textverarbeitung mit Druckerausgabe (Groß- und Kleinbuchstaben, Blocksatz). BASIC-Renumber (First, Incr., Start, End). Info gegen Freiumschlag bei R. Witulski, Fritz-Pütter-Str. 14, 4400 Münster, Tel.: 0251-77 85 67

Programmieren Sie noch immer in Nostalgie ?

lernen Sie fliegen mit :

neu !

"STARTING FORTH" von Leo Brodie
dem humorvoll geschriebenen und illustrierten FORTH-Lehrbuch und Benutzer-Handbuch **48,70 DM**

6809-FORTH-System mit Assembler und Editor; FLEX (5.25/8") von Talbot **450,00 DM**

6809-TARGET-Compiler (5.25/8") erzeugt direkt lauffähigen Maschinencode **1350,00 DM**

DR.J.SCHRENK
Postfach 904
7500 KARLSRUHE-41

Kleinanzeigen

AIM 65 - Instant PASCAL Handbuch in Deutsch, wortgetreue, genehmigte und empfohlene Übersetzung des engl. Originals, 26,- DM 6502-Universalplatine lt. Titelbild MC 2/82 unbest. 73,- DM, Uhrenprogramm f. obige Platine in 2716 inkl. Source 45,- DM. Preise sind Endpreise zzgl. Porto/NN-Gebühren. H.J. Regge, Fesenfeld 57, 2800 Bremen 1, Telefon 042171 114, abends.

Verkaufe Color-Daten-Monitor mit Grünschalter u. 8-Kanal TV (CTP3245-c). von Sanyo, neu, kaum gebraucht: DM 1200 VHB. R. Hammerschmidt, Heinr.-Heine-Str. 4, 4006 Erkrath.

AIM 65 zu verkaufen, 4K RAM, BASIC, Assembler, Gehäuse, Netzgerät. VB 1.150,- J. Thiel, Tel. 04101 - 22 711.

CRT-2 Graphikerweiterung: 8192 Punkte m. einblendbarer Schrift! 4K Character-EPROM, 6502-Treibersoftware und BASIC-Plotprogramm DM 75,-. Info gegen Freiumschlag. Roland Mester, Kleiststr. 8, 6054 Rodgau 3.

Wegen Systemumstellung: Original BASIC ROMs mit Handbuch für AIM 65 zu verkaufen. VB DM 140,-. R. Murswiek, Tel. 06241 - 54 217.

******** CBM - Hardware ********

Platzprobleme? ROM-Box schaltet mit POKE-Befehl 16 Eproms um für DM 395. FLOPPY SPEED UP-KIT verdreifacht die Zugriffsgeschwindigkeit auf die 8050-Diskette für DM 395. Programmiergerät für 2K und 4K Eproms im Gehäuse incl. Maschinensprache-Software DM 395. 4KBytes Ramplatine in jede ROMfassung steckbar DM 225. Info gratis:

Martin Roßmöller, Kaiserstr. 34, 53 Bonn, Tel. 02 28 - 22 54 03

CBM-2/3/4/8001 Speicherbeleg., 630 Adr. 20 DM * Beschreibung von ROM-Routinen für reelle Arithm , Tape- & IEEE-Bus-I/O, Parameterübergaben an Maschinenprogramme etc. 25 DM * Zusammen 83 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.

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

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,--

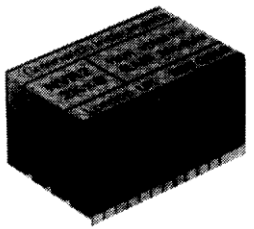
ISAM-Routinen
Datenhaltungsprogramm, Indexed Sequential Accesss Method
siehe Besprechung in Heft 23 des 65xx MICRO MAG, DM 298,--

Stellberg Computer-Systeme

COMMODORE EPSON C' ITOH SOFTWARE INTERFACE

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

Nichtflüchtige RAM-Module



Ausführliche Datenblätter auf Anfrage

Preise(1-5 Stück) DM/St.

	netto	inkl. MWSt, V&V
NVR2	140	163,85
GR2516, GR2716	170	197,75
GR2532, GR2732	240	276,85
GR2364	370	423,75
GR2764, 28-polig,	370	423,75

TecSys GmbH

Arabellastraße 13, 8000 München 81
Tel. (089) 91 39 34 und 91 48 28

* Daten bleiben 10 Jahre erhalten
* Eingebauter Schutz gegen Stromausfall
* 250 ns statisches RAM
* Lithium-Batterie-Unterstützung
* ROM/EPROM Versionen mit externem R/W-Anschluß

RAM-Pinout
NVR2 ist ein Baustein, welcher mit gängigen 2kx8 RAM (6116 usw.) pinkompatibel ist. Innen ist aber CMOS-RAM mit niedrigstem Strombedarf, eine Lithiumbatterie und entspr Umschaltelektronik, d.h. Programme/Daten bleiben bei Stromausfall erhalten.

ROM/EPROM-Pinout
Diese Art nichtflüchtigen RAM's ist auch im ROM/EPROM Emulatorformat erhältlich. Die Bausteine haben eine Kapazität von 2K, 4K, 8K. Unentbehrlich für Programmentwicklung!

AIM- und CBM-Besitzer!
Wir haben ein besonderes Informationsblatt.

Sockel-Adapter,
24 polig in mehreren Ausführungen

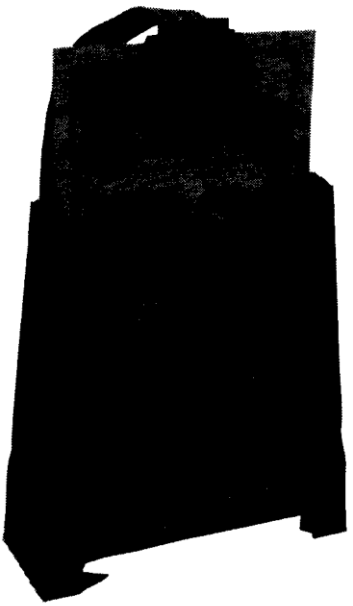
VC 20 Commodore

Zubehör und Software ab Lager
Sprechen Sie mit uns über den Preis

Helmut Rennen
GmbH & Co. KG.
Büro-Organisation + Datenverarbeitung

Tel. (02 11) 30 60 90
Martinstr. 55 · 4000 Düsseldorf 1

Relaisplatine E-941



Ihr Computer schaltet Leistung mit der Relais-Platine E-941.

Die Platine wird über einen 16-poligen Flachkabel-Steckverbinder an das Parallelport Ihres Rechners angeschlossen. So können Sie z.B. vom AIM Applikationsstecker die vier Steuerkanäle für die Relais und die notwendige Versorgungsspannung +5V= und +24V=, sowie die Groundleitung über das Flachbandkabel zur Platine führen. Die Rechnerausgänge sind auf der Platine durch ein Treiber IC und durch Optokoppler geschützt.

Auf der Platine selbst werden die gesetzten Ausgänge des Rechners nochmals angezeigt, so daß Sie auf einen Blick feststellen können, ob zum gesetzten Ausgang auch das zugehörige Relais angezogen hat. Die vier Relais haben je zwei Wechslerkontakte, die Sie mit max. 1700 VA bei Wechselspannung und mit 250 Watt bei Gleichspannung belasten können. Die Relais-Spulen sind mit Löschdioden beschaltet.

Als Steckverbinder für die Leistungsseite wurde ein DIN 41612.F Steckverbinder im 5,08 Raster verwandt. Der Steckkartenhalter ist mit Schraubklemmen ausgerüstet und erlaubt den Schaltschrankeinbau der Relaisplatine E 941.

Die Versorungsspannung muß nicht unbedingt über das Flachkabel geführt werden, sondern kann auch innerhalb des Schaltschrankes über die Schraubklemmen 1 + 2 in die Platine geleitet werden.

Preis pro Stück DM 275,00 incl. MWSt.
Staffelpreise auf Anfrage.

STECKER

INGENIEURBÜRO

Postfach 60 07 66 5000 Köln 60 Tel.: (0221) 7 12 40 18

NEUE TELEFONNUMMER!!!

micCON
INDIVIDUAL SOFTWARE
& SYSTEM ENGINEERING

AIM 65/PC 100

aktuell: **Typenradschreibmaschine Olivetti Praxis 35**
alle Zeichen und Funktionen von AIM 65/PC 100 steuerbar — P35/ME — DM 1398,-

EPROM für Monitorbereich Fxxx mit Groß-und Kleinschreibung im Editor unter Beibehaltung aller Editorfunktionen. Incl. Treiber für P35/ME Macht aus dem AIM 65 ein Textsystem. Deutscher Zeichensatz vom AIM 65 generierbar — AIM-F-P35 — DM 68,-

aktuell: **EuroAIM 65 - AIM 65 auf Europakarte**
100 % soft- und hardwarekompatibel
EuroAIM 65 CPU-Karte — SS 50/E 003-01 — DM 468,-
EuroAIM 65 I/O-Karte — SS 50/E 207-00 — DM 498,-

aktuell: **MFE mit SS 50/E - die Adapterkarte, die mehr kann.**
Decodierung des Axxx-Bereichs mit Fassung für EPROM 2532 für Axxx Bereichsausbau. Single Step Erweiterung für die Bereiche B000-DFFF. Single Step jetzt mit Video möglich. Bufferung von Daten- Adreß- und Steuerbus. 64-pol. VG-Leiste für SS 50/E Buserweiterung, damit erweiterbar mit allen EuroAIM 65 Baugruppen (z.Z. 12 verschiedene Baugruppen) — SS 50/E 401-01 — DM 248,-

PET/CBM

aktuell: **Typenradschreibmaschine Olivetti Praxis 35**
alle Zeichen und Funktionen über IEC 625 oder IEEE 488 Bus steuerbar. Interface mit eigener 6502 CPU und ca. 1k-Byt Datenpuffer. Mit PET/CBM Sonderzeichensatz. Deutscher Zeichensatz von jedem PET/CBM generierbar — P35/M625 — DM 1798,-
P35/M488 — DM 1798,-

An alle Selbstbau-Freaks

Original Olivetti Praxis 35 - Die Typenradschreibmaschine, die mehr kann. (Mit Umbauanleitung für Interfaceanschluß) — P35 — DM 1159,-

Alle Preise incl. MwSt.

Wir senden Ihnen gerne Informationsmaterial. Vermerken Sie bitte auf Ihrer Anfrage, was für Sie von Interesse ist.

MICCON • BERNHARD HECKL
ALTE WALLENSTEINSTRASSE 146
8500 NÜRNBERG 80
TELEFON 0911 / 651747

FORTH für CBM-Rechner

Leistungsfähiger Forth-Compiler auf dem deutschsprachigen Markt

Forth ist eine strukturierte Programmiersprache, die Programme in kompakter Form kompiliert und dabei eine sehr schnelle Ausführungszeit erzielt. So wird zum Beispiel eine leere Schleife 10.000 mal in einer Sekunde durchlaufen. Ferner bietet sie die Möglichkeiten der modularen und strukturierten Programmierung. Die Forth Interest Group (FIG) hat die Sprache Forth 1978 standardisiert.

Computer-Service Lowinski bringt nun einen Standard FIG Forth Compiler für cbm-Systeme 3001/4001/8001 (jeweils für 16 und 32 KB) auf den deutschsprachigen Markt. Das Grundpaket enthält alle FIG-Forth-Befehle, einen zeilenorientierten Editor sowie einen Makro-Assembler, mit dem Erweiterungen des Forth-Wortschatzes auch in Maschinensprache möglich sind. Um die Commodore-Diskettenstation betreiben zu können, ist ein Diskettenhandler inbegriffen. Er bietet wie in BASIC Möglichkeiten der Programm- und Datenspeicherung. Darüber hinaus ist es möglich, virtuelle Arrays auf der Diskette zu generieren. Das System holt sich hierzu die benötigten Variablen selbständig in einen Blockspeicher und stellt sie dann den normalen Funktionen zur Verfügung. Wenn eine Variable verändert wird, wird diese wieder auf die Diskette zurückgeschrieben. In Verbindung mit den Floppy-Stationen cbm 4040 und 8050 (DOS 2.1) können relative Files verarbeitet werden.

Als Erweiterung zum Grundpaket steht ein **Floating Point Package** zur Verfügung, das die Verwendung aller vom cbm her bekannten Realarithmetik (inklusive der Funktionen) gestattet,. Das System ermöglicht es, Realzahlen einzugeben, Variablen zu definieren und auf der Diskette abzuspeichern. Dieses Paket erweitert die Integer-Rechenfähigkeit des Forth.

Ein weiteres Zusatzpaket bietet die im Standard-Forth nicht vorhandene **String-Verwaltung,** wobei alle String-Funktionen wie in BASIC benutzt werden können. Jeder String darf, im Gegensatz zu BASIC, im Rahmen des verfügbaren Speicherplatzes beliebig lang sein. Zusätzlich gibt es den Variablentyp String-Pointer.

Alle Versionen sind ab sofort lieferbar.

Grundpaket	DM 250,--
Floating Point Package	DM 150,--
String-Paket	DM 150,--
Gesamtpaket	DM 480,--
Bedienungsanleitung (engl.)	DM 29,-- (wird bei Kauf angerechnet)

Software
Wir bieten weiter diverse Standardprogrammpakete (Bauhauptgewerbe, Versicherungsmakler Fakturierung, Finanzbuchhaltung, Lohnabrechnung). Darüberhinaus erstellen wir individuelle, kundenspezifische Programme. Bitte fordern Sie Prospektmaterial an.

Hardware

16 KB EPROM-Karte
4 x 4 KB Erweiterung für cbm-kompatible ROMs und für EPROMs 2532, 2732 und 2716 (untere und obere 2 KB Einstellbar), durch Programm umschaltbar DM 285,--

48 KB EPROM-Karte
Je 6 cbm-kompatible EPROMs/ROMs für 2 x 4 KB Bereiche, durch Programm umschaltbar DM 465,--

Uhren-Karte, Batterie-gepuffert
Uhrzeit, Datum (inkl. Schaltjahr) bleiben auch nach dem Abschalten erhalten DM 285,--

40-Bit-Parallelport
Vier 8-Bit-Parallelports + je zwei Handshake-Leitungen DM 565,--

EPROM-Programmiergerät für EPROMs 2532, 2732, 2716 DM 890,--

Umfangreiche Software mit $FF-Test, Verify, Ausdruck, Diskketteninterface etc. DM 50,--

Alle Preise inkl. MWSt., ohne Fracht und Verpackung.

WOLFGANG LOWINSKI
Computer-Service

Gallwitzstraße 10
7800 Freiburg im Breisgau
Telefon (0761) 403593

Computer-Baugruppen im Europaformat

Einheitliches Format Einfach-Europakarte 100x160 mm, nur 5V Stromversorgung
MCS-Bus für AIM65-kompatible Karten, VME-Bus für 6809-Karten

6502 Einplatinen-Computer mit 20 kB ROM/EPROM/RAM-Plätzen zur freien Verfügung, VIA, RIOT, d.h. 36 E/A-Leitungen, 3 Timer, serielle Schnittstelle, Anschluß für unkodierte Tastatur. Erweiterbar mit Speicher- und Videokarte. Bereits in der kleinsten Ausbaustufe läuft die software des AIM65 (Monitor, Assembler, Basic, Forth u.s.w.). Preis: ab 580,--DM.

6809 single-board-computer - auch als seriell gekoppelter Zusatz für AIM65. mit CPU (*), VIA, 2 ACIAs (1*), RS232C (*), Timer 6840 (*), 4 EPROM/RAM-Steckplätze (1*). Minimalkonfiguration (*): 450,--DM.
ROM-Satz (6809-Monitor, AIM65-Terminal-ROM) 70,--DM.

6801 Ein-chip Computer auf Europakarte für Steuerungszwecke und Programmentwicklungen. Mit Monitor-ROM, 2k RAM, 29 + 2 E/A-Leitungen, Timer, vollduplex seriellem Interface, adaptierbarem Bus, wire-wrap Experimentierfeld. 450,-DM.
MC6801-Applikationshandbuch (hard- und software, 420 Seiten, engl.) 29,80 DM.

Speicherbaugruppen im Einfach-Euroformat:
Statisch mit autonomem Refresh ohne Einfluß auf das CPU-Timing. Nur eine 5V-Betriebsspannung. MCS-Bus für AIM65-Bus-Expansion. VME-Bus für 6809-Systeme.

128 kB 996,--DM. dito teilbestückt 64 kB 792,--DM.
256 kB 1.292,--DM. 512 kB 2.280,--DM.
1 MB, 2MB -Speicherbaugruppen im Einfach-Europakartenformat in Vorbereitung.

64 kB Sonderausführung für unmittelbaren Einbau in Siemens PC100 - der Bus-Exp.-Stecker J3 bleibt frei, ROM-Sockel werden nicht verdeckt, die vorhandene Stromversorgung reicht aus, Adressierung in 2x32 kB -banks, Format 100x100 mm. 782,--DM. Dito 128 kB (4 banks zu 32kB) add-on-RAM für PC100/AIM65: 895,--DM.

32 kB CMOS-RAM batteriegepuffert, 200ns, für alle 8-Bit CPUs: 998,92 DM.

Video-Interface 24x80 Zeichen, mit exklusiver Texteditierung
Beschreibung wie in Nr.23 des MICRO MAG (Anzeige) oder im Prospekt (anfordern!)
Video-Interface 694,35 DM, Option 1 (neue Monitor-ROMs für AIM65) 97,-DM

Bus-Expansion für AIM65:
besteht aus AIM-Adapter mit Daten- und Adreßpufferung, Schreibschutzschaltern f.16x4 kB, programmierbarem bankselect, Verbindungsflachkabel und Buskarte mit 7 (erweiterbar) Steckplätzen, aktivem Busterminator. 271,20 DM.

P35/2 Mikroprozessor-gesteuerte tragbare Schreibmaschine als Typenraddrucker mit integrierter Centronics-Parallel-Schnittstelle. Max. 12 Zeichen/Sek., automat. Umschaltung von KB1 zu KB2, 3 Schriftgrößen 10/12/15 Zeichen/Zoll, Carbon- und Textilfarbband, zahlreiche Typenräder lieferbar. Wir bieten Olivetti-Service!
Olivetti Praxis 35 Schreibmaschine ohne Interface 1.230,--DM
P35/2 Drucker/Schreibmaschine mit Centronics-Schnittstelle 1.750,--DM.
- zusätzlich mit RS232C -Empfänger 1.950,--DM
- zusätzlich mit IEC-Bus-Interface für cbm, einschl. Bus-Kabel 2.150,--DM
- als Vollduplex-Druckerterminal 2.450,--DM
- andere Ausführungen (P2000,VC20,PC100 etc.) auf Anfrage

Philips MDCR-Laufwerk 380,-DM. AIM65-Drucker 85,-DM. Alle Preise incl.MwSt.

DIPL.-ING. HORST NEUDECKER

INGENIEURBÜRO FÜR MESSTECHNIK

Mehringplatz 13, 1000 Berlin 61, Tel.: 030 - 251 20 00 und 262 45 18

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..COPYRIGHT 1982 by Roland Löhr. Alle Rechte vorbehalten, auch die des auszugsweisen Nachdruckes, der Übersetzung, der fotomechanischen Wiedergabe und die der Verbreitung auf magnetischen und sonstigen Trägern.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 49,-- (Inlandsendpreis). Ausland/foreign via surface mail DM 54,--, 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 14-24 sind unbeschränkt nachlieferbar.

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 65xx MICRO MAG
ca, 230 Seiten, Sammelband der Hefte 1-6 dieser Zeitschrift, gebunden, ohne Anzeigen. Das wertvolle Arbeitsbuch mit einem Leitfaden für die Programmierung und vielen anderen grundlegenden Darstellungen — DM 26,--

Das Buch 7-13 des 65xx MICRO MAG neu !
Sammelband ohne Anzeigen, ca. 340 Seiten, geb., enthält neben etwa 20 allgemeinen Darstellungen 25 Artikel und Programme für CBM/PET sowie 35 für AIM 65/PC 100. Eine Fundgrube für die fortschreitende Systemnutzung — DM 42,--

6502 Software Design
von L. J. Scanlon. Das beliebte Lehrbuch für die Programmierung in Maschinensprache ca. 270 Seiten, engl., mit vielen verständlich aufgebauten Beispielen — DM 36,--

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 59,--

AIM 65 Laboratory Manual And Study Guide
von L. J. Scanlon (jr.). ca. 185 Seiten, engl. Lehrbuch mit vielen Übungen zur Benutzung und Programmierung des AIM. Sehr geeignet für Anfänger — DM 26,--

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 Endpreise einschl. Verp. und Porto. Nachnahme: +DM 1,50

FORTH-Programmierkursus am 24./25. September 1982 in Ahrensburg bei Hamburg.
Prospekt und Anmeldung beim Herausgeber.