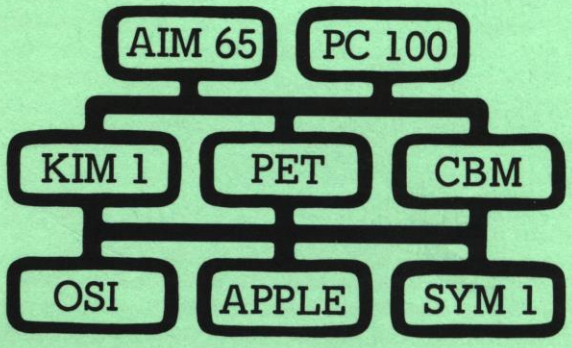
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 9,50 Nr. 27 Oktober 1982



Inhaltsverzeichnis

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

1-Chip Mikroprozessor 6805 und 68705

Hardware

Motorola liefert eine ganze Familie von 1-Chip Mikroprozessoren, die gemeinsam auf dem MC6805 fußen. Es gibt solche mit maskenprogrammiertem ROM und solche mit EPROM, RAM, Timer und Urlader-ROM on chip. Letztere tragen die Typenbezeichnungen MC68705xx. Diese sind von besonderem Interesse, wenn es um kleine Serien und Prototypen geht. Insbesondere wird man mit ihnen intelligente Interfaces und selbständige Steuerungen aufbauen, bei denen von Anlage zu Anlage etwas verschiedene Parameter der zu steuernden Maschinen zu berücksichtigen sind.

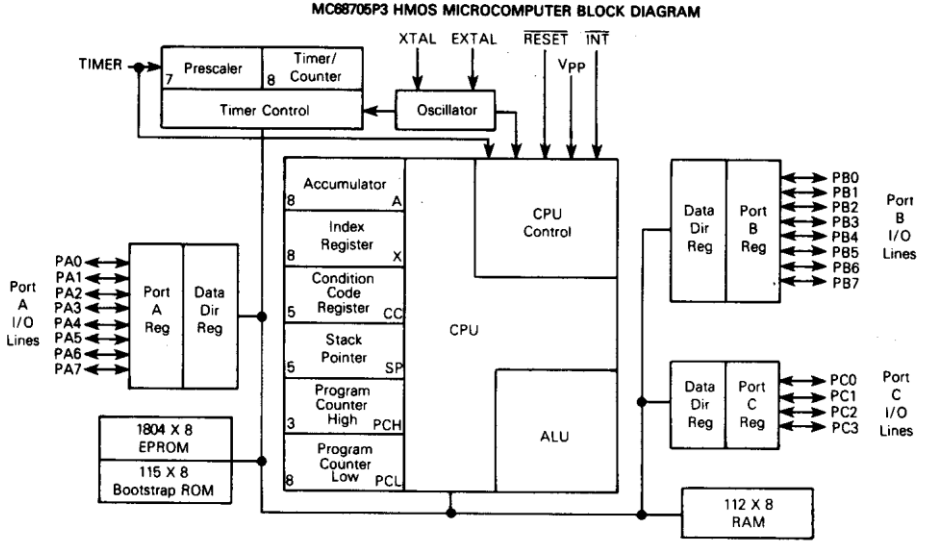
Es ist hier nicht der Platz, jedes einzelne Glied der Produktfamilie vorzustellen. Es ist jedoch auf folgende Gruppierung hinzuweisen: Es gibt diese 1-Chipper mit 28 und 40 Pin, entsprechend mit 20 oder mit 32 i/O-Pin, es sind also auf Interfaces ausgerichtete Mikroprozessoren. Bei den Typen mit 40 Pin gibt es Versionen, in denen sich 8 Pin als Eingänge für 4 A/D-Wandler benutzen lassen. Und es gibt zwei CMOS-Versionen, mit ROM bzw. mit gemultiplextem Adreß-/Datenbus.

Die Versionen unterscheiden sich ferner durch die Größe des RAM on chip (64 oder 112 Byte) und durch die Größe des EPROM zwischen 1,8 bis 3,8 KB (bzw. ROM).

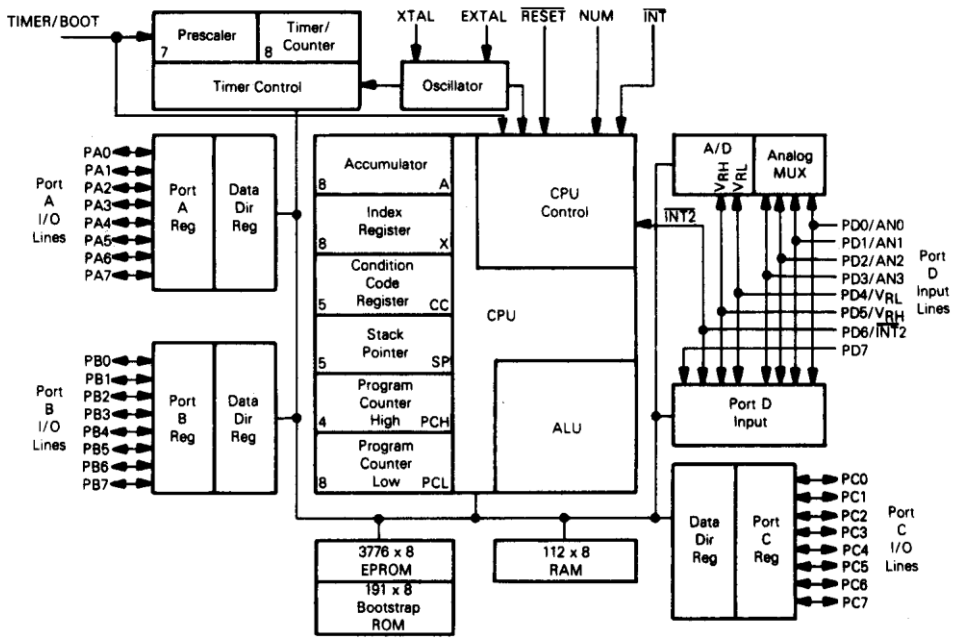
Alle Typen arbeiten mit +5 Volt, haben den Taktoszillator auf dem Chip und lassen sich durch einen Quarz oder durch ein RC-Glied bzw. durch Überbrückung der Oszillatoreingänge takten. Zur Hardware gehört ferner ein interruptfähiger Untersetzungs-Timer und ein EPROM-Urlader, der es gestattet, ein extern erstelltes und in ein EPROM abgelegtes Programm in das EPROM der Zentraleinheit einzubrennen. Die Datenblätter enthalten hierfür den Vorschlag für eine externe Beschaltung. Die maskenprogrammierten Typen enthalten statt dieses Urladers ein 'self check' ROM.

Für die Verwendung in Prototypen und in kleinen Serien werden die Typen MC68705P3 (mit 20 I/O-pin) und MC68705R3 (32 I/O-Pin bzw. 24 I/O-Pin plus 4 A/D-Kanäle) von besonderem Interesse sein. Für sie nachfolgend die Hardware-Blockdiagramme:

MC68705P3 HMOS MICROCOMPUTER BLOCK DIAGRAM



MC68705R3 HMOS MICROCOMPUTER BLOCK DIAGRAM

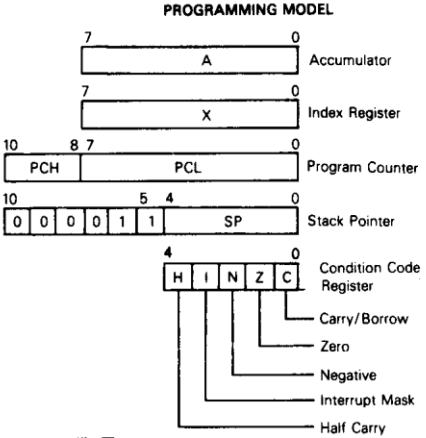


Maschinenregister

Die Register der Maschine sind im nachfolgenden 'Programming Model' dargestellt. Die Hauptregister Akku und Indexregister X sind jeweils 8 Bit breit, das Statusregister (Condition Code Register) entspricht dem des 6800. Der Programmzähler ist 11 oder 12 Bit breit, ausreichend, um die Adreßräume von 2K bzw. 4K zu überstreichen.

Der Hardwarestack der 6805 liegt immer in der 0-Speicherseite, in der allein ja RAM ansprechbar ist. Bei einem RESET wird der Stackpointer auf hex 07F gesetzt. Für Unterprogramme und Interrupts stehen maximal 31 Ablagestellen auf dem Stack zur Verfügung. Die Speicherseite 0 enthält neben dem RAM für Arbeitsspeicher (hex 11-60) und Stack (61-67) auch von 00-0F die memory mapped Adressen für die Ports, Datenrichtungsregister und weitere Kontrollregister (Timer, A/D-Wandler). Ab Adresse 80 beginnt der EPROM-Bereich.

PROGRAMMING MODEL



Der Befehlssatz

Der Befehlsatz der CPU ist übersichtlich und symmetrisch aufgebaut und untersützt vor allem auch Befehle zum Löschen und Setzen einzelner Bits in der Nullseite mit ihren I/O-Adressen. Hierzu informiert nebenstehend ganzseitig die Übersicht Instruction Set Opcode Map.

Die beiden ersten Spalten enthalten in den Gruppen BTB (Bit Test and Branch) und BSC (Bit Set/Clear) die Möglichkeit, in Abhängigkeit jedes einzelnen Bit eine Verzweigung vorzunehmen oder jedes einzelne Bit zu löschen oder zu setzen. Die Adressierung bezieht sich immer auf die 0-Adreßseite, weil hier die Interfaces und das veränderbare RAM liegen.

Die Gruppe REL enthält 8 Gruppen von antivalenten Verzweigungsbefehlen. Neben den auch vom 6502 her bekannten Befehlen sind erwähnenswert: BRA/BRN (verzweige immer/nie), BHI/BLS (if higher/if lower or same), die beim 6502 nur durch ein Filter von 2 Verzweigungsbefehlen aufzulösen sind, sowie die Verzweigungsbefehle BMC/BMS (if interrupt mask clear/set) und BIL/BIH (wenn Interrupt-Eingangspin auf low/high). Daneben gibt es den kurzen Subroutinebefehl BSR (branch to subroutine).

Die READ/MODIFY/WRITE-Gruppe enthält die bekannten Befehle zum Verschieben und Rollieren, zum Prüfen (TST), Komplementieren, Invertieren und zum Inkrementieren/Dekrementieren (alle auch für den Akku und Register X). Auf ihre Adressierungsart wird weiter unten eingegangen.

Die Kontrollbefehle dürften in ihrer Wirkung weitgehend bekannt sein. SWI entspricht dem BRK, RSP = Reset Stack Pointer. WAIT und STOP für die CMOS-Typen verringern die Leistungsaufnahme.
Die letzte große Gruppe (Register/Memory) umfaßt die bekannten arithmetischen, logischen, die Vergleichsbefehle sowie JMP und JSR (BSR).

Dem Programmierer eines 68er oder 65er bereitet der Befehlssatz des 6805 damit keinerlei Schwierigkeiten, das gilt auch für die Adressierung.

Die Adressierungsarten

Der Mechanismus der Verzweigungsbefehle ist wie beim 6800 oder 6502 gewohnt. Auch die Inherent-Befehle sind nicht besonders zu erwähnen.

Die Gruppe BSC (Bit Set/Clear) kann nur auf das I/O und das RAM der 0-Speicherseite angewandt werden., ebenso die Adressierungsart DIR (Direct Page). Die Adressierungsarten IMM (immediate) bedeutet, wie gewohnt, Direktoperand. EXT entspricht 'extended' oder 'absolut' mit nachfolgender effektiver Adresse in 2 Byte.

Es bleiben die indizierten Adressierungsarten. IX bedeutet 'indiziert ohne Offset'. Der Inhalt von Register X wird also als die effektive Adresse in der 0-Seite benutzt, geeignet, um z.B. Tabellen (auch im Anfang des EPROM von 080 bis 00FF) zu überstreichen.

IX1 bedeutet Offset 8 Bit gegen Register X (der Offset steht in dem Byte, das dem Opcode folgt). IX2 bedeutet 2-Byte Offset gegen Inhalt des Registers X. Bei dieser Adressierungsart wird man die Tabellen-Basisadresse in den Befehl schreiben und hernach X über die Tabellenelemente streichen lassen. Wenn man will, dem 6502 verwandt bei ABSOLUT,X.

Dokumentation

Für jeden Typ der Familie gibt es ein eigenes ausführliches Datenblatt, das für den Designer alle interessierenden Fragen einschl. Schaltungsvorschlägen abhandelt. Der Programmierer sollte zusätzlich das 'M6805 M146805 Benützer-Handbuch' (Doc. M6805UM(AD)) heranziehen.

Design 5IVE Kit

Für den schnellen Aufbau eines ersten Experimentiersystems liefern die Motorola-Distributoren einen Baukasten, der den hier besonders erwähnten 68705P3 (1,8K EPROM, 24 Pin, davon 20 I/O) nebst Datenblättern für die Familie enthält (Preis ca. DM 204,- + MWSt). Daneben sind Bausteine und Datenblätter für einfache Interfaces und für das Einbrennen enthalten: MC14447P A/D Converter Subsystem, MC146818P Real Time Clock + RAM, MC144110P 6-Kanal D/A-Konverter,

M6805 FAMILY INSTRUCTION SET OPCODE MAP

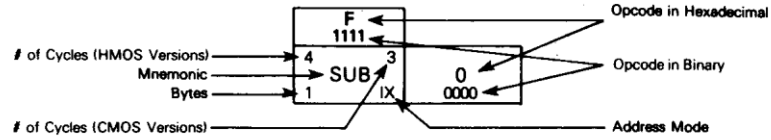
	Bit Manipulation		Branch	Read/Modify/Write					Control		Register/Memory						
	BTB 0 0000	BSC 1 0001	REL 2 0010	DIR 3 0011	INH(A) 4 0100	INH(X) 5 0101	IX1 6 0110	IX 7 0111	INH 8 1000	INH 9 1001	IMM A 1010	DIR B 1011	EXT C 1100	IX2 D 1101	IX1 E 1110	IX F 1111	Hi / Low
0 0000	10 5 BRSET0 3 BTB	7 5 BSET0 2 BSC	4 3 BRA 2 REL	6 5 NEG 2 DIR	4 3 NEGA 1 INH	4 3 NEGX 1 INH	7 6 NEG 2 IX1	6 5 NEG 1 IX	9 9 RTI 1 INH		2 2 SUB 2 IMM	4 3 SUB 2 DIR	5 4 SUB 3 EXT	6 5 SUB 3 IX2	5 4 SUB 2 IX1	4 3 SUB 1 IX	0 0000
1 0001	10 5 BRCLR0 3 BTB	7 5 BCLR0 2 BSC	4 3 BRN 2 REL						6 6 RTS 1 INH		2 2 CMP 2 IMM	4 3 CMP 2 DIR	5 4 CMP 3 EXT	6 5 CMP 3 IX2	5 4 CMP 2 IX1	4 3 CMP 1 IX	1 0001
2 0010	10 5 BRSET1 3 BTB	7 5 BSET1 2 BSC	4 3 BHI 2 REL								2 2 SBC 2 IMM	4 3 SBC 2 DIR	5 4 SBC 3 EXT	6 5 SBC 3 IX2	5 4 SBC 2 IX1	4 3 SBC 1 IX	2 0010
3 0011	10 5 BRCLR1 3 BTB	7 5 BCLR1 2 BSC	4 3 BLS 2 REL	6 5 COM 2 DIR	4 3 COMA 1 INH	4 3 COMX 1 INH	7 6 COM 2 IX1	6 5 COM 1 IX	11 10 SWI 1 INH		2 2 CPX 2 IMM	4 3 CPX 2 DIR	5 4 CPX 3 EXT	6 5 CPX 3 IX2	5 4 CPX 2 IX1	4 3 CPX 1 IX	3 0011
4 0100	10 5 BRSET2 3 BTB	7 5 BSET2 2 BSC	4 3 BCC 2 REL	6 5 LSR 2 DIR	4 3 LSRA 1 INH	4 3 LSRX 1 INH	7 6 LSR 2 IX1	6 5 LSR 1 IX			2 2 AND 2 IMM	4 3 AND 2 DIR	5 4 AND 3 EXT	6 5 AND 3 IX2	5 4 AND 2 IX1	4 3 AND 1 IX	4 0100
5 0101	10 5 BRCLR2 3 BTB	7 5 BCLR2 2 BSC	4 3 BCS 2 REL								2 2 BIT 2 IMM	4 3 BIT 2 DIR	5 4 BIT 3 EXT	6 5 BIT 3 IX2	5 4 BIT 2 IX1	4 3 BIT 1 IX	5 0101
6 0110	10 5 BRSET3 3 BTB	7 5 BSET3 2 BSC	4 3 BNE 2 REL	6 5 ROR 2 DIR	4 3 RORA 1 INH	4 3 RORX 1 INH	7 6 ROR 2 IX1	6 5 ROR 1 IX			2 2 LDA 2 IMM	4 3 LDA 2 DIR	5 4 LDA 3 EXT	6 5 LDA 3 IX2	5 4 LDA 2 IX1	4 3 LDA 1 IX	6 0110
7 0111	10 5 BRCLR3 3 BTB	7 5 BCLR3 2 BSC	4 3 BEQ 2 REL	6 5 ASR 2 DIR	4 3 ASRA 1 INH	4 3 ASRX 1 INH	7 6 ASR 2 IX1	6 5 ASR 1 IX		2 2 TAX 1 INH		5 4 STA 2 DIR	6 5 STA 3 EXT	7 6 STA 3 IX2	6 5 STA 2 IX1	5 4 STA 1 IX	7 0111
8 1000	10 5 BRSET4 3 BTB	7 5 BSET4 2 BSC	4 3 BHCC 2 REL	6 5 LSL 2 DIR	4 3 LSLA 1 INH	4 3 LSLX 1 INH	7 6 LSL 2 IX1	6 5 LSL 1 IX		2 2 CLC 1 INH	2 2 EOR 2 IMM	4 3 EOR 2 DIR	5 4 EOR 3 EXT	6 5 EOR 3 IX2	5 4 EOR 2 IX1	4 3 EOR 1 IX	8 1000
9 1001	10 5 BRCLR4 3 BTB	7 5 BCLR4 2 BSC	4 3 BHCS 2 REL	6 5 ROL 2 DIR	4 3 ROLA 1 INH	4 3 ROLX 1 INH	7 6 ROL 2 IX1	6 5 ROL 1 IX		2 2 SEC 1 INH	2 2 ADC 2 IMM	4 3 ADC 2 DIR	5 4 ADC 3 EXT	6 5 ADC 3 IX2	5 4 ADC 2 IX1	4 3 ADC 1 IX	9 1001
A 1010	10 5 BRSET5 3 BTB	7 5 BSET5 2 BSC	4 3 BPL 2 REL	6 5 DEC 2 DIR	4 3 DECA 1 INH	4 3 DECX 1 INH	7 6 DEC 2 IX1	6 5 DEC 1 IX		2 2 CLI 1 INH	2 2 ORA 2 IMM	4 3 ORA 2 DIR	5 4 ORA 3 EXT	6 5 ORA 3 IX2	5 4 ORA 2 IX1	4 3 ORA 1 IX	A 1010
B 1011	10 5 BRCLR5 3 BTB	7 5 BCLR5 2 BSC	4 3 BMI 2 REL							2 2 SEI 1 INH	2 2 ADD 2 IMM	4 3 ADD 2 DIR	5 4 ADD 3 EXT	6 5 ADD 3 IX2	5 4 ADD 2 IX1	4 3 ADD 1 IX	B 1011
C 1100	10 5 BRSET6 3 BTB	7 5 BSET6 2 BSC	4 3 BMC 2 REL	6 5 INC 2 DIR	4 3 INCA 1 INH	4 3 INCX 1 INH	7 6 INC 2 IX1	6 5 INC 1 IX		2 2 RSP 1 INH		3 2 JMP 2 DIR	4 3 JMP 3 EXT	5 4 JMP 3 IX2	4 3 JMP 2 IX1	3 2 JMP 1 IX	C 1100
D 1101	10 5 BRCLR6 3 BTB	7 5 BCLR6 2 BSC	4 3 BMS 2 REL	6 4 TST 2 DIR	4 3 TSTA 1 INH	4 3 TSTX 1 INH	7 5 TST 2 IX1	6 4 TST 1 IX		2 2 NOP 1 INH	8 6 BSR 2 REL	6 5 JSR 2 DIR	8 7 JSR 3 EXT	9 8 JSR 3 IX2	8 7 JSR 2 IX1	7 6 JSR 1 IX	D 1101
E 1110	10 5 BRSET7 3 BTB	7 5 BSET7 2 BSC	4 3 BIL 2 REL						2 2 STOP* 1 INH		2 2 LDX 2 IMM	4 3 LDX 2 DIR	5 4 LDX 3 EXT	6 5 LDX 3 IX2	5 4 LDX 2 IX1	4 3 LDX 1 IX	E 1110
F 1111	10 5 BRCLR7 3 BTB	7 5 BCLR7 2 BSC	4 3 BIH 2 REL	6 5 CLR 2 DIR	4 3 CLRA 1 INH	4 3 CLRX 1 INH	7 6 CLR 2 IX1	6 5 CLR 1 IX	2 2 WAIT* 1 INH	2 2 TXA 1 INH		5 4 STX 2 DIR	6 5 STX 3 EXT	7 6 STX 3 IX2	6 5 STX 2 IX1	5 4 STX 1 IX	F 1111

Abbreviations for Address Modes

INH	Inherent
IMM	Immediate
DIR	Direct
EXT	Extended
REL	Relative
BSC	Bit Set/Clear
BTB	Bit Test and Branch
IX	Indexed (No Offset)
IX1	Indexed, 1 Byte (8-Bit) Offset
IX2	Indexed, 2 Byte (16-Bit) Offset

*CMOS only

LEGEND



16x16 Bit EEPROM MCM2801P, MC14499 7-Segment LED-Treiber, MC144115P 16-Segment LCD-Treiber und MC14040BCP 12-Bit Zähler.

Interessant ist ferner die enthaltene Schrift 'Desing 5IVE Applications' (ca. 60 Seiten plus zwei Schaltpläne für den Betrieb von Interfaces) mit Assembler-Listen für wichtige Systemprogramme, z.B. Port- und RAM-Zuweisung. Für dieses Kit gibt es eine kleine Platine, die jedoch bei Abschluß dieses Heftes hier noch nicht vorlag.

Software-Entwicklung

Zu jedem Computer gehört mindestens ein guter Assembler, den es natürlich auch von Motorola gibt. Es ist aber nicht unbedingt erforderlich, für die Programmierung ein besonderes Entwicklungssystem anzuschaffen. Die preiswertere Möglichkeit besteht in der Benutzung eines Cross-Assemblers, der auf einer anderen Maschine den 6805 Zielcode generiert. Als bequemes Vehikel hierzu bietet sich die Sprache FORTH an. Der Herausgeber hofft, etwa zu Ende Oktober einen entsprechenden Cross-Assembler zur Lieferung freigeben zu können, ebenso einen für den MC6809.

Die in diesem Artikel enthaltenen Übersichten wurden der Motorola-Dokumentation entnommen.

R.L. #

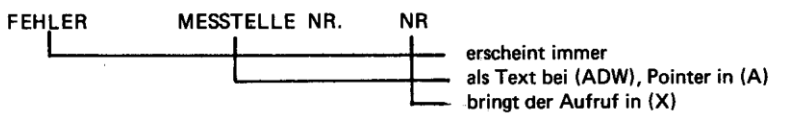
Bernhard Kokula, 6700 Ludwigshafen

Prozeßtechnik mit Mikroprozessoren (5)

In Heft 25, Seite 54ff. wurde der Dreipunktregler dargestellt. Dieses Modul wird um Error-Meldungen und um einen Test ergänzt:

Error-Meldungen

Das Modul (ERR) kann Fehlermeldungen aus verschiedenen Modulen bedienen. Dazu gibt es aus:



(X) könnte auch Pointer zu individuellem Meßstellen-Text sein. Mit wenig Textspeicher läßt sich so eine eindeutige Fehleraussage erzielen. (ERR) kann weiterhin über ein Ausgangsbit einen Alarm auslösen. Bei Aufruf von (ERR) muß dazu das Carry-Flag=1 sein. Mit ALQUIT kann der Alarm quittiert werden. Ob nach dem Fehler das Programm weiterarbeitet oder abbricht, liegt in der Hand des Programmierers, (ERR) kehrt zum aufrufenden Modul zurück.

Test-Programm

Mit dem Test-Programm können alle drei Module getestet werden. Es zeigt die Daten in den 'HEX-REG' im AIM-Display und bringt Fehlermeldungen auf den AIM-Printer. Ein Leitungsbruch z.B. am Meßwiderstand des 555-Timers führt zu einer Meßzeit von mehr als 65 ms. Der Timer der VIA 6522 hat dann einen Nulldurchgang und setzt das T1-Interrupt Flag, das hier abgefragt wird.

```
MODUL: <ERR>  ERROR MELDUNGEN
              02B4        0194
02B4          0195 ;DAS AUFRUFENDE PROGRAMM BRINGT DIE NOETIGEN
02B4          0196 ;VEKTOREN MIT:
02B4          0197 ;<X> = MESSTELLEN NUMMER
02B4          0198 ;<A> = TEXT-VEKTOR
02B4          0199 ;<C> = 0: KEIN ALARM
02B4          0200 ;<C> = 1:      ALARM
02B4          0201
02B4          0202 ;+++++ VARIABLEN & KONSTANTEN <ERR>
02B4          0203
```

```
02B4          0204 ERPORT =UDRA          ;USER-6522-PORT A
02B4          0205 ALARM  = %10000000    ;BIT 7 ALARM
02B4          0206
02B4          0207        *=VARDPR       ;VARIABLEN-LINK VON <DPR>
00B7          0208 VARERR =*             ;KEINE VARIABLEN NOETIG
00B7          0209
00B7          0210        *=PRODPR       ;PROGRAMM-LINK VON <DPR>
02B4          0211 ;+++++ INITIALISIERE ALARM-PORT
02B4          0212
02B4 A980     0213 INIERR LDA #ALARM     ;ALARM-BIT
02B6 0D03A0   0214        ORA ERPORT+2   ;DATA-DIR.REG.
02B9 8D03A0   0215        STA ERPORT+2
02BC 60       0216        RTS
02BD          0217
02BD          0218 ;----- ALARM QUITTIEREN
02BD A980     0219 ALQUIT LDA #ALARM     ;ALARM-BIT
02BF 49FF     0220        EOR #$FF       ;MASKE INVERTIEREN
02C1 2D01A0   0221        AND ERPORT     ;ALARM BIT LOESCHEN
02C4 8D01A0   0222        STA ERPORT
02C7 60       0223        RTS
02C8          0224
02C8          0225
02C8          0226 ;      ++++++++++ PROGRAMM <ERROR>
02C8          0227
02C8 48       0228 ERR    PHA
02C9 9008     0229        BCC ER1        ;KEIN ALARM
02CB          0230
02CB A980     0231        LDA #ALARM     ;ALARM-MASKE
02CD 0D01A0   0232        ORA ERPORT     ;PORT IN DEM ALARM-BIT LIEGT
02D0 8D01A0   0233        STA ERPORT
02D3          0234
02D3 A000     0235 ER1    LDY #0         ;1.TEXT, KOMMT IMMER
02D5 20E402   0236        JSR TEXT       ;'FEHLER'
02D8 68       0237        PLA            ;ENTHAELT TEXT-ANFANG
02D9 A8       0238        TAY
02DA 20E402   0239        JSR TEXT       ;FEHLER TYP
02DD 8A       0240        TXA            ;MESSTELLEN NUMMER
02DE 2046EA   0241        JSR NUMA       ; -AUSGEBEN
02E1 4C4AF0   0242        JMP IPSO       ;PRINT WAS IM BUFFER
02E4          0243
02E4          0244 ;TEXT AUSGABE, '00' IST END-ZEICHEN
02E4          0245 ;----- TEXT-ANFANG KOMMT IN <Y>
02E4 B9F002   0246 TEXT   LDA TEX,Y
02E7 F006     0247        BEQ TE0
02E9 20BCE9   0248        JSR OUTALL     ;GIB TEXT AUS
02EC C8       0249        INY
02ED D0F5     0250        BNE TEXT       ;IMMER
02EF 60       0251 TE0    RTS
02F0          0252
02F0          0253 ;----- TEXT SPEICHER
02F0 0D       0254 TEX    .BYT $0D,'FEHLER: ',0 ;VORWORT
02F1 4645     0254
02F9 00       0254
02FA 4D45     0255 TE1    .BYT 'MESSTELLE ',0 ;TEXT ZU (ADW)
0304 00       0255
0305          0256 TE2                   ;TEXT ANDERER MODULE
0305          0257
0305          0258 PRDERR =*
0305          0259
```

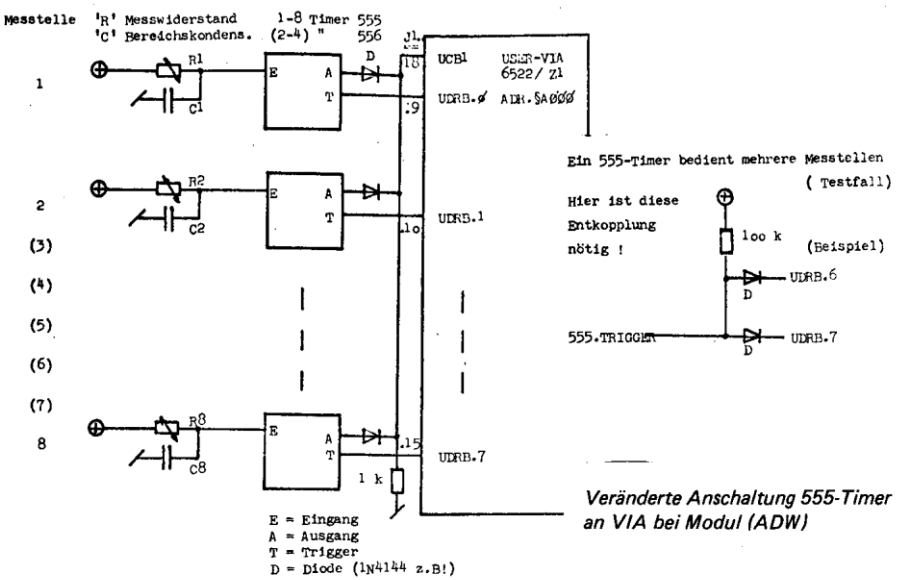
```
0305          0260 ;----- AIM ADRESSEN (ERROR)
0305          0261 IPSO   =$F04A          ;PRINTOUT (BUFFER)
0305          0262 NUMA   =$EA46          ;AUSGABE (HEX)
0305          0263 OUTALL =$E9BC          ;AUSGABE (ASCII)
0305          0264
------------------------------------------------------------
TESTPROGRAMM (3-PUNKTREGLER)
0305          0266 ;ES BENUTZT DIE MODULE:
0305          0267 ;(ADW) ANALOG/DIGITAL-WANDLER 555
0305          0268 ;(DPR) 3-PUNKT-REGLER
0305          0269 ;(ERR) ERROR-MELDUNGEN
0305          0270
0305          0271        *= PROERR       ;PROGRAMM-LINK VON (ERR)
0305          0272 ;+++++ INITIALISIERE ZUERST USER-TASTE F3
0305 A94C     0273 START  LDA #$4C        ;'JMP'
0307 8D1201   0274        STA F3
030A A91D     0275        LDA #<TEST
030C 8D1301   0276        STA F3+1
030F A903     0277        LDA #>TEST
0311 8D1401   0278        STA F3+2
0314          0279
0314          0280 ;----- INITIALISIERE DIE MODULE
0314 200302   0281        JSR INIADW
0317 205D02   0282        JSR INIDPR
031A 20B402   0283        JSR INIERR
031D          0284
031D          0285
031D          0286 ;      ++++++++++ PROGRAMM (DPR-TEST)
031D          0287
031D          0288 ;----- MESSWERTE HEREINHOLEN
031D A205     0289 TEST   LDX #ZYKLUS-1   ;MESSTELLEN ANZAHL
031F 86B6     0290        STX XPOINT      ;MODUL-INDEX
0321 202102   0291 TES1   JSR ADW         ;AUFRUF A/D-WANDLER
0324 C6B6     0292        DEC XPOINT
0326 10F9     0293        BPL TES1        ;NOCH NICHT FERTIG
0328          0294
0328          0295 ;----- REGLER BEDIENEN
0328 A201     0296        LDX #DPZAHL-1   ;DPR-ANZAHL
032A 86B6     0297        STX XPOINT
032C 206602   0298 TES2   JSR DPR         ;AUFRUF 3-PUNKT-REGLER
032F C6B6     0299        DEC XPOINT      ;NAECHSTEN REGLER
0331 10F9     0300        BPL TES2
0333          0301
0333          0302 ;----- MESSWERTE ANZEIGEN
0333 20F0E9   0303        JSR CRLF        ;NEUE ZEILE
0336 A200     0304        LDX #0
0338 B5B0     0305 LOOP1  LDA HEXREG,X
033A 2046EA   0306        JSR NUMA        ;(HEX)DATE IN ASCII AUSGEBEN
033D 203EE8   0307        JSR BLANK
0340 E8       0308        INX             ;DIE NAECHSTE BITTE
0341 E006     0309        CPX #ZYKLUS     ;WAR ES DIE LETZTE ?
0343 D0F3     0310        BNE LOOP1       ;NEIN
0345          0311
0345          0312 ;ALARM QUITTIEREN MIT BELIEBIGER TASTE
0345          0313 ;----- PROGRAMM ABBRECHEN MIT 'ESC'
0345 20EFEC   0314        JSR ROONEK      ;EINE TASTE GEDRUECKT ?
0348 88       0315        DEY             ;DANN IST (Y)= 0
0349 300D     0316        BMI TES3        ;NEIN
```

```
034B 20BD02  0317          JSR ALQUIT      ;ALARM QUITIEREN
034E 203CE9  0318          JSR READ        ;HOLE TASTE
0351 C91B    0319          CMP #$1B        ;'ESC'?
0353 D003    0320          BNE TES3
0355 4CA1E1  0321          JMP COMIN       ;ZURUECK ZUM MONITOR
0358 4C1D03  0322 TES3     JMP TEST        ;AUF EIN NEUES
035B         0323
035B         0324 ;----- AIM-ADRESSEN (DPR-TEST)
035B         0325 COMIN  =$E1A1           ;ZRUECK ZUM MONITOR
035B         0326 F3     =$0112           ;USER-TASTE F3
035B         0327 BLANK  =$EB3E
035B         0328 CRLF   =$E9F0
035B         0329 READ   =$E93C           ;LESE DATE EIN
035B         0330 ROONEK =$ECEF           ;TASTATUR-ABFRAGE
035B         0331
035B         0332        .END
```

Verbesserter Analog-/Digital-Wandler ADW.4

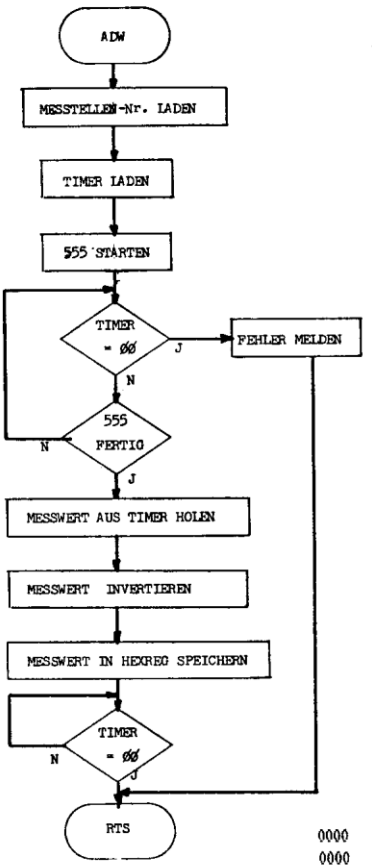
Dieses Modul ist eine verbesserte Version des im ersten Teil (Heft 20, S. 5 ff.) abgedruckten MUX 555. Es ist Voraussetzung für den Betrieb des weiter unten dargestellten Puls-Proportional-Reglers. Nachfolgend die geänderte Beschaltung.

Wie in Teil 1 schon dargestellt, wird hier jeder 555-Timer mit einem eigenen Startimpuls bedient. Dafür ist nur ein gemeinsamer Eingang für die Impulsantwort nötig. Der Vorteil dieser Lösung ist, daß die Ausgangsimpulse zugleich für andere Module bei Multiplex-Ein- und Ausgaben verwendet werden. Dazu muß z.B. lediglich die jeweilige Date ausgegeben werden, bevor mit dem ADW-Startimpuls zugleich ein Latch-Eingang eines 7-Segmentdecoders des Moduls ANZEIG aus Heft 23 bedient wird. Ein weiterer Vorteil ist, daß als gemeinsamer Eingang der Pin CA1 oder CB1 der Anwender-VIA 6522 benutzt werden kann. Die Pins CA2 und CB2 sind als Ausgänge wertvoller. Als letzte Änderung ist die Arbeitszeit je Meßstelle konstant 65 ms und nicht mehr direkt abhängig vom jeweiligen Meßwert.



Veränderte Anschaltung 555-Timer an VIA bei Modul (ADW)

Schaltung von 555-Timer siehe MICRO MAG, Heft 2o S.4



Aufruf (ADW)

1 - 8 Messwerte möglich

Volle Zeit

Startpuls ausgeben an 555-Bit

Flußdiagramm (ADW)
Analog/Digital-Wandler

Antwort-Puls an UCB1

UT1-High

T1 läuft rückwärts

Volle Zeit abwarten

```
0000        0001 ;(PPR-TEST)PULS-PROPORTIONAL REGLER TEST/12.6.82/KOK
0000        0002 PRG    =$0200          ;### PROGRAMM BEGINN
0000        0003 VAR    =$B0            ;### VARAIBALEN BEGINN
0000        0004
0000        0005 ;-----------------------
0000        0006 ;PAR 'A/D-WANDLER ADW.4'
0000        0007 ;-----------------------
0000        0008
0000        0009 ;FUER 1-8 A/D-WANDLER, EINZEL-CLOCK-OUT
0000        0010 ;MIT EXTERNEM 'XPOINT'-AUFRUF, KONSTANTZEIT 65 MS.
0000        0011
0000        0012 ;#### ADW.4 VARIABLEN & KONSTANTEN
0000        0013 ZYKLUS =8              ;FUER 1-8 MESSWERTE, HIER FUER 
0000        0014 ;      MUSS BEI DEN MASKEN BERUECKSICHTIGT WERDEN!
0000        0015
0000        0016 OFFMSK =%11111111      ;START PULS ENDE
0000        0017 ;      SOVIELE '1' VON RECHTS WIE 'ZYKLUS'
0000        0018
0000        0019 ADPORT =UDRB           ;USER-PORT B
0000        0020
```

```
0000          0021        * =VAR          ;VARIABLEN BEGINN
00B0          0022 HEXREG *=*+ZYKLUS ,EINGABE REGISTER
00B8          0023 XPOINT *=*+1           ;MESSTELLEN AUSWAHL REGISTER
00B9          0024 VARADW =*
00B9          0025
00B9          0026        * =PRO          ;PROGRAMM BEGINN
0200          0027 ;----- INITIALISIERUNG NO-INTERUPT
0200          0028 ;       (USER-VIA 6522 AIM.65)
0200 A97F     0029 INIADW LDA #%01111111  ;DISABLE
0202 8D0EA0   0030        STA UIER        ;INTERUPT
0205          0031
0205          0032 ;----- PORT B <ZYKLUS> X OUT SETZEN
0205 A9FF     0033        LDA #0FFMSK     ;OUT
0207 8D02A0   0034        STA ADPORT+2    ;DAT.DIR.REG.B
020A          0035
020A          0036 ;----- INIT UCB1 ALS LO-PULS-EINGANG
020A AD0CA0   0037        LDA UPCR        ;HOLE OLD
020D 29EF     0038        AND #%11101111  ;BIT 0=0: UCB1 LOACTIVE
020F 8D0CA0   0039        STA UPCR
0212          0040
0212          0041 ;----- INIT TIMER 1
0212 AD0BA0   0042        LDA UACR        ;HOLE OLD
0215 293F     0043        AND #%00111111  ;ONE SHUT MOD, BIT 6+7=0
0217 8D0BA0   0044        STA UACR
021A          0045
021A          0046
021A          0047 ;++++++++++ 'PROGRAMM ADW.4'
021A          0048
021A A6B8     0049 ADW    LDX XPOINT      ;WELCHE MESSTELLE IST DRAN?
021C A0FF     0050        LDY #$FF        ;VOLLE ZEIT..
021E 8C04A0   0051        STY UT1L        ;..LADE T1-LO
0221 AD00A0   0052        LDA ADPORT      ;IN PORT B..
0224 3D4B02   0053        AND SETMSK,X    ;..START-BIT WIRD 0
0227 8D00A0   0054        STA ADPORT      ;STARTE AUSGEWAEHLTEN 555
022A 8C05A0   0055        STY UT1CH       ;LADE UND STARTE T1
022D 09FF     0056        ORA #0FFMSK     ;ALLE BIT WIEDER 1
022F 8D00A0   0057        STA ADPORT      ;ENDE START PULS 555
0232 A910     0058        LDA #%00010000  ;UCB1-INT.MASKE
0234 2C0DA0   0059 AD1    BIT UIFR        ;ERROR FALLS T1 '00'
0237 5003     0060        BVC AD2         ;NEIN
0239 4C72FF   0061        JMP ERROR       ;PROGRAMM ABBRUCH
023C F0F6     0062 AD2    BEQ AD1         ;ZEIT LAEUFT NOCH
023E          0063
023E          0064 ;----- AUSGEWAEHLTER 555 IST ABGELAUFEN
023E          0065 ;      ERGEBNIS LIEGT NUN IM TIMER T1
023E AD05A0   0066        LDA UT1CH       ;NUR HI
0241 49FF     0067        EOR #$FF        ;INVERTIERE DATE
0243 95B0     0068        STA HEXREG,X    ;AUFFANG-REG
0245          0069
0245          0070 ;----- TIMER IST AUCH TAKTGEBER, VOLLE ZEIT ABWARTEN
0245 2C0DA0   0071        BIT UIFR        ;WARTE AUF T1-INTERUPT BIT
0248 50FB     0072        BVC *-3         ;LAEUFT NOCH
024A 60       0073        RTS
024B          0074
024B          0075 ;----- MASKEN FUER DEN 555--START
024B          0076 ;      SOVIELE MASKEN WIE <ZYKLUS>
024B FE       0077 SETMSK .BYT %11111110  ;BIT 0
024C FD       0078        .BYT %11111101  ;BIT 1
```

```
0240 FB     0079        .BYT %11111011,%11110111,%11101111
024E F7     0079
024F EF     0079
0250 DF     0080        .BYT %11011111,%10111111,%01111111
0251 BF     0080
0252 7F     0080
0253        0081 PRDADW =*
0253        0082
0253        0083 ERROR  =$FF72          ;PRINT AIM
0253        0084
0253        0085 UDRB   =$A000          ;DATA REG B
0253        0086 UT1L   =$A004          ;TIMER 1,LOW
0253        0087 UT1CH  =$A005          ;TIMER 1,HIGH
0253        0088 UACR   =$A00B          ;AUX CONTR REG
0253        0089 UPCR   =$A00C          ;PER CONTR REG
0253        0090 UIFR   =$A00D          ;INT FLAG  REG
0253        0091 UIER   =$A00E          ;INT ENABL REG
```

Diese Serie wird fortgesetzt mit dem (PPR), Puls-Proportional-Regler

Michael Beland, 2206 Sparrienshoop

AIM User Keyboard

Das nachfolgende Programm dekodiert ab UKEYB die Tastatur des AIM 65 vollständig, d.h. es wird Code von hex 01 bis 7F und von A0 bis FF erzeugt. Die Belegung der Tastatur mit Sonderzeichen ist dabei wie folgt:

TASTE	INTERNAT. CODE	INTERNAT. REF.VERSION	DEUTSCHE CODE	DEUTSCHE REF.VERSION
CTRL ,	$1C			
CTRL /	$1F			
^L @	$40	@	$C0	§
^ F1	$5B	[	$DB	Ä
^L 0	$5C	\	$DC	Ö
^ F2	$5D	]	$DD	Ü
^ F3	$5E	^	$DE	^
^R DEL	$5F	_	$DF	_
^R @'	$60	`	$E0	`
F1'	$7B	{	$FB	ä
R 0'	$7C	|	$FC	ö
F2'	$7D	}	$FD	ü
F3'	$7E	~	$FE	ß

Tasten mit ' nur in der Betriebsart UNLOCK

Es lassen sich mit den Tasten CTRL 0 bis CTRL 9 zehn Flags von hex 00 auf 80 und umgekehrt umschalten. Dabei erscheint auf dem Display die Mitteilung XON (XOFF), wobei X die Flagnummer ist. In diesem Programm sind belegt:

CTRL 1 'SHIFTLOCK/UNLOCK (00/80)
CTRL 2 das MSB des erzeugten Zeichencodes ist '0' bzw. '1'
CTRL 3 Display an/aus
CTRL 4 Bildschirm an/aus
CTRL 5 2. Drucker an/aus
CTRL 6 bis CTRL 0 sind unbenutzt

Der Programmteil TVDP leitet die über DILINK kommenden Zeichen weiter an einen 2. Drucker, Bildschirm oder Display. Die Ausgabeeinheiten sind separat mittels des CTRL-X-Tasten ein- und ausschaltbar. Auch wenn das Display ausgeschaltet ist, wird der Displaybuffer bedient (wichtig für Monitorbefehle).

Die Ausgabe des DELETE-Zeichens ist beim AIM 65 auf das Display zugeschnitten. Kommt von der Tastatur ein DELETE-Zeichen, so wird über das DILINK innerhalb der ersten 20 Spalten ein Space gesendet. Stehen mehr als 20 Zeichen im Display, so geschieht nichts. Das Programm RDRUB gibt im Editormode Delete-Zeichen richtig an Bildschirm und zweiten Drucker weiter. Die gleiche Aufgabe erfüllt das Programm BASRUB, wenn man in BASIC arbeitet. Größte Zeilenlänge in BASIC ist 71 Zeichen, bei den ersten 60 Zeichen ist Delete möglich. Unter FORTH auf dem AIM 65 wird die korrekte Delete-Weitergabe durch das Programm FORUB übernommen.

Im Editor wird das User-Keyboard mit R IN=U/Spacetaste, in BASIC mit LOAD IN=U/Spacetaste eingeschaltet. In FORTH wird der in der USER-AREA lokalisierte Vektor UKEY (in hex 22C/22D) auf FORUB gesetzt (z.B. in FORTH mit nnnn UKEY ! , wobei nnnn die Adresse von FORUB ist).

Da der UIN-Vektor durch das Programm benutzt wird, wurde ein neuer UIN-Vektor unter dem Namen UIN3 geschaffen. Die Eingabe läuft über UIN3, wenn anstelle der Spacetaste 'U' getippt wird.

```
0000          0000 ;.AIM65 - USER KEYBOARD
0000          0001 ;.**********************
0000          0002
0000          0003 ;.VERSION 4.2
0000          0004 ;.COPYRIGHT BY
0000          0005 ;.MICHAEL BELAND
0000          0006 ;.WALDSTR. 41
0000          0007 ;.2206 KL. OFFENSETH-SPARRIESHOOP
0000          0008 ;.27.08.1982
0000          0009
0000          0010 VARIAB =$2320
0000          0011 CODE   =$2336
0000          0012 PRINT  =$2F4D          ;START OF PRINTER-SUBROUT
0000          0013 TV     =$2636          ;START OF SCREEN-SUBROUT
0000          0014
0000          0015        *=VARIAB
2320          0016 ;;10 KEYBOARD ALTERABLE FLAGS
2320          0017 FLAG0  *=*+1           ;<CTRL 0>,NOT USED
2321          0018 SLOFLG *=*+1           ;<CTRL 1>,SHIFTLOCK/UNL<00/80>
2322          0019 KBSET  *=*+1           ;<CTRL 2>,ASCII/GERMAN KEYB
2323          0020 DISFLG *=*+1           ;<CTRL 3>,DISPLAY ON/OFF
2324          0021 TVFLG  *=*+1           ;<CTRL 4>,TV ON/OFF
2325          0022 PR2FLG *=*+1           ;<CTRL 5>,2ND PRINTER ON/OFF
2326          0023 FLAG6  *=*+1           ;<CTRL 6>,NOT USED
2327          0024 FLAG7  *=*+1           ;<CTRL 7>,NOT USED
2328          0025 FLAG8  *=*+1           ;<CTRL 8>,NOT USED
2329          0026 FLAG9  *=*+1           ;<CTRL 9>,NOT USED
232A          0027 COL    *=*+1           ;CHARACTERS PER LINE
232B          0028 SAVX   *=*+1           ;SAVE X
232C          0029 DELFLG *=*+1           ;IF DEL OR BS-KEY 80 ELSE 00
232D          0030 DELNUM *=*+1           ;IF >=0 AND DEL/BS; 2 DEL TO TV
232E          0031
232E          0032 PHXY   =$EB9E
232E          0033 PLXY   =$EBAC
232E          0034 GETKEY =$EC40
232E          0035 OUTDIS =$EF05
```

```
232E           0036 OUTDD1 =$EF7B
232E           0037 REDOUT =$E973
232E           0038 RED2   =$E976
232E           0039 PSLS   =$E7DC
232E           0040 PSL1   =$E837          ;PRINT "/"
232E           0041 TOGMES =$E021          ;ON OFF
232E           0042 DILINK =$A406          ;VEKTOR TO TVDP
232E           0043 CURPO2 =$A415
232E           0044 DIBUF  =$A438
232E           0045 REA1   =$E956
232E           0046 UIN1   =$108           ;OLD UIN (POINTS TO USRIN)
232E           0047 UIN2   =$104           ;INTERNAL VECTOR
232E           0048 UIN3   =$100           ;NEW UIN
232E           0049
232E           0050        *=CODE
2336           0051 ;.**  DILINK BRANCHING  **
2336 8E2B23    0052 TVDP   STX SAVX
2339 2C2323    0053        BIT DISFLG      ;DISPLAY ON ?
233C 1017      0054        BPL TVDP1       ;YES
233E C90D      0055        CMP #$0D        ;NO,DON'T DISPLAY CHARACTER
2340 F013      0056        BEQ TVDP1       ;BUT IF NOT CR
2342 C98D      0057        CMP #$8D
2344 F00F      0058        BEQ TVDP1
2346 AE15A4    0059        LDX CURPO2      ;PUT IT INTO DISPLAY-BUFFER
2349 E03C      0060        CPX #60
234B 1003      0061        BPL *+5
234D 9D38A4    0062        STA DIBUF,X
2350 EE15A4    0063        INC CURPO2
2353 D003      0064        BNE TVDP2
2355 2005EF    0065 TVDP1  JSR OUTDIS      ;DISPLAY CHAR.
2358 2C2423    0066 TVDP2  BIT TVFLG       ;TV ON ?
235B 3003      0067        BMI TVDP3
235D 203626    0068        JSR TV
2360 2C2523    0069 TVDP3  BIT PR2FLG      ;2ND PRINTER ON ?
2363 3003      0070        BMI TVDP4
2365 204D2F    0071        JSR PRINT
2368 C90D      0072 TVDP4  CMP #$0D        ;COUNT NO. OF CHR'S
236A F004      0073        BEQ TVDP5       ;OF THE LINE
236C C98D      0074        CMP #$8D        ;IF 0D OR 8D
236E D005      0075        BNE TVDP6
2370 A200      0076 TVDP5  LDX #0          ;RESET COL
2372 8E2A23    0077        STX COL
2375 EE2A23    0078 TVDP6  INC COL
2378 AE2B23    0079        LDX SAVX
237B 60        0080        RTS
237C           0081
237C           0082 ;.**  UIN BRANCHING  **
237C 9003      0083 USRIN  BCC *+5
237E 6C0401    0084 USRIN1 JMP (UIN2)
2381 2037E8    0085        JSR PSL1        ;PRINT "/"
2384 2073E9    0086        JSR REDOUT
2387 C955      0087        CMP #'U'        ;USER ?
2389 F01E      0088        BEQ USRIN2
238B EA        0089        NOP             ;OTHER INPUT DEVICE ?
238C EA        0090        NOP
238D EA        0091        NOP
238E EA        0092        NOP
```

```
238F BA       0093          TSX               ;EDITOR-BASIC KEYB BRANCHING
2390 BD0201   0094          LDA $102,X
2393 C9F7     0095          CMP #$F7          ;EDITOR ?
2395 D006     0096          BNE BASKB
2397 A05F     0097 EDIKB    LDY #<RDRUB
2399 A224     0098          LDX #>RDRUB
239B D012     0099          BNE USRIN3
239D A0DA     0100 BASKB    LDY #<BASRUB
239F A224     0101          LDX #>BASRUB
23A1 D00C     0102          BNE USRIN3
23A3 EA       0103          NOP               ;SET VEKTOR
23A4 EA       0104          NOP               ;TO OTHER INPUT DEVICE
23A5 EA       0105          NOP
23A6 EA       0106          NOP
23A7 EA       0107          NOP
23A8 EA       0108          NOP
23A9 AC0001   0109 USRIN2   LDY UIN3          ;USER
23AC AE0101   0110          LDX UIN3+1
23AF 8C0401   0111 USRIN3   STY UIN2
23B2 8E0501   0112          STX UIN2+1
23B5 18       0113          CLC
23B6 90C6     0114          BCC USRIN1
23B8          0115
23B8          0116 ;.**  USER KEYBOARD  **
23B8          0117 ;.RECOGNIZES SMALL/BIG LETTERS
23B8          0118 ;.SHIFTL 0->$5C,CTRL ,->$1C,CTRL /->1F
23B8          0119 ;.SHIFTR 0->$7C,SHIFTR DEL->$5F,SHIFTR @->$60
23B8          0120 ;.10 FLAGS: CTRL 0 ... CTRL 9
23B8          0121 ;.SHIFTLOCK/UNLOCK<CTRL 1>
23B8          0122 ;.ASCII/GERMAN CODE<CTRL 2>
23B8          0123 ;.X,Y ARE NOT CHANGED
23B8          0124
23B8 B00B     0125 UKEYB    BCS UKB
23BA A900     0126          LDA #0            ;KB-INITIALISATION
23BC 8D2223   0127          STA KBSET         ;ASCII
23BF 8D2123   0128          STA SLOFLG        ;SHIFTLOCK
23C2 A90D     0129          LDA #$0D
23C4 60       0130          RTS
23C5 209EEB   0131 UKB      JSR PHXY
23C8 2040EC   0132          JSR GETKEY        ;GET CHR
23CB C97F     0133          CMP #$7F          ;DEL ?
23CD F04F     0134          BEQ DELCHR
23CF E000     0135          CPX #0            ;CTRL?,SHIFTL?,SHIFTR?
23D1 F02E     0136          BEQ ESCAPE
23D3 E010     0137          CPX #$10          ;CTRL ?
23D5 D012     0138          BNE SHIFTL
23D7 C930     0139          CMP #'0'          ;CTRL 0-9 ?
23D9 3004     0140          BMI *+6
23DB C93A     0141          CMP #$3A
23DD 3047     0142          BMI GOTOG
23DF C92C     0143          CMP #','
23E1 F004     0144          BEQ *+6
23E3 C92F     0145          CMP #'/'
23E5 D02C     0146          BNE UKB1
23E7 4930     0147          EOR #$30
23E9 E020     0148 SHIFTL   CPX #$20          ;SHIFTL ?
23EB D006     0149          BNE SHIFTR
```

```
23ED C930     0150          CMP #'0'
23EF D022     0151          BNE UKB1
23F1 A95C     0152          LDA ##5C
23F3 E040     0153 SHIFTR   CPX ##40          ;SHIFTR ?
23F5 D01C     0154          BNE UKB1
23F7 C940     0155          CMP #'@'
23F9 F00D     0156          BEQ UKB00
23FB C930     0157          CMP #'0'
23FD D014     0158          BNE UKB1
23FF A95C     0159          LDA ##5C
2401 C91B     0160 ESCAPE   CMP ##1B
2403 D003     0161          BNE UKB00
2405 4C56E9   0162          JMP REA1          ;CLR BUFFERS,COMIN
2408 C940     0163 UKB00    CMP ##40          ;ALPHA CHR ?
240A 9007     0164          BCC UKB1
240C AE2123   0165          LDX SLOFLG        ;SHIFTLOCK ?
240F F002     0166          BEQ UKB1
2411 0920     0167          ORA ##20          ;NO,SMALL CHR'S
2413 C920     0168 UKB1     CMP ##20          ;CTRL CHR ?
2415 9003     0169          BCC UKBRTN
2417 4D2223   0170          EOR KBSET         ;NO,ASCII/GERMAN
241A 20ACEB   0171 UKBRTN   JSR PLXY
241D 60       0172          RTS
241E E040     0173 DELCHR   CPX ##40          ;SHIFTR ?
2420 D0F8     0174          BNE UKBRTN
2422 A95F     0175          LDA ##5F
2424 D0ED     0176          BNE UKB1
2426 202C24   0177 GOTOG    JSR TOG
2429 4CC823   0178          JMP UKB+3
242C 48       0179 TOG      PHA               ;CHANGE FLAG0,ACCU
242D 290F     0180          AND ##0F          ;CONVERT TO HEX-NUMBER
242F AA       0181          TAX
2430 BD2023   0182          LDA FLAG0,X
2433 4980     0183          EOR ##80
2435 9D2023   0184          STA FLAG0,X
2438 08       0185          PHP
2439 A203     0186          LDX #3
243B A002     0187          LDY #2            ;'ON' <FLAG=00>
243D 28       0188          PLP
243E 1002     0189          BPL TOG1
2440 A005     0190          LDY #5            ;'OFF' <FLAG=80>
2442 B921E0   0191 TOG1     LDA TOGMES,Y
2445 204D24   0192          JSR ODD1          ;DISPLAY ONLY
2448 88       0193          DEY
2449 CA       0194          DEX
244A D0F6     0195          BNE TOG1
244C 68       0196          PLA               ;DISPLAY FLAG-NUMBER
244D 8E2B23   0197 ODD1     STX SAVX
2450 0980     0198          ORA ##80
2452 207BEF   0199          JSR OUTDD1
2455 AE2B23   0200          LDX SAVX
2458 60       0201          RTS
2459          0202
2459          0203 ;.EQUIVALENT TO RDRUB(AIM65-MONITOR)
2459          0204 ;.READ WITH RUBOUT OR DELETE POSSIBLE
2459 20F224   0205 RB2      JSR BASRB1        ;A<-$7F,DELFLG<-$80
245C 20DCE7   0206          JSR PSLS
245F 207724   0207 RDRUB    JSR CUREAD
2462 F00D     0208          BEQ RDR1          ;BRANCH IF DEL OR BS
```

```
2464 48      0209         PHA
2465 AD15A4  0210         LDA CURPO2
2468 C93C    0211         CMP #60
246A 68      0212         PLA
246B B003    0213         BCS RDRTS
246D 4C76E9  0214         JMP RED2
2470 60      0215 RDRTS   RTS
2471 88      0216 RDR1    DEY
2472 10E5    0217         BPL RB2
2474 C8      0218         INY
2475 F0E8    0219         BEQ RDRUB
2477         0220
2477 B00F    0221 CUREAD  BCS CUR2          ;GET CHR FROM KEYBOARD
2479 A900    0222         LDA #0
247B 8D2C23  0223         STA DELFLG
247E 20B823  0224 CUR1    JSR UKEYB
2481 C97F    0225 DELBS   CMP #$7F          ;IF DEL OR BS RETURN WITH Z=1
2483 F002    0226         BEQ CURRTN
2485 C908    0227         CMP #$08
2487 60      0228 CURRTN  RTS
2488 8A      0229 CUR2    TXA               ;DISPLAY CURSOR /DEL/BS FOR TV
2489 48      0230         PHA               ;SAVE X
248A 2C2323  0231         BIT DISFLG        ;DISPLAY ON ?
248D 300C    0232         BMI CUR3
248F AE15A4  0233         LDX CURPO2        ;YES
2492 E014    0234         CPX #20           ;CURSOR ?
2494 B005    0235         BCS CUR3
2496 A9DE    0236         LDA #$DE          ;YES
2498 207BEF  0237         JSR OUTDD1
249B         0238 ;.HANDLE DEL OR BS FOR TV & 2ND PRINTER
249B 2C2C23  0239 CUR3    BIT DELFLG        ;DEL? BS?
249E 101B    0240         BPL CUR5
24A0 A900    0241         LDA #0            ;YES
24A2 8D2C23  0242         STA DELFLG        ;CLR
24A5 AE2A23  0243         LDX COL           ;0,1 OR 2 DEL ?
24A8 E001    0244         CPX #1
24AA F00F    0245         BEQ CUR5
24AC E017    0246         CPX #23
24AE 1008    0247         BPL CUR4
24B0 2C2D23  0248         BIT DELNUM
24B3 3003    0249         BMI CUR4
24B5 20C824  0250         JSR DEL           ;2 DEL
24B8 20C824  0251 CUR4    JSR DEL           ;ONLY 1 DEL
24BB AD2A23  0252 CUR5    LDA COL           ;MORE THAN 21 CHR'S ?
24BE C916    0253         CMP #22           ;SET OR RESET CARRY-FLAG
24C0 6E2D23  0254         ROR DELNUM        ;STORE IT IN MSB OF DELNUM
24C3 68      0255         PLA
24C4 AA      0256         TAX
24C5 38      0257         SEC
24C6 B0B6    0258         BCS CUR1
24C8 AD2A23  0259 DEL     LDA COL           ;DEL POSSIBLE TO START OF LINE
24CB C901    0260         CMP #$01
24CD F0B8    0261         BEQ CURRTN
24CF CE2A23  0262         DEC COL
24D2 CE2A23  0263         DEC COL
24D5 A97F    0264         LDA #$7F
24D7 4C5823  0265         JMP TVDP2         ;TV & PRINTER
24DA         0266
```

```
24DA            0267 ;.SUBROUTINE LIKE RDRUB BUT MODIFIED FOR BASIC
24DA            0268 ;.ROWLENGTH: 71 CHR'S
24DA            0269 ;.DELETE POSSIBLE TO FIRST 60 CHR'S ONLY
24DA 900A       0270 BASRUB BCC BASRB
24DC A948       0271        LDA #72          ;IF MORE THAN 71 CHR'S
24DE CD2A23     0272        CMP COL
24E1 B003       0273        BCS BASRB
24E3 A90D       0274        LDA #$0D         ;THEN RETURN
24E5 60         0275        RTS
24E6 207724     0276 BASRB  JSR CUREAD
24E9 D0FA       0277        BNE *-4          ;IF NOT DEL OR BS THEN RTN
24EB AD2A23     0278        LDA COL          ;IF DEL OR BS  AND COL<62
24EE C93E       0279        CMP #62          ;THEN DEL OR BS ARE ALLOWED
24F0 B0F4       0280        BCS BASRB        ;ELSE GET NEW KEY
24F2 A980       0281 BASRB1 LDA #$80
24F4 8D2C23     0282        STA DELFLG
24F7 A97F       0283        LDA #$7F
24F9 60         0284        RTS
24FA            0285
24FA            0286 ;.FORTH IS ABLE TO WORK WITH USER-KEYBORD
24FA            0287 ;.IF THE VEKTOR UKEY ($22C,$22D) POINTS TO FORUB
24FA 208824     0288 FORUB  JSR CUR2
24FD F0F3       0289        BEQ BASRB1
24FF 60         0290        RTS
2500            0291
2500            0292 T      =$00
2500            0293 F      =$80
2500            0294        *=DILINK
A406 3623       0295        .WOR TVDP
A408            0296        *=UIN1
0108 7C23       0297        .WOR USRIN
010A            0298        *=VARIAB
2320 80         0299        .BYT F,T,T,T,T,F,F,F,F,F ;CTRL 0 ... CTRL 9
2321 00         0299
2322 00         0299
2323 00         0299
2324 00         0299
2325 80         0299
2326 80         0299
2327 80         0299
2328 80         0299
2329 80.        0299
232A            0300        .END                                    #
```

Bücher

Rolle, G.: Personal Computer Lexikon, *Verlag Markt & Technik, Haar 1982, ISBN 3-922120-18-0, 136 Seiten, DM 19,50 (kart.). Das Buch hat den Untertitel 'die 1000 wichtigsten Hard- und Softwarebegriffe des Personal Computing mit ausführlicher Erklärung . Die Begriffe werden in Deutsch erklärt, zusätzlich wird die englische Übersetzung des deutschen Suchbegriffes angegeben. Um das Studium englischsprachiger Dokumentationen zu erleichtern, befindet sich im Anhang ein Register englisch-deutsch mit Seitenangabe. - Die Herausgabe eines Lexikons für ein sich schnell entwickelndes Fachgebiet ist immer problematisch, es kann immer nur eine Momentaufnahme des Sprachschatzes liefern und muß auch einen Kompromiß zwischen Verständlichkeit und Ausführlichkeit für die beiden großen Benutzerkreise der privat und der beruflich befaßten Personen schließen. Das ist hier recht gut gelungen, insbesondere für Leser mit geringen sprachlichen Vorkenntnissen.*

Jochen Richter, Moskau

AIM mit Floppy CBM 8050

Gestützt auf Beiträge des Herausgebers in 1) und 2) wurde ein Programm zum Anbinden der CBM-Floppy 8050 an den AIM 65/PC 100 entwickelt, das dem Benutzer alle komfortablen Möglichkeiten der 8050 an die Hand und an das Display gibt. Es wird beim Verfasser seit mehreren Monaten betrieben, hier wird es zusammen mit anderen Dienstleistungsprogrammen aus einer zweiten Monitorebene angesprungen.

Das Programm stellt drei verschiedene Dienstleistungen zur Verfügung (hier über die Funktionstasten F1 bis F3 aufrufbar):

F1 richtet die die In- und Outputvektoren auf das Programm. Dumps und Loads, sowie aus der Editorebene List (L) und Read (R), werden mit der Floppy ausgeführt. Hier werden Dumps automatisch als PRG-Dateien, Editor-Lists als SEQ-Dateien aufgezeichnet. Wie bei Kassettenbetrieb können Teile des Textes gelistet und an eine definierte Stelle eingelesen werden.

F2 ruft das Directory auf und listet es auf dem Drucker oder auf einem angeschlossenen Bildschirm.

F3 öffnet den Kommandokanal und gestattet den Aufruf der Dienstleistungen der Floppy-Intelligenz, wie Löschen, Formatieren, Kopieren, Umbenennen etc.. Die Kommandos werden jedoch ohne Anführungszeichen eingegeben. Alle Kommandooperationen werden nach der Ausführung quittiert. Die Syntax und ihre Wirkung sind unter 3) ausführlich beschrieben.

Hardwareseitig wurde der in 1) beschriebene am User-VIA simulierte IEEE488-Bus ohne Änderung übernommen. Lediglich auf die Bustreiber mußte verzichtet werden, da beim Verfasser kurzfristig nicht erhältlich. Mittlerweile hängen am 'nackten' Bus ein Drucker CBM 3022, ein Drucker MX-80 und die Floppy parallel, und auch nach mehr als 6-monatiger Betriebszeit konnte noch kein Lesefehler oder Belastungsdefekt festgestellt werden. (Nachahmung ohne Gewähr!)

Die Unterprogramme zur Bussimulation wurden nahezu unverändert aus 2) übernommen, sie sind dort zusammen mit den Funktionen des IEEE-Busses in einleuchtender Form dargestellt. So konnte auf ihre Kommentierung im vorliegenden Programm verzichtet werden. In 2) wird vom Herausgeber in seinem Programm ein direkter Speicherabzug als Dump an die Floppy geleitet. Im vorliegenden Programm wird davon abgesehen und der normale AIM 65-Dump benützt. Einmal, um wenigstens eine theoretische Formatkompatibilität zu erhalten, hauptsächlich aber, um die komfortable Möglichkeit des Zusammenbindens mehrerer Dumps in einem Programm (MORE?) nicht zu verlieren. Das umständliche Blockformat verlängert zwar die Aufzeichnung, der Zeitverlust liegt jedoch eher im Zehntelsekundenbereich und Speicherplatz stellt die 8050 ausreichend zur Verfügung.

Die Programmzeilen 298/299 dienen dazu, die Fehlermeldungen auf einem beim Verfasser als Sichtgerät betriebenen alten 'PET' invertiert darzustellen. Sie müssen bei anderen Video-Displays entfallen.

Literaturhinweise:

1) 65xx MICRO MAG, Heft Nr. 19, Juni 1981
2) 65xx MICRO MAG, Heft Nr. 21, Oktober 1981
3) Commodore CBM 8050 Bedienungshandbuch.

Hantierungsbeispiele

1. Ein Objectprogramm soll unter dem Namen 'DUMP1' gespeichert werden:
   Nach F1 wird nach 'D' mit 'FROM' und 'TO' der zu dumpende Bereich beschrieben. Nach 'OUT=U' meldet sich das Programm mit dem Prompt 'WRITE:'. Man gibt ein: 0:DUMP1. Nach 'RETURN' wird die PRG-Datei 'DUMP1' auf das Laufwerk 0 aufgezeichnet.

2. Der Source-Text 'LISTE9' soll in den Editor eingelesen werden:
   Nach F1 wird der Editor initialisiert. Nach 'IN=U' kommt der Prompt: 'READ:'.
   Eingabe: 1:LISTE9,S. Die SEQ-Datei mit dem Namen 'LISTE9' wird vom Laufwerk 1
   in den Editor gelesen.
3. Das Inhaltsverzeichnis der Diskette im Laufwerk 0 soll gelistet werden:
   Nach F2 antwortet man auf den Prompt: 'DIRECTORY:' mit $0.
4. Auf der Diskette im Laufwerk Nr. 1 soll die Datei mit dem Namen 'JUNK17' gelöscht werden:
   Nach F3 auf den Prompt: 'KDO:' antworten mit: S1:JUNK17 . Die Floppy quittiert mit
   '01,FILES SCRATCHED 01,00,00' (falls die Datei vorhanden war).

Die Syntax aller Kommandos ist im Anwenderhandbuch ausführlich beschrieben. Sie ist die gleiche wie unter BASIC-3 des Commodore.

```
0000            0000 ; ************************
0000            0001 ; *  AIM TO FLOPPY 8050  *
0000            0002 ; ************************
0000            0003 ;                          J.R. 1982
0000            0004 ;
0000            0005 ;
0000            0006 START  =$C000
0000            0007 VIA    =$A000
0000            0008        *=VIA
A000            0009 PORTB  =*
A000            0010 PORTA  =*+1
A000            0011 DDRB   =*+2
A000            0012 DDRA   =*+3
A000            0013 ACR    =*+11
A000            0014 PCR    =ACR+1
A000            0015 WRITE  =$E0
A000            0016 READ   =$C0
A000            0017 ATNHIG =$0E
A000            0018 ATNLOW =$0C
A000            0019 UNLISN =$3F
A000            0020 UNTALK =$5F
A000            0021 CR     =$D
A000            0022 LL     =$E8FE
A000            0023 REENTR =$F6CF
A000            0024 MON    =$E956
A000            0025 INFLAG =$A412
A000            0026 OUTFLG =$A413
A000            0027 STIY   =$A429
A000            0028 OUTPRT =$E97A
A000            0029 RDRUB  =$E95F
A000            0030 CRLF   =$E9F0
A000            0031 BYTOUT =$EA46
A000            0032        *=$1D
001D            0033 JUMPER *=*+2
001F            0034 WHO    *=*+1
0020            0035 STRING *=*+$1F
003F            0036 FLG    *=*+1
0040            0037 ;
0040            0038        *=$10C
010C 4C00C0     0039        JMP INITI       ;READ OR WRITE
010F 4C17C0     0040        JMP DIRC        ;DISPL. DIRECTORY
0112 4C1BC0     0041        JMP KOMMDO      ;TO COMMAND CHANNEL
0115            0042 ;
```

```
0115            0043          *=START
C000 A91F       0044 INITI    LDA #<USOINI
C002 8D0A01     0045          STA $10A
C005 A9C0       0046          LDA #>USOINI
C007 8D0B01     0047          STA $10B
C00A A927       0048          LDA #<USIINI
C00C 8D0801     0049          STA $108
C00F A9C0       0050          LDA #>USIINI
C011 8D0901     0051          STA $109
C014 4C56E9     0052          JMP MON
C017 A00F       0053 DIRC     LDY #15          ;DIRECTORY ENTRY
C019 D014       0054          BNE COMDIN
C01B A00B       0055 KOMMDO   LDY #11          ;COMMAND ENTRY
C01D D010       0056          BNE COMDIN
C01F B07C       0057 USOINI   BCS DURCH        ;COMING FROM OUTPUT
C021 A000       0058          LDY #0           ;IDENTIFY
C023 841F       0059          STY WHO
 F008     0060          BEQ COMDIN
C027 B074       0061 USIINI   BCS DURCH        ;COMING FROM INPUT
C029 A901       0062          LDA #1           ;IDENTIFY
C02B 851F       0063          STA WHO
C02D A006       0064          LDY #6
C02F 20FEE8     0065 COMDIN   JSR LL           ;RESTORE IN-/OUTFLG
C032 A200       0066          LDX #0           ;ENABLE DISPLAY
C034 20F0E9     0067          JSR CRLF
C037 B9B4C0     0068 MESOUT   LDA MES,Y        ;DISPLAY MESSAGE
C03A C8         0069          INY
C03B 207AE9     0070          JSR OUTPRT
C03E 10F7       0071          BPL MESOUT
C040 A000       0072          LDY #0
C042 205FE9     0073 KEYS     JSR RDRUB        ;GET COMMANDO STRING
C045 992100     0074          STA STRING+1,Y   ;STORE IT
C048 C8         0075          INY
C049 C90D       0076          CMP #CR
C04B D0F5       0077          BNE KEYS
C04D 207AE9     0078          JSR OUTPRT
C050 8420       0079          STY STRING       ;STORE STRING LENGTH
C052 A521       0080          LDA STRING+1
C054 C930       0081          CMP #$30         ;DISK #0 ?
C056 F00A       0082          BEQ KE1
C058 C931       0083          CMP #$31         ;OR #1 ?
C05A F006       0084          BEQ KE1          ;YES, CHECK SYNTAX
C05C C924       0085          CMP #'$ ;NO, CHECK IF DIRECTORY
C05E F051       0086          BEQ DIR1         ;IF SO GO THERE
C060 D04C       0087          BNE KOM1         ;NO, MUST BE COMMAND
C062 B92000     0088 KE1      LDA STRING,Y     ;CHECK SYNTAX OF STRING
C065 C92C       0089          CMP #',
C067 F009       0090          BEQ KE2
C069 C93A       0091          CMP #':
C06B F019       0092          BEQ PRG          ;MUST BE PROGRAM
C06D 88         0093          DEY
C06E D0F2       0094          BNE KE1
C070 F02E       0095          BEQ ERR          ;SYNTAX ERROR
C072 C8         0096 KE2      INY
C073 B92000     0097          LDA STRING,Y     ;CHECK IF PRG OR SEQ
C076 C950       0098          CMP #'P
C078 F00C       0099          BEQ PRG
C07A C953       0100          CMP #'S
```

```
C07C F004     0101          BEQ KE3
C07E C957     0102          CMP #'W
C080 D01E     0103          BNE ERR
C082 E61F     0104 KE3      INC WHO          ;=SEQ
C084 E61F     0105          INC WHO
C086 A51F     0106 PRG      LDA WHO          ;=PRG
C088 0A       0107          ASL A
C089 A8       0108          TAY
C08A B9DCC0   0109          LDA TAB,Y        ;GET JMP-ADDRESS
C08D 851D     0110          STA JUMPER
C08F C8       0111          INY
C090 B9DCC0   0112          LDA TAB,Y
C093 851E     0113          STA JUMPER+1
C095 A955     0114          LDA #'U ;RESTORE IN-/OUTFLAG
C097 8D12A4   0115          STA INFLAG
C09A 8D13A4   0116          STA OUTFLG
C09D 6C1D00   0117 DURCH    JMP (JUMPER)     ;JUMP
C0A0 A019     0118 ERR      LDY #25          ;DISPLAY 'SYNTAX ERROR'
C0A2 B9B4C0   0119 ERR1     LDA MES,Y
C0A5 C8       0120          INY
C0A6 207AE9   0121          JSR OUTPRT
C0A9 10F7     0122          BPL ERR1
C0AB 4C82E1   0123          JMP $E182
C0AE 4C8DC1   0124 KOM1     JMP KOMDO
C0B1 4C02C1   0125 DIR1     JMP DIREC
C0B4 5752     0126 MES      .BYTE 'WRITE',$BA
C0B9 BA       0126
C0BA 5245     0127          .BYTE 'READ',$BA
C0BE BA       0127
C0BF 4B444F   0128          .BYTE 'KDO',$BA
C0C2 BA       0128
C0C3 4449     0129          .BYTE 'DIRECTORY',$BA
C0CC BA       0129
C0CD 0D       0130          .BYTE $0D,' SYNTAX ERROR',$BF
C0CE 2053     0130
C0DB BF       0130
C0DC 9FC1     0131 TAB      .WORD WRIPRG,REDPRG,WRISEQ,REDSEQ
C0DE 20C2     0131
C0E0 ECC1     0131
C0E2 29C2     0131
C0E4 28       0132 OUTAB1 .BYTE $28,$F1,$28,$61,$28,$E1
C0E5 F1       0132
C0E6 28       0132
C0E7 61       0132
C0E8 28       0132
C0E9 E1       0132
C0EA 28       0133 OUTAB2 .BYTE $28,$F8,$28,$68,$28,$E8
C0EB F8       0133
C0EC 28       0133
C0ED 68       0133
C0EE 28       0133
C0EF E8       0133
C0F0 28       0134 INTAB1 .BYTE $28,$F0,$48,$60,$28,$E0
C0F1 F0       0134
C0F2 48       0134
C0F3 60       0134
C0F4 28       0134
C0F5 E0       0134
```

```
COF6 28       0135 INTAB2 .BYTE $28,$F8,$48,$68,$28,$E8
COF7 F8       0135
COF8 48       0135
COF9 68       0135
COFA 28       0135
COFB E8       0135
COFC 28       0136 COMTAB .BYTE $28,$6F
COFD 6F       0136
COFE 48       0137 ERRTAB .BYTE $48,$6F,$28,$EF
COFF 6F       0137
C100 28       0137
C101 EF       0137
C102 A20C     0138 DIREC  LDX #INTAB1-OUTAB1 ;READ & SHOW DIRECTOR\
C104 209EC2   0139        JSR OUTPUT
C107 202BC3   0140        JSR HEADER
C10A 2053C2   0141        JSR CHECK
C10D 20B4C2   0142        JSR INPUT
C110 A001     0143        LDY #1          ;SET HEADER FLAG
C112 8C29A4   0144        STY STIY
C115 2056C3   0145 DIR9   JSR ACC1        ;GET CHARACTER
C118 08       0146        PHP             ;SAVE STATUS
C119 2022C1   0147        JSR REDIRC
C11C 28       0148        PLP             ;CHECK IF LAST CHARACTER
C11D B0F6     0149        BCS DIR9        ;NO, NEXT ONE
C11F 4C3FC2   0150        JMP FINE        ;YES, GO HOME
C122 AC29A4   0151 REDIRC LDY STIY        ;CHECK HEADER FLAG
C125 F02E     0152        BEQ DIR2        ;=0: NO HEADER
C127 4E29A4   0153        LSR STIY        ;CLEAR FLAG
C12A A005     0154        LDY #5          ;PULL START BYTES
C12C 2056C3   0155 DIR8   JSR ACC1
C12F 88       0156        DEY
C130 D0FA     0157        BNE DIR8
C132 20F0E9   0158        JSR CRLF
C135 2056C3   0159 DIR3   JSR ACC1        ;1. CHARACTER
C138 48       0160        PHA
C139 297F     0161        AND #$7F        ;MAKE IT ASCII
C13B 207AE9   0162        JSR OUTPRT
C13E 68       0163        PLA
C13F D0F4     0164        BNE DIR3        ;'0' MEANS HEADER END
C141 20F0E9   0165        JSR CRLF
C144 A018     0166        LDY #24         ;UNDERLINE HEADER
C146 A92D     0167        LDA #'-'
C148 207AE9   0168 DIR4   JSR OUTPRT
C14B 88       0169        DEY
C14C D0FA     0170        BNE DIR4
C14E 20F0E9   0171        JSR CRLF
C151 20F0E9   0172        JSR CRLF
C154 60       0173        RTS
C155 C901     0174 DIR2   CMP #1           ;START BYTES ?
C157 D033     0175        BNE DIR99       ;NO, FORGET THEM
C159 2056C3   0176        JSR ACC1        ;PULL 2. START BYTE
C15C 2056C3   0177        JSR ACC1        ;PULL # OF TRACKS
C15F 48       0178        PHA             ;AND DISPLAY
C160 2056C3   0179        JSR ACC1        ;THEM VICEVERSA
C163 2046EA   0180        JSR BYTOUT      ;HI-BYTE/LO-BYTE
C166 68       0181        PLA
C167 2046EA   0182        JSR BYTOUT
C16A A920     0183        LDA #$20
C16C 207AE9   0184        JSR OUTPRT
```

```
C16F 2056C3  0185 DIR5    JSR ACC1          ;PULL NEXT
C172 F015    0186         BEQ DIR6          ;'0' MEANS LINE END
C174 C920    0187         CMP #$20          ;IF 'SPACE'
C176 F0F7    0188         BEQ DIR5          ;FORGET IT
C178 297F    0189         AND #$7F
C17A 207AE9  0190         JSR OUTPRT
C17D 2056C3  0191 DIR7    JSR ACC1
C180 297F    0192         AND #$7F
C182 F005    0193         BEQ DIR6
C184 207AE9  0194         JSR OUTPRT
C187 D0F4    0195         BNE DIR7
C189 20F0E9  0196 DIR6    JSR CRLF
C18C 60      0197 DIR99   RTS
C18D A218    0198 KOMDO   LDX #COMTAB-OUTAB1
C18F 209EC2  0199         JSR OUTPUT        ;OPEN CHANNEL 15
C192 202BC3  0200         JSR HEADER        ;OUTPUT COMMSTRING
C195 20B4C2  0201         JSR INPUT
C198 2056C3  0202         JSR ACC1          ;GET ANSWER
C19B 48      0203         PHA               ;DISPLAY IT
C19C 4C6BC2  0204         JMP CHE1          ;CLOSE SHOP
C19F B00C    0205 WRIPRG  BCS FETCH1
C1A1 209EC2  0206         JSR OUTPUT
C1A4 A200    0207         LDX #$0
C1A6 202BC3  0208         JSR HEADER        ;OPEN CHANNEL 1
C1A9 2053C2  0209         JSR CHECK         ;LOOK IF ERROR
C1AC 60      0210         RTS
C1AD 68      0211 FETCH1  PLA
C1AE C93B    0212         CMP #';'          ;CHECK IF 3 ZEROS
C1B0 D00A    0213         BNE K2            ;FOLLOW A ';'
C1B2 48      0214         PHA               ;MEANING END OF DUMP
C1B3 A920    0215         LDA #%00100000
C1B5 853F    0216         STA FLG
C1B7 68      0217 K1      PLA
C1B8 20C9C2  0218         JSR SEND
C1BB 60      0219         RTS
C1BC C930    0220 K2      CMP #$30
C1BE F007    0221         BEQ K3
C1C0 48      0222         PHA
C1C1 A900    0223         LDA #0
C1C3 853F    0224         STA FLG
C1C5 F0F0    0225         BEQ K1
C1C7 48      0226 K3      PHA
C1C8 063F    0227         ASL FLG
C1CA 243F    0228         BIT FLG
C1CC 10E9    0229         BPL K1
C1CE A008    0230         LDY #8
C1D0 20C9C2  0231 K4      JSR SEND
C1D3 88      0232         DEY
C1D4 D0FA    0233         BNE K4
C1D6 20FBC2  0234         JSR EOILO
C1D9 A90D    0235         LDA #$0D
C1DB 20C9C2  0236         JSR SEND
C1DE 201FC3  0237         JSR UNLIS
C1E1 A61F    0238         LDX WHO
C1E3 2004C3  0239         JSR PRIMAD
C1E6 201FC3  0240         JSR UNLIS
C1E9 4C56E9  0241         JMP MON
C1EC B00C    0242 WRISEQ  BCS FETCH
```

```
C1EE 209EC2   0243          JSR OUTPUT
C1F1 A206     0244          LDX #OUTAB2-OUTAB1
C1F3 202BC3   0245          JSR HEADER       ;OPEN CHANNEL 8
C1F6 2053C2   0246          JSR CHECK
C1F9 60       0247          RTS
C1FA 68       0248 FETCH    PLA
C1FB C90D     0249          CMP #CR          ;IF A 'CR' FOLLOWS
C1FD D004     0250          BNE FET1         ;A 'CR' THEN
C1FF C501     0251          CMP $01          ;END OF OUTPUT
C201 F006     0252          BEQ ENDE
C203 8501     0253 FET1     STA $01
C205 20C9C2   0254          JSR SEND
C208 60       0255          RTS
C209 20C9C2   0256 ENDE     JSR SEND         ;CLOSE SHOP
C20C 20FBC2   0257          JSR EOILO
C20F A501     0258          LDA $01
C211 20C9C2   0259          JSR SEND
C214 201FC3   0260          JSR UNLIS
C217 A61F     0261          LDX WHO
C219 2004C3   0262          JSR PRIMAD
C21C 201FC3   0263          JSR UNLIS
C21F 60       0264          RTS
C220 B018     0265 REDPRG   BCS ACC2
C222 209EC2   0266          JSR OUTPUT       ;OPEN CHANNEL 0 FOR READ-PRG
C225 A20C     0267          LDX #INTAB1-OUTAB1
C227 D007     0268          BNE READ1
C229 B00F     0269 REDSEQ   BCS ACC2
C22B 209EC2   0270          JSR OUTPUT       ;OPEN CHANNEL 8 FOR READ-SEQ
C22E A212     0271          LDX #INTAB2-OUTAB1
C230 202BC3   0272 READ1    JSR HEADER
C233 2053C2   0273          JSR CHECK        ;LOOK IF ERROR
C236 20B4C2   0274          JSR INPUT
C239 60       0275 BACK     RTS
C23A 2056C3   0276 ACC2     JSR ACC1
C23D B0FA     0277          BCS BACK
C23F 209EC2   0278 FINE     JSR OUTPUT
C242 2018C3   0279          JSR UNTAL
C245 A61F     0280          LDX WHO
C247 204DC3   0281          JSR PRIMHI
C24A 201FC3   0282          JSR UNLIS
C24D 20FEE8   0283          JSR LL
C250 4C56E9   0284          JMP MON
C253 201FC3   0285 CHECK    JSR UNLIS
C256 8A       0286          TXA
C257 48       0287          PHA
C258 A21A     0288          LDX #ERRTAB-OUTAB1
C25A 209EC2   0289          JSR OUTPUT       ;OPEN CHANNEL 15
C25D 204DC3   0290          JSR PRIMHI
C260 20B4C2   0291          JSR INPUT
C263 2056C3   0292          JSR ACC1
C266 C930     0293          CMP #$30         ;IF ERROR #0 THEN
C268 F026     0294          BEQ OK           ;OK
C26A 48       0295          PHA
C26B A90D     0296 CHE1     LDA #CR
C26D 207AE9   0297          JSR OUTPRT
C270 A912     0298          LDA #$12         ;ONLY TO INVERT DISPLAY
C272 207AE9   0299          JSR OUTPRT       ;ON CBM SCREENS
C275 68       0300          PLA
```

```
C276 207AE9  0301          JSR OUTPRT
C279 2056C3  0302 ERRMES   JSR ACC1        ;DISPLAY ERROR #
C27C 08      0303          PHP             ;AND MESSAGE
C27D 207AE9  0304          JSR OUTPRT
C280 28      0305          PLP
C281 B0F6    0306          BCS ERRMES
C283 209EC2  0307          JSR OUTPUT
C286 2004C3  0308          JSR PRIMAD
C289 201FC3  0309          JSR UNLIS
C28C 68      0310          PLA
C28D 4C82E1  0311          JMP $E182
C290 201FC3  0312 OK       JSR UNLIS
C293 68      0313          PLA
C294 AA      0314          TAX
C295 CA      0315          DEX
C296 CA      0316          DEX
C297 209EC2  0317          JSR OUTPUT
C29A 204DC3  0318          JSR PRIMHI
C29D 60      0319          RTS
C29E A90F    0320 OUTPUT   LDA #$0F
C2A0 8D02A0  0321          STA DDRB
C2A3 8D00A0  0322          STA PORTB
C2A6 A9FF    0323          LDA #$FF
C2A8 8D03A0  0324          STA DDRA
C2AB 8D01A0  0325          STA PORTA
C2AE A9E0    0326          LDA #WRITE
C2B0 8D0CA0  0327          STA PCR
C2B3 60      0328          RTS
C2B4 A90F    0329 INPUT    LDA #$0F
C2B6 8D02A0  0330          STA DDRB
C2B9 A90D    0331          LDA #$0D
C2BB 8D00A0  0332          STA PORTB
C2BE A900    0333          LDA #0
C2C0 8D03A0  0334          STA DDRA
C2C3 A9C0    0335          LDA #READ
C2C5 8D0CA0  0336          STA PCR
C2C8 60      0337          RTS
C2C9 49FF    0338 SEND     EOR #$FF
C2CB 8D01A0  0339          STA PORTA
C2CE AD00A0  0340          LDA PORTB
C2D1 2901    0341          AND #1
C2D3 0906    0342          ORA #$06
C2D5 8D00A0  0343          STA PORTB
C2D8 AD00A0  0344 SEND1    LDA PORTB
C2DB 2960    0345          AND #$60
C2DD C920    0346          CMP #$20
C2DF D0F7    0347          BNE SEND1
C2E1 A90F    0348          LDA #$0F
C2E3 8D00A0  0349          STA PORTB
C2E6 A9FF    0350          LDA #$FF
C2E8 8D01A0  0351          STA PORTA
C2EB AD00A0  0352 SAMPLE   LDA PORTB
C2EE 2960    0353          AND #$60
C2F0 C940    0354          CMP #$40
C2F2 D0F7    0355          BNE SAMPLE
C2F4 60      0356          RTS
C2F5 A9EC    0357 ATNLO    LDA #WRITE+ATNLOW
C2F7 8D0CA0  0358          STA PCR
C2FA 60      0359          RTS
```

```
C2FB AD00A0   0360 EOILO  LDA PORTB
C2FE 29FE     0361        AND #$FE
C300 8D00A0   0362        STA PORTB
C303 60       0363        RTS
C304 20F5C2   0364 PRIMAD JSR ATNLO
C307 BDE4C0   0365        LDA OUTAB1,X
C30A E8       0366        INX
C30B 20C9C2   0367        JSR SEND
C30E BDE4C0   0368        LDA OUTAB1,X
C311 E8       0369        INX
C312 861F     0370        STX WHO
C314 20C9C2   0371        JSR SEND
C317 60       0372        RTS
C318 20F5C2   0373 UNTAL  JSR ATNLO
C31B A95F     0374        LDA #UNTALK
C31D D005     0375        BNE UNLIS1
C31F 20F5C2   0376 UNLIS  JSR ATNLO
C322 A93F     0377        LDA #UNLISN
C324 20C9C2   0378 UNLIS1 JSR SEND
C327 2050C3   0379        JSR ATNHI
C32A 60       0380        RTS
C32B 20F5C2   0381 HEADER JSR ATNLO
C32E 20EBC2   0382        JSR SAMPLE
C331 204DC3   0383        JSR PRIMHI
C334 A001     0384        LDY #1
C336 B92000   0385 NLOOP  LDA STRING,Y
C339 20C9C2   0386        JSR SEND
C33C C8       0387        INY
C33D C420     0388        CPY STRING
C33F D0F5     0389        BNE NLOOP
C341 20FBC2   0390        JSR EOILO
C344 B92000   0391        LDA STRING,Y
C347 20C9C2   0392        JSR SEND
C34A 201FC3   0393        JSR UNLIS
C34D 2004C3   0394 PRIMHI JSR PRIMAD
C350 A9EE     0395 ATNHI  LDA #WRITE+ATNHIG
C352 8D0CA0   0396        STA PCR
C355 60       0397        RTS
C356 2C00A0   0398 ACC1   BIT PORTB
C359 30FB     0399        BMI ACC1
C35B AD01A0   0400        LDA PORTA
C35E 48       0401        PHA
C35F A909     0402        LDA #9
C361 8D00A0   0403        STA PORTB
C364 18       0404        CLC
C365 AD00A0   0405        LDA PORTB
C368 2910     0406        AND #$10
C36A F001     0407        BEQ ACC3
C36C 38       0408        SEC
C36D A90B     0409 ACC3   LDA #$B
C36F 8D00A0   0410        STA PORTB
C372 2C00A0   0411 WART   BIT PORTB
C375 10FB     0412        BPL WART
C377 A90D     0413        LDA #$D
C379 8D00A0   0414        STA PORTB
C37C 68       0415        PLA
C37D 49FF     0416        EOR #$FF
C37F 60       0417        RTS
C380          0418        .END
```

FORTH (4)

10. Der innere Sprachaufbau

FORTH ist eine 'gefädelte' schnell ausführende Sprache. In diesem Abschnitt soll die innere Sprachstruktur dargestellt werden, also insbesondere auch das Prinzip des Fädelns.

Unter FORTH sind folgende Betriebsweisen des Systems möglich:

Tischrechnermodus
Kompilierung
Ausführungsmodus unter Programm

Tischrechnermodus

In diesem Modus führt FORTH die Eingaben sofort aus. Wenn wir eintippen 4 4 + . (RETURN), so antwortet das System unmittelbar mit 8 OK. Hier erfolgte die Eingabe direkt von der Tastatur. Die Folge der unmittelbar auszuführenden Anweisungen (FORTH-Worte) mag gleichermaßen aber auch von einem anderen Eingabestrom herkommen, z.B. vom Texteditor her, von Massenspeichern oder von den sog. 'screens', die auch ein vereinbartes Format für Massenspeicherung darstellen.

Der Tischrechnermodus zeichnet sich dadurch aus, daß die Anweisungen nur einmal ausgeführt werden. Das System hat keine Anweisung, sich ein Programm zu 'merken'. Und man muß hinzufügen: Im Tischrechnermodus ist FORTH eine Interpretersprache. Das System muß den Eingabestrom zum Zwecke der unmittelbaren Ausführung interpretieren und dabei die schon bekannten FORTH-Worte aufsuchen und zur Ausführung bringen.

Kompilierung

Anders im Kompilermodus: Hier wird der Eingabestrom zu einem Programm umgesetzt, das beliebig wiederholbar ist. In einer Kompilierung werden neue FORTH-Worte gebildet, die in ihrer Summe das Programm ausmachen. Der Kompilermodus wird durch die uns schon bekannten Schlüsselworte Doppelpunkt (:) bzw. CODE (für Assemblierung) hereingerufen und mit dem Semikolon (;) bzw. END-CODE wieder verlassen (siehe auch Abschnitt 3.3). Nach dem Doppelpunkt bzw. nach CODE erwartet das System einen Namen/einen Bezeichner, unter dem die neue Definition in die Reihe der ausführbaren FORTH-Worte (das Dictionary) eingetragen werden soll.

Zwischen dem Namen und dem abschließenden Semikolon/END-CODE kann dann eine beliebige Menge von schon bekannten FORTH-Worten stehen. Mit der Kompilierung wird eine neue Definition ein gleichberechtigtes, aufschlagbares und ausführbares FORTH-Wort wie alle anderen vor ihm.

Was ist nun Kompilierung? Hier zunächst folgende Erklärung: Im Kompilermodus legt das System zunächst einen Namenskopf mit weiterer Verwaltungsinformation für die neue Definition an. Der auf den Namen folgende Eingabestrom wird nun (wie im Tischrechnermodus) ebenfalls interpretiert, aber nicht zur unmittelbaren Ausführung gebracht. Stattdessen 'fädelt' das System eine Reihe von Wortadressen in die neue Definition ein. Die Wortadressen sind dabei Pointer auf den ausführenden Teil der in der neuen Definition benutzten FORTH-Worte. Oder im übertragenen Sinne: Die Wortadressen sind eine Tabelle für indirekte Sprünge auf Unterprogramme. Das hat den Vorteil, das das neue FORTH-Wort nie mehr bezüglich der in ihm enthaltenen Elemente interpretiert werden muß. Durch seine Wort-Tabelle ist es künftig unmittelbar und schnell ausführbar.

Zum Beispiel nachstehend der bytemäßige Aufbau des kompilierten Wortes ALLOT, das n Speicherzellen reserviert. - Zur Vervollständigung ist darauf hinzuweisen, daß das FORTH-System auch durch die 'defining words' in den Kompilermodus versetzt wird. Zu dieser Gruppe zählen CONSTANT, VARIABLE, USER und VOCABULARY.

```
HEX ' ALLOT NFA 10 DUMP
 B9F1   85 41 4C 4C        Namenskopf mit Flag- und Zählbyte 85
 B9F5   4F D4 E2 B9        String ALLOT, Linkfeld, Codefeld B756
 B9F9   56 B7 57 B8        Wortadressen von `DP´, plus-store und ';
 B9FD   C7 B6 5B B5
OK
```

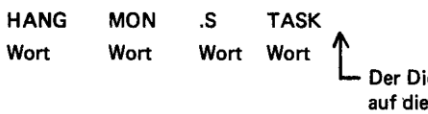
Ausführungsmodus unter Programm

Ein Programm besteht aus einer Folge von bereits bekannten (bereits kompilierten) FORTH-Worten. Dabei ist typisch, daß das Hauptprogramm des Anwenders im zuletzt definierten FORTH-Wort steht, das seinerseits hierarchisch tiefer stehende Programm-Module aufruft. Im Programm-Modus führt das FORTH-System einen eigenen Programmzähler, der die Wortadressen der Definitionen abrastert und die damit bezeichneten Routinen zur Ausführung bringt.

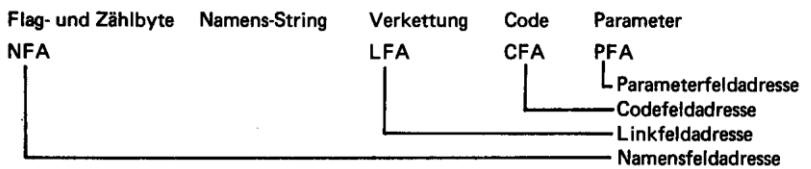
Bei den ausführenden Routinen (Worten) unterscheidet man 'primaries' und 'secondaries'. Erstere stellen prozessorabhängig den Kern der Sprache dar. Sie sind in Maschinensprache geschrieben. 'Secondaries' sind in der Sprache FORTH geschrieben und werden letztendlich durch 'primaries' ausgeführt.

10.1 Der Aufbau der FORTH-Worte

Das 'dictionary' (die Menge der ausführbaren FORTH-Worte) besteht aus einer in sich verketteten Aneinanderreihung von FORTH-Worten, z.B. für die obersten Einträge beim AIM-FORTH:

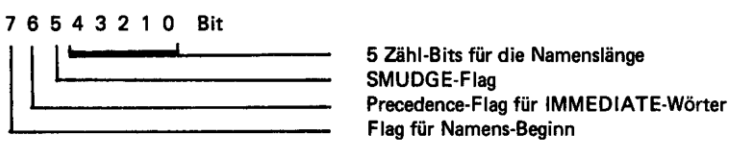


Wir schauen jetzt gewissermaßen mit einer Lupe in FORTH-Worte hinein und finden folgenden grundsätzlichen Aufbau und die Bezeichnungen:



Das Flag- und Zählbyte

Betrachten wir zunächst das erste Byte in dieser Kette. Hier sein Aufbau:



Die unteren 5 Bit geben an, wieviele Byte der Namens-String hat. Man kann also Namen bis zu 32 Byte Länge verwenden (2^5). Bit 7 kennzeichnet den Beginn eines Namens. Bit 6 ist das 'precedence bit'. Ist es gesetzt, so ist das Wort IMMEDIATE, d.h. es führt in der Kompilierungsphase unmittelbar aus (davon später). Bit 5 ist das SMUDGE-Flag. Dieses Bit macht das Wort auffind- und benutzbar. Wenn die Kompilierung wegen eines Versehens der Programmierers verunglückt, so fehlt dieses Flag. Das Wort erscheint zwar mit VLIST, kann aber nicht benutzt werden. Auch ein FORGET des Wortes ist nicht möglich. Man muß zuvor den Befehl SMUDGE geben, der das SMUDGE-Flag EX-ODERt. Dafür nachstehende Bedienungsfolge:

```
   : TEST UNBEKANNT 5 5 + . ;
UNBEKANNT ?
VLIST
  314 TEST
OK
' TEST
TEST ?
SMUDGE FORGET TEST OK
```

Der Namens-String

Die Buchstaben des Bezeichners stehen als ASCII-Zeichen im Speicher. Im letzten Byte (Zeichen) ist Bit 7 gesetzt, um eine Begrenzung zu geben. Das Flag- und Zählbyte des Namens wird durch folgende Befehlssequenz adressiert:

```
' Name NFA        (Tick, Name, NFA)
```

Die Verkettung

Im Link-Feld (2 Byte) steht die Namensfeldadresse (NFA) des voraufgehenden FORTH-Wortes. Die Verkettungsinformation dient dem Interpreter. Er sucht das Vokabular rückwärts von der letzten Definition her ab (von LATEST). Findet er im Linkfeld die Information 0000, so ist das unterste FORTH-Wort erreicht. Das Linkfeld einer Definition wird mit folgender Anweisungsfolge erreicht:

```
' Name LFA        (Tick, Name, LFA)
```

Das Codefeld

Im Codefeld steht eine Wortadresse, und zwar die Adresse einer FORTH-Routine, die die neue Definition zur Ausführung bringt. Man findet hier die Wortadressen der 'defining words' : (Doppelpunkt), CONSTANT, VARIABLE, USER und VOCABULARY oder von 'defining words' die der Anwender geschaffen hat.

Eine Sonderrolle spielt das 'defining word' CODE: Im Codefeld steht die Adresse des direkt dahinter folgenden Parameterfeldes. Das neue Wort wird also durch den Maschinen-(Assembler-)code ausgeführt, der im Parameterfeld steht.

Die Codefeldadresse einer Definition wird mit dem Wort CFA aufgeschlagen, also z.B.

```
' TASK  CFA       (Tick, Name, CFA)
```

Das Parameterfeld

Während das Codefeld also Wortfamilien mit gleichartigem Verhalten in der Abarbeitung bezeichnet (die Definition erfolgte mit Doppelpunkt, VARIABLE usw.), enthält das Parameterfeld die individuelle Information des neuen FORTH-Wortes. Bei Konstanten und Variablen findet sich hier Zahleninformation, bei CODE-Definitionen stehen hier Befehle in Maschinensprache und bei den im allgemeinen wohl mehrheitlichen Doppelpunkt-Definitionen stehen hier die gefädelten Wortadressen der benutzten FORTH-Worte (genauer gesagt: deren Codefeld-Adressen). Das Parameterfeld kann nach dem Willen des Anwenders aber auch andere Information enthalten, wenn er neue 'defining words' schafft, also z.B. ASCII-Strings, Arrays mit Zahlen- oder Prozeßinformation. Die Anlage neuer defining words mit <BUILDS und DOES> wird weiter unten besprochen.

Der FORTH-Befehl PFA (schlage die Parameterfeldadresse auf) liest vorwärts ab Stelle NFA der Definition. An den Befehlen TICK, NFA, LFA, CFA und PFA sehen wir, daß FORTH in sich transparent ist: Alle wichtige Information kann bequem aufgeschlagen und ggfs. auch verändert werden.

11. Das Dictionary - Das Vokabular

Der Zuordnungszähler

Das Dictionary ist, vor allem für den Interpreter, eine verkettete (gelinkte) Liste kompilierter FORTH-Worte. Das Vokabular wird durch den Dictionary Pointer DP verwaltet. Er enthält normalerweise den Zeiger auf die nächste freie Speicherzelle hinter dem Vokabular. Das Befehlswort HERE holt den Inhalt von DP auf den Datenstack. Die Definition von HERE ist damit:

```
: HERE DP @ ;
```

Man kann den Zeiger DP natürlich auch manipulieren. Der normale Weg führt über xx ALLOT, wobei xx die Menge Bytes ist, die im dictionary für Zwecke des Anwenders reserviert werden soll, z.B. für ein Array, einen Pufferbereich oder für mehrfach genaue Zahlen. ALLOT entspricht dem RMB oder dem *=*+xx (Reserve Memory Bytes) im Assembler. Wir haben damit die Definition:

```
: ALLOT DP +! ;  Erhöhe DP um das Argument auf dem Datenstack
```

Mehr mit dem Holzhammer kann man DP auf jeden beliebigen Wert setzen, um z.B. mit Offset zu kompilieren, z.B.:

```
1000 DP !          Setze DP auf 1000
44 CONSTANT ALPHA
```

Mit VLIST sehen wir dann den bewirkten Adressensprung zu ALPHA.

Weitere Befehle zur Kontrolle des Vokabulars

Die erwähnten 'defining words' schreiben Information byte- oder wortweise in neue Definitionen ein und benutzen dabei die FORTH-Worte

```
,        (n ---)    Comma speichert 1 Wort (2 Byte) ab Adresse HERE
C,       (b ---)    C-Comma speichert 1 Byte bei Adresse HERE
```

Die Definition dieser Worte entspricht (s.a. Abschnitt 8):

```
: , HERE ! 2 DP +! ;    Schreibe 2 Byte nach HERE, DP=DP+2
: C, HERE C! 1 DP +! ;  Schreibe 1 Byte nach HERE, DP=DP+1
```

Es ist dem Anwender unbenommen, mit diesen beiden Befehlen in eigenen 'defining words' zu arbeiten oder z.B. ein Array oder einen String mit ihnen zu füllen.

Das Wort SMUDGE macht einen Eintrag in das Vokabular gültig. Wird SMUDGE zweimal angewandt, so ist das letzte Wort wieder 'unsmudged'.

FORGET NameX löscht alle Vokabulareinträge einschließlich und ab Wort NameX. Natürlich ist FORGET nur im RAM anwendbar. Und der Eintrag NameX muß das SMUDGE-Flag gesetzt haben. FORGET setzt den dictionary pointer DP entsprechend zurück. Es ist möglich, ein Vokabular gegen unbeabsichtigtes FORGET zu schützen. Das Wort FENCE bringt die Adresse einer Anwendervariablen auf den Stack, in die man eine Adresse einschreiben kann, unterhalb derer ein FORGET verhindert wird.

Im Zusammenhang mit FORGET ist das 'dummy word' TASK zu erwähnen. TASK ist ein Leerwort, es tut nichts, es ist nur vorhanden, und zwar im RAM. Es hat die Definition: : TASK ;
Damit ist TASK mit FORGET löschbar und mit FORGET TASK löscht man auch alle von Anwender später definierten Worte. Man ist also in der Lage, einen mißglückten Programmierversuch zu löschen und die Systemmeldungen NOT UNIQUE zu vermeiden, wenn man im folgenden Durchgang wieder die alten Bezeichner verwendet. In Quelltexten für die Kompilierung steht daher fast immer am Anfang

```
FORGET TASK : TASK ;
```

Das Wort FORGET hat aber auch für die sog. Programm-Overlaytechnik Bedeutung. Damit ist es auch auf kleinen Systemen möglich, beliebig große FORTH-Programme zu fahren (man hat einen beliebig großen virtuellen Speicher). Wir kommen im Rahmen der Massenspeicherung bei den 'screens' darauf zurück. Zum Prinzip ist zu sagen, daß man hinter einen die 'screens' aufrufenden Verwaltungsteil ein Leerwort wie z.B. DUMMY setzt. Die Overlay-Screens beginnen dann jeweils mit FORGET DUMMY : DUMMY ;. Die neuen Anweisungen in den screens finden damit wieder einen bereinigten Arbeitsspeicher.

LATEST wurde bereits erwähnt. Diese Funktion liefert auf dem Datenstack die Namensfeldadresse (NFA) des obersten (zuletzt definierten) FORTH-Wortes ab (innerhalb des hereingerufenen Vokabulars, s.u.).

Das reservierte Wort IMMEDIATE wurde bereits im Zusammenhang mit dem Flag- und Zählbyte erwähnt. Es setzt das 'precedence flag' zu 1. Ein Wort, das durch ein IMMEDIATE direkt hinter dem Semikolon seiner Definition als 'unmittelbar' erklärt wurde, führt seine erklärten Aktionen bereits in der Kompilierung aus. Ein simples Beispiel:

```
: LOC ." DICTIONARY POINTER IS " HERE . ; IMMEDIATE
: DUMMY LOC ;
```

Während der Kompilierung von DUMMY erscheint der Inhalt des dictionary pointers als Meldung. Das Wort LOC führt also zur Kompilierungszeit aus.

Der Haupteinsatz des IMMEDIATE liegt in der Kompilersteuerung, insbesondere in der Erzeugung der Kontrollstrukturen BEGIN/WHILE/UNTIL/REPEAT/AGAIN oder IF/ELSE/THEN, die ja unmittelbar (und nicht erst zur Ausführungszeit) Steuerinformation in eine Definition einbinden müssen. Steuerinformation bedeutet hier FORTH-interne Verzweigungen/Branches.

STATE ist eine FORTH-interne Variable. Im Tischrechnermodus wird man sie zu 0 finden:

```
STATE @ . 0 OK
```

Man kann aber auch ein Wort bilden, das während der Kompilierung den STATE ausgibt, z.B.:

```
: SS STATE @ . ; IMMEDIATE
```

Die Benutzung erfolgt dann z.B. so:

```
: DUMMY SS ;
```

DUMMY zeigt dann das wegen der Kompilierung von Null verschiedene Statusbyte.

Die Systemvariable STATE wird abgefragt, um sicherzustellen, daß eine Reihe von Worten nur während einer Kompilierung gebraucht werden dürfen. Diese Abfrage erfolgt mit ?EXEC.

12. Neue Vokabulare

Es wurde schon erwähnt, daß man mit VLIST einen Abzug der definierten FORTH-Worte erhält. In Abschnitt 9 wurde dargestellt, daß man mit ASSEMBLER VLIST ein Vokabular gelistet bekommt, das den Befehlssatz des Assemblers zunächst listet und erst danach den der reinen FORTH-Definitionen. Mit dem Wort ASSEMBLER wird also ein zweites Vokabular aufschlagbar, das in sich natürlich auch wieder verkettet ist.

Der Benutzer hat nun die Freiheit, beliebig weitere Vokabulare einzubinden, die sich gegenseitig 'vollkommen ignorieren'. Man kann z.B. für einen Cross-Assembler ein MC6809-Vokabular schaffen, das wiederum die hinlänglich bekannten Mnemonics LDA, STA, usw. benutzt, die dann aber nicht mit denen des mitgelieferten 6502-Assemblers kollidieren, weil sie in einer in sich verbundenen Liste stehen, die zwar letztlich auf den Kern des FORTH mit VLIST zurückführt, die aber nicht mit den anderen Vokabular-Listen verbunden ist.

Das 'defining word' für die Schaffung neuer Vokabulare heißt VOCABULARY. Es wird in folgender Form benutzt:

```
VOCABULARY LATEIN
```

Es hinterläßt im laufenden Vokabular (dem CURRENT) einen Eintrag LATEIN, der bei Benutzung eine Vokabular-Umschaltung bewirkt. Neue weitere Definitionen werden in Form einer verketteten Liste dem neuen Vokabular angehängt. Es ist sogar möglich, einem Tochtervokabular baumartig ein weiteres anzuhängen:

```
ASSEMBLER DEFINITIONS
VOCABULARY LATEIN
: AGER ." ACKER" ;
```

Wenn wir jetzt VLIST geben, so finden wir die neuen Einträge LATEIN und AGER in die Worte des Assemblers eingebunden. Die Rückkehr zu FORTH erfolgt mit dem Wort FORTH. Wenn wir unter FORTH VLIST geben, so sind die neuen Einträge unsichtbar. - Feinheiten zur Kompilierung werden später nachzutragen sein. Im Moment ist es wichtig, die höchst nützlichen 'defining words' darzustellen.

13. Defining Words

Bei der Besprechung der Struktur von FORTH-Worten wurde unter CFA (Codefeldadresse) erwähnt, daß im Codefeld die Adresse der FORTH-Routine steht, die das Wort ausführt, exekutiert. Soweit der Benutzer noch keine eigenen defining words geschrieben hat, findet er hier die Codefeldadressen von : (Doppelpunkt), CONSTANT, VARIABLE, USER oder VOCABULARY. Sofern

die Definition alternativ mit dem Schlüsselwort CODE eingetragen wurde, weist das Codefeld auf das unmittelbar anschließende Parameterfeld, in dem ein Maschinenprogramm steht.

Der Benutzer hat unter FORTH die Möglichkeit, eigene 'defining words' zu definieren, die ihm neue Datentypen beliebiger Ausdehnung und Komplexität schaffen, denn das den Datentyp exekutierende Grundwort kann beliebige Verarbeitungen a) zum Zeitpunkt der Kompilierung und b) zur Ausführungszeit der von ihm geschaffenen Wortfamilie vornehmen. Ein defining word hat die Grundstruktur

: NameX <BUILDS Anweisungsfolge1 DOES> Anweisungsfolge2 ;

Dabei bezeichnet NameX die Wortfamilie/den Datentyp (wie z.B. die Wortfamilien CONSTANT, VARIABLE usw.). <BUILDS und DOES> klammern eine Anweisungsfolge1 ein, die zur Zeit der Kompilierung ausgeführt wird, also z.B. Berechnungen und Abspeicherungen. Die Anweisungsfolge2 (nach DOES>) legt das Verhalten des Tochterwortes von NameX zur Zeit der Exekution (des Tochterwortes) fest. Dieses Verhalten kann ein beliebiges mit FORTH formulierbares sein. Ein Datentyp STRING legt z.B. legt z.B. zur Ausführungszeit die Startadresse und die Länge des Strings auf dem Datenstack ab. Ein Datentyp ARRAY rechnet z.B. die auf dem Datenstack übergebene Nummer n des anzusprechenden Elementes in die Adresse um, unter der im Array das Element n gefunden werden kann. Zur Ausführungszeit mag sogar geprüft werden, ob das Element überhaupt eine erlaubte Nummer trägt und nicht zur Bereichsüberschreitung führt usw..

Beispiele für defining words wurden in den Heften 25 und 26 dieser Zeitschrift schon verschiedentlich gegeben, sie werden hier nur zitiert, z.B. DCONST (doppelt genaue Zahlen), DARRAY mit obigen prüfenden Eigenschaften und $CONST für Strings u.a.m..

Es ist ein großer Vorteil des FORTH, daß man mit defining words beliebige neue Datentypen aufbauen kann, die dem Anwendungsfall angepaßt sind. Während sie normalerweise global sind, d.h. von allen Programmteilen aus angesprochen werden können, mögen sie durch die Overlaytechnik auch zu lokalen Daten des Overlays werden. Andere Sprachen mögen da nur zwischen INTEGER, REAL, STRING, POINTER oder RECORD unterscheiden. FORTH akzeptiert nicht nur solche Definitionen, sondern auch jede andere des Benutzers, z.B. BDC-Datentypen mit 24 Stellen oder so Genauigkeit und Exponent. In Norddeutschland würde man dazu sagen: Das ist ein Klax!

'Defining Words', die man nach dem Gesagten wohl übersetzen sollte als 'einen Datentyp bestimmende Befehle', gehören mithin zum nützlichsten Inventar der Sprache FORTH. Der Leser möge hierzu möglichst viele Studien in Programmen und Übungen ausführen, um die notwendige Sicherheit zu erlangen. Auch bei Fehlbedienungen wird sich der Computer nicht in einer brenzligen Wolke verabschieden.

Dazu einige Anregungen: In der Anweisungsfolge1 (zur Kompilierzeit) werden nach den Beobachtungen gelegentlich einige Umrechnungen vorgenommen, meist endet sie mit den FORTH-Worten ',' oder 'C,' (Comma, C-Comma) für die Ablage von 2 oder 1 Byte im dictionary. Die Anweisungsfolge2 (zur Ausführungszeit) kann beliebig komplex sein, oft holt sie nur ein Byte oder ein Wort, sie dupliziert, rechnet oder prüft nach Bedarf, wie wir es bei den zitierten Beispielen für doppelt genaue Zahlen und Strings gesehen haben.

(WIRD FORTGESETZT) R.L.

Bücher

Krizan,P.; Kaufmann, K-D.: Spaß mit BASIC für Anwender, *Idea Verlag, Puchheim 1982, ISBN 3-88793-005-3, 176 Seiten, DM 26. Das Buch ist für Besitzer von Klein-, Hobby- und Heim– Personalcomputern eine nützliche Ergänzung zu dem beliebten Lern- und Lehrbuch Spaß mit BASIC. Es handelt sich um eine Programmsammlung, ein Potpouri in folgender Gliederung: Mathematisch-technischer Themenkreis, allgemeiner und wirtschaftlicher Themenkreis, Grafik, Testprogramme, Spielprogramme, Sprache, Lernprogramme. Insbesondere der Anfänger wird in die Lage versetzt, mit dem Abtippen der Programme eine nützliche Sammlung aufzubauen und auf diesem Wege seine Kenntnisse in BASIC zu verbessern. Insgesamt sind es 47 Programme.*

Cross-Assembler unter FORTH

Wie es im konkreten Fall für den MC6809 und den MC6805 von Motorola beim Verfasser schon ausgearbeitet wurde, eignet FORTH u.a. hervorragend, um Cross-Assembler für eine andere CPU als die des Entwicklungssystemes zu formulieren. Der Vorteil liegt auf der Hand: Man muß nicht für jede CPU, die man einsetzen will, ein spezielles Entwicklungssystem anschaffen. Wie an anderer Stelle dargelegt, ist es sogar möglich, eine bedingte Assemblierung zu veranlassen und auch Macros generieren zu lassen, die je nach Vorstellungskraft des Programmierers aus entsprechend übergebenen Parametern und einem einzigen FORTH-Wort 10 oder mehr ausführbare Maschinenbefehle erzeugen. Es kommt nach solchen erfolgreichen Übungen fast von selbst, wenn einen das FORTH-Fieber endgültig erfaßt.

Hier soll kein ausgearbeiteter Cross-Assembler vorgestellt werden. Es ist vielmehr Absicht, die Leitideen für die Schaffung eines solchen darzulegen, um zu eigenen Arbeiten anzuregen. Ziel sollte es dabei sein, auf dem Entwicklungssystem Code zu erzeugen, der direkt mit den Hilfsmitteln des Betreibers in ein EPROM übergebrannt werden kann. Man muß sich also in einem virtuellen Adreßraum bewegen und sich diese Relativierung immer vor Augen halten.

Der Leser erinnert, daß man dem FORTH neue Befehlsworte durch Definition zwischen Doppelpunkt und Semikolon (: Name Anweisungsfolge ;) oder zwischen den Worten CODE (Name) und END-CODE (für Assembler-Befehle) anfügen kann. Auch eine Mischform ist möglich mit ;CODE. Die anderen bekannten 'defining words' erzeugen Datentypen oder Vokabulare.

Jedes neue FORTH-Wort enthält eine Kopfleiste mit Namens- und Verwaltungsinformation. Diese Kopfleiste darf in keinem Fall in die Cross-Assembly geraten, dorthin gehört nur der für die Ziel-CPU generierte Code. Wir müssen also eine Bereichstrennung für den Speicherraum vornehmen, in dem sich das FORTH normalerweise mit seinen Konstanten, Variablen und neuen FORTH-Worten bewegt, und jedem Bereich, in den der Zielcode abzulegen ist.

Wir benötigen also ein Wort ASSIGN oder ähnlich, das die RAM-Basisadresse bestimmt, von der ab (nicht überschneidend) der Zielcode abgelegt wird. Man wird etwa formulieren:

```
0 VARIABLE RAMAREA
: ASSIGN RAMAREA ! ;
```

Die zweite essentielle Einrichtung ist die Schaffung eines Zuordnungszählers (PC oder Sternadresse), der die Belegung des Zielbereiches verwaltet. PC ist daher vom dictionary pointer DP des FORTH getrennt zu halten, denn wir wollen ja keine Eintragung in das FORTH-Vokabular machen. Es ist also zu formulieren

```
0 VARIABLE PC
: ORG PC ! ;
```

Die ORG-Anweisung setztt mithin den Zuordnungszähler auf das auf dem Datenstack übergebene Argument, also z.B.

```
100 ORG      (PC=100)
```

PC enthält damit eine virtuelle Adresse, die nicht nur für die Ablage von Bytes relativ zu RAMAREA wichtig ist, sondern auch für die Berechnung von Verzweigungsweiten und für die Adressierung von absoluten Sprüngen.

Der erzeugte Maschinencode ist als eine Tabelle anzusehen, die ab der mit ASSIGN bestimmten tatsächlichen Adresse des Entwicklungssystemes abgelegt wird. Die nächste jeweils freie physische Ablagestelle ergibt sich dann aus

```
: LOCATE RAMAREA  PC @ + ;
```

Es ist nun zu fragen, wie man Bytes (Opcodes, Adressen Textketten usw.) an der durch LOCATE bestimmten Adresse ablegt. Das geschieht mit dem Befehlswort '!' (store) oder mit 'C!' (c-store). Der Schreibbefehl für ein auf dem Datenstack übergebenes Byte lautet damit

```
: WRITE LOCATE C! 1 PC +! ;
```

Im zweiten Teil der Anweisung wird der Zuordnungszähler zugleich um 1 weiterbewegt.

Damit steht das Gerüst eines Cross-Assemblers, seine Bekleidung hängt von der zu programmierenden Zielmaschine und von den Komfortwünschen des Programmierers ab. Der hexadezimale Befehlssatz des Zielprozessors und seine Adressierungsarten sind zu analysieren. Es ist z.B. zu prüfen, wie eine Indizierung einen Grund-Opcode verändert und ob (wie beim MC6809) Pre-Bytes erforderlich sind und wieviele Post-Bytes durch die Art der Indizierung anzuhängen sind. Für Verzweigungen sind Berechnungsroutinen für die Sprungweiten zu entwickeln zusammen mit der Prüfung, ob die Verzweigung noch im erreichbaren Abstand liegt. Man wird also eine weitere Variable (z.B. MODE) benutzen, die von den Operatoren für die Adressierungsart verändert wird und von deren Inhalt man die weiteren Bearbeitungen abhängig macht.

Für die Übersetzung der Assembler-Mnemonics wie STA usw. wird man sich vorwiegend der sog. 'defining words' bedienen, um gleichartige Befehle zu Wortfamilien zu gruppieren. Am einfachsten liegen die Dinge bei den 'inherent'-Befehlen, die aus nur 1 Byte bestehen:

```
: INHERENT <BUILDS C, DOES> @ WRITE ;
```

Zur Zeit der Kompilierung legen sie 1 Byte unter dem Mnemonic z.B. TAX, ab, zur Ausführungszeitholen sie es auf den Datenstack und schreiben es in die nächste freie Zelle (WRITE). Worte der INHERENT-Familie wären beim 6502 z.B.

```
HEX
18 INHERENT CLC
60 INHERENT RTS  usw.
```

Mehr-Byte-Befehle sind entsprechend der Adressierungsart komplizierter, sind aber (von Sonderfällen abgesehen) immer gruppenweise in das Schema anderer defining words einreihbar. Beispielsweise ist zu verhindern, daß ein Statement wie 55 # STA, durchläuft (Sonderfall!).

Kontrollstrukturen mit BEGIN, ... UNTIL, oder mit IF, ... ELSE, ... THEN stellen einen weiteren Schwierigkeitsgrad dar. Man braucht nicht nur Lokalisierungsinformation an den Endpunkten dieser Statements (z.B. VIRTUAL-LOC PC @ ;), sondern auch Kontrollinformation, ob die Kontrollstrukturen in der richtigen Abfolge und Zuordnung stehen. Das geschieht im allgemeinen mit dem Wort ?PAIRS. Weiterhin müssen an den Endpunkten möglichst Verzweigungsbefehle erzeugt werden (auch Long Branches), nur notfalls JMP-Befehle. Rückwärts sind von UNTIL, aus, bei THEN, bei WHILE, bei IF, die endgültigen Verzweigungsweiten einzusetzen. Die strukturierenden Worte sind als IMMEDIATE zu kenhzeichnen, damit sie bereits bei der Assemblerung ausführen.

Bei Vergleichsoperationen ist man zunächst geneigt, dieselben Bezeichner wie im native FORTH zu verwenden. Damit würde man jedoch die nach wie vor auch unter Cross-Assembler intakten Prüf-Fähigkeiten des FORTH abschneiden. Man entschließt sich dann zu Operatoren, die auch in anderen höheren Sprachen geläufig sind, z.B. '.GT' für Greater Than usw..

Großer Komfort entsteht, wenn auf mögliche Programmierfehler durch WARNINGs hingewiesen wird, die nicht nur eine Nummer ausgeben, sondern auch auf die möglichen Ursachen hinweisen. Weiterhin sollte man sicherstellen, daß im Rahmen des Programmierbaren eine beliebige Reihenfolge von Operatoren und Operanden (auch symbolischen Operanden) eingegeben werden kann. Die abschließende Eingabe wird gleichwohl immer der mnemonische Befehl sein. Man bedenke auch, daß BCC oder FCC Hexadezimalziffern sein können und daher nicht für Branch on Carry Clear oder Form Constant Characters herhalten können. Menmonics schreibt man daher möglichst gleichmäßig als BCC, usw. und Assembler-Anweisungen als .FCC oder .BYT.

Ein Cross-Assembler sollte weiterhin ein eigenes VOCABULARY bilden.

Das Schreiben eines Assemblers oder eines Cross-Assemblers ist wohl die gründlichste Methode, den Befehlssatz der Ziel-CPU kennenzulernen, was dann allerdings noch nicht die Übung in demselben ausmacht, die erst durch das Programmieren gewonnen wird.

R.L.

Rolf Schöne, 8000 München 19

FORTH Turtle-Grafik für GDP EF9365

A. Problem

Die Implementierung eines Grafiksystems auf einem Mikrocomputer braucht heute wohl keine Rechtfertigung mehr. So ist nur noch eine Wahl zwischen koordinatenbezogener Grafik und der sogenannten Turtle-Grafik zu treffen. Wir wollen hier die Turtle-Grafik wählen. Argumente dafür finden sich in 1).

B. Lösung

Es sind nun die Hardwarekomponenten auszuwählen und dafür Steuerprogramme zu schreiben.

1. Hardware

Für Hobbyisten kommt als Anzeigegerät wohl nur ein Raster Scan Display in Frage, so daß auch preiswerte Fernsehgeräte verwendet werden können. Die Erzeugung der Videosignale wollen wir nicht der Zentraleinheit überlassen, um diese für wichtigere Aufgaben einsetzen zu können. Daher benötigen wir einen speziellen Controller. Unsere Wahl fällt auf den Graphic Display Processor EF9365, mit dem wir eine Auflösung von 512 x 512 Bildpunkten erreichen. 2) 3)

2. Software

Nun wird es Zeit, sich für eine Programmiersprache zu entscheiden, in der die unterstützende Software zu schreiben ist.

2.1 Maschinenprogramme, zu deren Entwicklung auch PL/65 herangezogen werden kann, werden sicher am schnellsten laufen und schnell bewegte Graphiken möglich machen. Gerade PL/65 unterstützt uns dabei auch in der Anwendung moderner Programmiertechniken (strukturiertes Programmieren, Top-Down-Design). Da jedoch abzusehen ist, daß wir trigonometrische Funktionen einsetzen müssen, wollen wir diesen Weg vorerst nicht weiter verfolgen.

2.2. BASIC scheidet wegen der geringen Laufgeschwindigkeit aus.

2.3 PASCAL wäre ausreichend schnell, böte Unterstützung bei den oben genannten Programmiertechniken und zudem leistungsfähige Datentypen, ist auf kleinen Systemen wie z.B. dem AIM 65 aber nicht verfügbar. (Anm. d. Hrsg.: Doch, für AIM gibt es Instant PASCAL!)

2.4 Warum also nicht FORTH? Diese Sprache gestattet eine bequeme Parameterübergabe an die Kommandos für die Turtle (über den Stack), komfortables Debugging (ohne jedesmal neu zu kompilieren zu müssen), den Einbau von Winkelfunktionen auch in kleine FORTH-Systeme und ist überdies auf Kleinrechnern verfügbar.

3. Nun muß ich zugeben, daß ich FORTH beim Schreiben dieses Programmes selbst erst gelernt habe. Man möge mir deshalb nachsehen, daß ich hier ein sicher noch zu verbesserndes Programm vorstelle, das zudem nur ein - allerdings bereits lauffähiges - minimales Grafiksystem darstellt. Weiter unten sollen daher einige mögliche und wünschenswerte Erweiterungen besprochen werden. Doch nun zum Programm:

3.1 In den Zeilen 8-16 des Listings werden die notwendigen Kommandos an den Controller vereinbart. Da wir dessen Leistungsfähgkeit vorab nicht voll ausschöpfen wollen, genügt hier eine Untermenge aller möglichen Kommandos.

3.2 Die für diese Kommandos notwendigen Hardwareregister des Controllers sind in den Zeilen 18-25 vereinbart.

3.3 Zeilen 30-32 initialisieren die Controlregister (vorerst nur für durchgezogene Vektoren und nicht für alpanumerische Zeichen).

3.4 Die beiden Worte in 33-40 erlauben das Einschreiben von Kommandos in das Commandregister.

3.5 Die folgenden Worte in 41-49 steuern die Feder. Nach PENDOWN hinterläßt die Turtle quasi

eine helle Spur auf dem Monitor. Mit ERASER kann eine solche Spur wieder gelöscht werden, indem man die Turtle nochmals darüber bewegt. Nach PENUP kann die Turtle bewegt werden, ohne eine Spur zu hinterlassen.

3.6 Das Wort in den Zeilen 50-53 vertauscht Lo- und Hi-Byte des zuoberst auf dem Stack liegenden 16-Bit Wortes. Es ist notwendig, weil die Positionsregister für X und Y die Reihenfolge Hi-Lo vorschreiben.

3.7 In 54 bis 91 finden sich die Funktionen 'sin' und 'cos'. Die dafür erforderliche Tabelle wird in 2 Schritten aufgebaut, um den begrenzten Stack von 6502-Systemen nicht überlaufen zu lassen.

3.8 Zeilen 92-108 beinhalten die eigentlichen Turtle-Kommandos: FORW bewegt die Turtle um eine Anzahl Einheiten vorwärts, die zuvor auf den Stack gebracht werden muß (hier 256). Die Bewegungsrichtung ist durch den Inhalt der Variablen HEAD bestimmt. BACKW wirkt wie FORW, bewegt die Turtle jedoch rückwärts, also entgegen der durch HEAD festgelegten Richtung.

LEFT ändert die Blickrichtung der Turtle um so viel Grad nach links, wie auf dem Stack vorgefunden werden. RIGHT ändert die Richtung nach rechts.

3.9 Ab Zeile 109 stehen einige Hilfsworte zur Steuerung der Turtle und für die Verwaltung des Bildschirms: SETXY setzt die Turtle auf die im Stack vorgefundenen Koordinaten. CENTER setzt sie in die Bildschirmmitte. HEADZERO läßt die Turtle nach 0 Grad (rechts) blicken. CLEAR löscht den Bildschirm und führt CENTER und HEADZERO aus. ?WHERE zeigt den gegenwärti gen Zustand der Turtle.

4. FORTH lebt von der Definition neuer Worte. So liegt es nahe, weitere Worte zu definieren, welchen die Leistungen des Controllers auch voll ausnutzen, hier aber aus Platzgründen nicht gezeigt werden sollen.

4.1 Durch ein neues Wort FORW ließe sich die Beschränkung der Vektorlänge auf max. 255 umgehen.

4.2 Mit Hilfe der in 4) oder 5) vorgeschlagenen Stringfunktionen lassen sich alphanumerische Zeichen und Sonderzeichen darstellen (aufrechte und geneigte Darstellung horizontal und vertikal, auch gemischt und in verschiedenen Größen).

4.3 Es sind vier Vektortypen möglich:

- durchgezogen
- punktiert
- gestrichelt
- strichpunktiert

4.4 Daneben werden durch den Controller noch unterstützt:

- der Anschluß eines Lichtgriffels
- verschiedene weitere Vektorarten
- eine Blockgrafik, die es gestattet, Flächen zu zeichnen.

4.5 Durch Hardwareerweiterungen ist direktes Auslesen das Bildspeichers, ein Read/Modify/Write-Betrieb und Farbgrafik möglich.

4.6. Durch den Ersatz von reinen FORTH-Worten durch Maschinencode (den FORTH-Assembler verwenden!) ergeben sich schnellere Ausführungszeiten und damit bei Bedarf schnell bewegbare Grafiken.

5. Das vorgestellte Programm ist für fig-FORTH V1.1. geschrieben. Für den AIM ist die in Zeile 18 vereinbarte Adresse zu ändern und in den Zeilen 84 und 101 das Wort MINUS jeweils durch NEGATE zu ersetzen.

C. Beispiel

Als Beispiel für den Einsatz der vorgeschlagenen Turtle-Grafik-Worte möge die SPIRALE dienen.

Literatur:

1) Abelson H., diSessa A.: Turtle Geometry, ISBN 0-262-01063-1
2) THOMSON-ECFIS: Datenblatt GDP 9365/9366
3) Klein R.-D.: CRT-Controller unterstützt Grafikfunktionen, ELEKTRONIK 8/81, S. 63 ff.
4) Rockwell: AIM 65 User's Manual
5) Löhr R.: Strings für FORTH, MICRO MAG Nr. 25, S. 3 ff.

Ein alternativer Programmiervorschlag des Herausgebers: Mit den FORTH-Worten 46 TABLE0 SIN0 und 45 TABLE1 SIN1 werden im vorstehenden Programm zwei Arrays SIN0 und SIN1 geladen, und zwar durch die 'defining words' TABLE0 und TABLE1. Bei großen Arrays ist es sicher günstiger, das Befehlswort 'comma' (,) im Zusammenhang mit VARIABLE zu verwenden. 'Comma' speichert ein auf dem Stack übergebenes Argument bei HERE in das 'dictionary' ein und bewegt den Zeiger DP (dictionary pointer) um zwei Bytes. Dieses Vorgehen hätte auch den Vorteil, daß man die Argumente in aufsteigender Reihenfolge notieren könnte (statt wie hier in umgekehrter). Also z.B.:

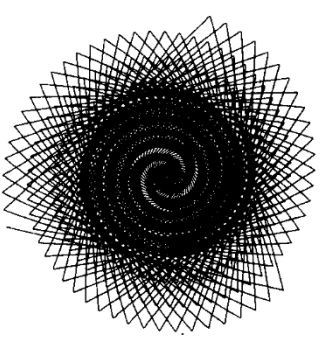
```
0 VARIABLE TABLE0 175 , 349 , 523 , 698 , OK
: SIN0 SWAP 2 * + @ ; OK
( DER DOES>-TEIL IST NUN IN SIN0 ) OK
```

```
0001   ( ------------------------------------ )
0002   (    FORTH Worte fuer Turtle Grafik    )
0003   ( mit Graphic Display Processor EF9365 )
0004   ( R.Schoene       TURGRAF      22.04.82 )
0005   ( ------------------------------------ )
0006
0007   : BASE !
0008   00000000 CONSTANT PN    ( Command/Status )
0009   00000001 CONSTANT ER
0010   00000010 CONSTANT DN
0011   00000011 CONSTANT UP
0012   00000100 CONSTANT RDY
0013   00000110 CONSTANT CLR
0014   00010001 CONSTANT VEC   ( Vector Command )
0015   00010011 CONSTANT DX-
0016   00010101 CONSTANT DY-
0017   HEX
0018   F5E0     CONSTANT SEL ( Hardware Address )
0019   SEL      CONSTANT CSREG      ( Registers )
0020   SEL 1 +  CONSTANT CREG1
0021   SEL 2 +  CONSTANT CREG2
0022   SEL 5 +  CONSTANT DXREG
0023   SEL 7 +  CONSTANT DYREG
0024   SEL 8 +  CONSTANT XREG
0025   SEL A +  CONSTANT YREG
0026   DECIMAL
0027   0        VARIABLE HEAD  ( Turtle Heading )
0028   0        VARIABLE CMD
0029
0030   : INIT ( --- cont vecs, non cycl, pen up )
0031     0 CREG1 C! 0 CREG2 C!
0032   ;
0033   : WAITREADY ( --- )
0034     BEGIN
0035       CSREG C@ RDY AND RDY =
0036     UNTIL
0037   ;
```

```
0038    : COMMAND ( Command --- )
0039      WAITREADY CSREG C!
0040    ;
0041    : PENDOWN ( --- )
0042      PN COMMAND DN COMMAND
0043    ;
0044    : ERASER ( --- )
0045      ER COMMAND DN COMMAND
0046    ;
0047    : PENUP ( --- )
0048      UP COMMAND
0049    ;
0050    : H()L ( word --- bytes switched )
0051      PAD ! PAD C@ PAD 3 + C! PAD 1+ C@
0052      PAD 2 + C! PAD 2 + @
0053    ;
0054    : TABLE0 ( Sinus 0...45 Degrees )
0055      (BUILDS 0 DO , LOOP DOES)
0056      SWAP 2 * + @
0057    ;
0058                                  7071 6947 6820 6691 6561
0059     6428 6293 6157 6018 5878 5736 5592 5446 5299 5150
0060     5000 4848 4695 4540 4384 4226 4067 3907 3746 3584
0061     3420 3256 3090 2924 2756 2588 2419 2250 2079 1908
0062     1736 1564 1392 1219 1045  872  698  523  349  175
0063        0
0064       46 TABLE0 SIN0
0065
0066    : TABLE1 ( Sinus 46...90 Degrees )
0067      (BUILDS 0 DO , LOOP DOES)
0068      SWAP 2 * + @
0069    ;
0070    10000 9998 9994 9986 9976 9962 9945 9925 9903 9877
0071     9848 9816 9781 9744 9703 9659 9613 9563 9511 9455
0072     9397 9336 9272 9205 9135 9063 8988 8910 8829 8746
0073     8660 8572 8480 8387 8290 8192 8090 7986 7880 7771
0074     7660 7547 7431 7314 7193
0075       45 TABLE1 SIN1
0076
0077    : S180
0078      DUP 90 ) IF 180 SWAP - THEN
0079      DUP 45 ) IF 46 - SIN1 ELSE SIN0 THEN
0080    ;
0081    : SIN ( Value Degrees --- Value )
0082      360 MOD DUP
0083      0( IF 360 + THEN DUP
0084      180 ) IF 180 - S180 MINUS ELSE S180 THEN
0085      10000 */MOD SWAP DUP
0086       4999 ) IF DROP 1+ ELSE
0087      -4999 ( IF 1 - THEN ENDIF
0088    ;
0089    : COS ( Value Degrees --- Value )
0090      360 MOD 90 + SIN
0091    ;
0092    : FORW ( Distance --- )
0093      DUP VEC CMD ! HEAD @ 360 MOD HEAD !
0094      HEAD @ COS DUP ABS DXREG C!
0095        0( IF DX- CMD @ OR CMD ! THEN
```

```
0096      HEAD @ SIN DUP ABS DYREG C!
0097        0< IF DY- CMD @ OR CMD ! THEN
0098      CMD @ COMMAND
0099    ;
0100    : BACKW ( Distance --- )
0101      MINUS FORW
0102    ;
0103    : LEFT ( Degrees --- )
0104      HEAD +!
0105    ;
0106    : RIGHT ( Degrees --- )
0107      HEAD @ SWAP - HEAD !
0108    ;
0109    : SETXY ( X Y --- )
0110      H()L YREG ! H()L XREG !
0111    ;
0112    : CENTER ( --- )
0113      256 256 SETXY
0114    ;
0115    : HEADZERO ( --- )
0116      0 HEAD !
0117    ;
0118    : CLEAR ( --- )
0119      CLR COMMAND CENTER HEADZERO
0120    ;
0121    : ?WHERE ( --- print Turtle State )
0122      ." X:" XREG @ H()L . ."   "
0123      ." Y:" YREG @ H()L . ."   "
0124      ." H:" HEAD @ .
0125    ;
0126


0001    ( Beispiel : Spirale mit FORTH/TURGRAF )
0002    (   Aufruf : #Ecken SPIRALE            )
0003
0004    0 VARIABLE ANGLE
0005    0 VARIABLE SIDE
0006
0007    : SPIRALE ( n --- )
0008      360 SWAP / 1+ ANGLE ! 0 SIDE !
0009      INIT CLEAR PENDOWN
0010      BEGIN SIDE @ 255 <
0011        WHILE SIDE @ 1+
0012              DUP FORW SIDE !
0013              ANGLE @ RIGHT
0014      REPEAT ;
0015
```



Michael Zimmermann, 6102 Pfungstadt

FORTH im Eigenbau (3)

Bei den bisher beschriebenen Routinen spielten sich alle Operationen ausschließlich auf dem Datenstack ab. In vielen Fällen, insbesondere zum Retten von Zwischenergebnissen, kann es sinnvoll sein, Daten zeitweilig vom Datenstack auf den Returnstack zu sichern, von wo man sie später wieder zurückholt. Hierzu dienen die Routinen

TOR TO R
Überträgt ein Byte vom Datenstack auf den Returnstack, verkürzt dabei den Datenstack um dieses Byte.

FROMR FROM R
Überträgt 1 Byte vom Returnstack auf den Datenstack und verlängert ihn dabei

Kontrollstrukturen

Die bisherigen FORTH-Worte dienten lediglich der Erstellung linearer Programme. Irgendwelche Wiederholungen von Programmteilen waren noch nicht möglich. Sie hatten eine Länge von:

2 Byte In der Regel nur Aufruf
3 Byte Aufruf und 1 Datenbyte, z,B. PUSHB
4 Byte Aufruf und 2 Datenbyte, z.B. PUSHW.

Für die Kontrollstrukturen ist das Wissen um diese Befehlslängen aus Gründen von Bedeutung, die später zu erläutern sind. Da FORTH eine strukturierte Sprache ist, wurde auch in dieser Implementation bewußt auf einen GOTO-Befehl verzichtet, jedoch großer Wert auf BEGIN- und DO-Strukturen gelegt. Diese Strukturen sollen über die Programmdokumentation hinaus eingehend beschrieben werden, weil sie in ihrer Wirkungsweise bedeutend komplexer sind, als die bis jetzt beschriebenen rein linearen, insbesondere in ihrer Interaktion zwischen Daten- und Returnstack. Sie sind auch nur in ihrer gegenseitigen Interaktion zu verstehen.

BEGIN-Schleifen

Hier handelt es sich um eine Kontrollstruktur, die ein oder mehrere Male durchlaufen werden soll. Hierbei sind bei manchen Formen Abfragen auf Bedingungen möglich, die die Anzahl der Durchläufe steuern. - Im Gegensatz zur nachstehenden DO-Schleife ist die Anzahl der Durchläufe am Beginn unbestimmt und wird einzig durch das Eintreten oder Ausbleiben einer Bedingung bestimmt, nicht aber durch Schleifenparameter, die auf dem Datenstack übergeben werden.

Der BEGIN-Befehl selber markiert den Anfang der Schleife, der zu durchlaufende Teil steht dahinter. BEGIN hat das ihm folgende Programmstück für eine wiederholte Ausführung vorzubereiten. Dies geschieht dadurch, daß der FORTH-Programmzähler, der bei Ausführung bereits auf den folgenden Befehl zeigt, auf den Returnstack übertragen wird.

BEGIN - AGAIN

Eine wiederholte Ausführung kann jetzt dadurch erfolgen, daß dieser Zählerstand dem Returnstack wieder entnommen wird. Im einfachsten Fall erfolgt das mit dem AGAIN-Befehl, der eine unbeschränkte Schleife ermöglicht. AGAIN entnimmt ein Wort, das durch BEGIN auf den Returnstack gebracht wurde, und legt es in den FORTH-Programmzähler. Da hierbei der Returnstack verkürzt wurde, muß AGAIN dieses Wort wieder dem Programmzähler entnehmen und im Stack ablegen, um eine weitere Wiederholung zu gewährleisten. Da dieses Wort die Adresse des Befehles ist, der BEGIN folgt, wird die Folge ein weiteres Mal abgearbeitet:

```
.WOR BEGIN
        Führe eine beliebige Anzahl von Befehlen ohne Begrenzung aus.
.WOR AGAIN
```

BEGIN - UNTIL

Diese Form der Schleife prüft nach Ausführung des wiederholt zu durchlaufenden Teiles eine Bedingung und veranlaßt die Wiederholung, wenn diese Bedingung noch nicht eingetreten ist. Symbolisch sei das folgendermaßen ausgedrückt-

.WOR BEGIN
Abarbeiten der Befehle, Setzen der Bedingung
.WOR UNTIL

Aus dieser Beschreibung ergeben sich die Funktionen, die UNTIL wahrnehmen muß. Da alle Datenübertragungen über den Datenstack vorgenommen werden sollen, muß zuerst das Byte, das das Ergebnis des vorgelagerten Bedingungs-Setzens darstellt, dem Datenstack entnommen und dieser entsprechend verkürzt werden. Ist der Inhalt dieses Bytes ungleich Null, d.h. wurde eine FALSE-Bedingung gesetzt, so ist dem Returnstack die auf den Anfang der Schleife zeigende Rücksprungadresse zu entnehmen und dieser entsprechend zu verkürzen. Ist der Inhalt gleich Null, d.h. es liegt eine TRUE-Bedingung vor, so ist wie beim AGAIN-Befehl zu verfahren.*) Aus dieser Beschreibung ist zu ersehen, daß die Schleife mindestens einmal durchlaufen wird, da ein Test auf Wiederholung erst am Ende erfolgt.

BEGIN - WHILE - REPEAT

Soll dies vermieden werden, so ist die Schleifenform mit der WHILE-Struktur zu wählen. Diese hat folgenden Aufbau:

.WOR BEGIN
Eine Anzahl von Befehlen, die sowohl etwas ausführen, als auch einen Bedingungsschlüssel auf dem Stack ablegen
.WOR WHILE
Aufruf eines 2-Byte FORTH-Befehles, in der Regel Aufruf einer Unterroutine
.WOR REPEAT

Bei dieser Konstruktion werden die Befehle zwischen BEGIN und WHILE unbedingt ausgeführt. Diese haben in jedem Fall ein logisches Byte auf dem Datenstack zu hinterlassen *). Dieses wird im WHILE-Befehl unter Verkürzung vom Datenstack geladen. Ist es Null, d.h. TRUE *), so wird der Folgebefehl (in der Regel der Aufruf einer Unterroutine) ausgeführt. Der darauf folgende REPEAT-Befehl verzweigt wieder zu BEGIN. - Ist das Test-Byte ungleich Null, d.h. FALSE, so wird der Return-Stack zuerst um die Rücksprungadresse zu BEGIN verkürzt und der Programmzähler um 4 erhöht. Dies bewirkt einen Sprung über den Ausführungsbefehl und den REPEAT-Befehl hinweg auf die Adresse, die dem Schleifenende folgt. Auf diese Art wird die Wiederholung verlassen 1).

Der in seiner Wirkung angedeutete REPEAT-Befehl hat exakt dieselben Funktionen wie der AGAIN-Befehl, er braucht also nicht weiter erläutert zu werden. Aus dieser Beschreibung ergibt sich eine Beschränkung der Sprungweiten auf 50 Sprünge vorwärts, beim WHILE ein Sprung nur über den 2-Byte Folgebefehl und über das darauf folgende REPEAT. Diese Beschränkung vermeidet eine größere Berechnung und Verwaltung der Sprungweiten, wie sie nur mit einem reinen Interpreter oder mit Einführung von Sprungbefehlen möglich wäre. Dem Wunsch nach Einfachheit stünde ein Interpreter entgegen. Auch die Einschränkung auf nur einen Aufruf hinter WHILE fördert das strukturierte Programmieren.

DO - Schleifen

Neben dieser Schleifenform, die bedingungsabhängig zu einem Ende findet, ist mit der DO - LOOP eine weitere Kontrollstruktur implementiert, die eine feste Anzahl von Durchläufen ermöglicht, und zwar unabhängig von irgendwelchen Ereignissen. Die Parameter für Anfangs- und Endwert werden vor Eintritt in die Schleife auf dem Datenstack abgelegt. Da oft aus der Schleife heraus ein Wert ermttelt wird, der dem Folgeprogramm ebenfalls auf dem Datenstack zur Verfügung stehen sollte, ist es sinnvoll, die Schleifenparameter auf den Returnstack zu übernehmen und dort auch die Schleifendurchläufe zu verwalten. Daher besteht die erste Aktion des DO-Befehles darin,

die Worte für Endwert und Anfangswert vom Datenstack auf den Returnstack zu übernehmen. Dabei wird natürlich der Datenstack um die Parameterzahl verkürzt. Der Anfangswert dient zugleich als Laufvariable. Sodann wird die eigentliche Wiederholungsmöglichkeit von der DO-Anweisung dadurch eingestellt, daß die Adresse des Folgebefehles ebenfalls auf dem Returnstack abgelegt wird. Die Schleife beginnt dann mit der Ausführung des auf DO folgenden Befehles.

LOOP

Die Anweisungsfolge wird durch den LOOP-Befehl abgeschlossen. Dieser erhöht die Schleifenvariable und prüft sie gegen den Endwert. Zum Zwecke des Verständnisses ist zu verdeutlichen, in welchem Zustand sich der Return-(=Hardware-)stack sich bei der Ausführung befindet:

SP	———
SP+1	Rücksprungadresse Low
SP+2 •	Rücksprungadresse High
SP+3	Schleifenvariable Low
SP+4	Schleifenvariable High
SP+5	Endwert Low
SP+6	Endwert High

Da der Stackpointer der CPU nicht direkt zur Adressierung verwendet werden kann, muß das X-Register diese Aufgabe übernehmen. Da dieses aber auch als Pointer auf den Datenstack Verwendung findet, ist es zunächst in XSAVE zu sichern. Sodann kann der Stackpointer in das X-Register übertragen werden. Darauf wird die Schleifenvariable erhöht und gegen den Endwert verglichen. Sind beide Werte gleich, so wird die Schleife beendet. Das geschieht dadurch, daß der Returnstack um die 6 Werte zur Verwaltung eben dieser Schleife erniedrigt wird; auch dies eine Operation, die wegen der geringen arithmetischen Fähigkeiten des Stackpointers im X-Register durchgeführt wird.

Anschließend werden Stackpointer und X-Register wieder berichtigt. - Wurde bei der Endabfrage ein Ungleich zwischen Laufvariabler und Endwert erkannt, so wird die Schleife ein weiteres Mal durchlaufen. Die hierfür erforderlichen Aufgaben werden von der REPEAT-Anweisung wahrgenommen.

IND

Im Ablauf einer derartigen Schleife ist es vielfach interessant, die Laufvariable weiter zu verarbeiten. Hierzu dient der IND-Befehl, der ihren Wert auf dem Datenstack ablegt. Er sichert zunächst das X-Register und überträgt den Stackpointer in dieses. Darauf werden die 2 Bytes der Laufvariablen in den Akkumulator und in das Y-Register geladen. Damit hat X seine Aufgabe zur Adressierung des Returnstack erfüllt und kann deshalb aus XSAVE zurückgeladen werden. Anschließend wird auf dem Datenstack erst einmal Platz geschaffen und dann der Inhalt von Akku und Y-Register dort abgelegt.

IF - Befehl

Neben den beschriebenen Schleifenanweisungen sind Anweisungen implementiert, die einer bedingten Befehlsausführung dienen. Diese Routinen haben folgende Form:

```
.WOR IF
        2-Byte FORTH-Befehl, der bei TRUE ausgeführt wird
.WOR ELSE
        2-Byte FORTH-Befehl, der bei FALSE ausgeführt wird
.WOR THEN
```

Der Passus ELSE und der ihm folgende Befehl können entfallen, wenn für den FALSE-Fall keine gesonderten Verarbeitungen vorgesehen sind. Der IF-Befehl erwartet auf dem Stack ein logisches Byte, er entnimmt dieses dort und legt es auf dem Returnstack ab. Ist dieses Byte ungleich Null, d.h. FALSE *), so wird der folgende 2-Byte-Befehl übersprungen, die Ausführung wird mit ELSE oder THEN fortgesetzt. Ist das Prüfergebnis aber gleich, d.h. TRUE, so wird sequentiell mit dem Folgebefehl fortgefahren, die Kontrolle läuft auch in diesem Falle wieder auf ELSE bzw. TRUE.

ELSE - Befehl

Dem ELSE folgt ein 2-Byte-Befehl, der im FALSE-Fall auszuführen ist. Es wird das Prüfbyte, das von IF auf den Returnstack gebracht wurde, geladen und sofort wieder dorthin kopiert. Ist dieses Prüfbyte gleich Null, d.h. TRUE, so ist der Folgebefehl zu überspringen, im anderen Fall ist er auszuführen.

THEN - Befehl

Die bedingte Ausführung endet in jedem Fall mit dem THEN-Befehl, der das Prüfbyte dem Returnstack entnimmt und es vergißt.

Vergleichsbefehle

Die Vergleichsbefehle stehen in engen Zusammenhang mit den bedingten Ausführungen. Während die Prüfung von Ein-/Ausgabeleitungen wohl meist über logische Operationen gehen wird, haben diese Befehle ihren Schwerpunkt beim Test von Zahlenwerten.

Grundsätzlich werden die Werte Next on Stack und Top on Stack miteinander verglichen. Beide werden dabei dem Stack entnommen. Das Ergebnis des Vergleiches wird als logisches Byte auf den Stack geschrieben und kann weiter benutzt werden. Die drei Befehle hierfür sind:

EQ	Prüfen auf gleich
LT	Prüfen auf kleiner
GE	Prüfen auf größer oder gleich

Diese Befehle haben einen identischen Aufbau. Zunächst wird im Y-Register als Vorbesetzung das Prüfbyte für FALSE gesetzt. Sodann wird die Vergleichsoperation durchgeführt und, abhängig von deren Ergebnis, am Ende zur Ja- oder zur Nein-Routine verzweigt (YESC bzw. NOTC).In YESC wird das TRUE-Kennzeichen nach Y geladen und mit NOTC fortgefahren. - Diese Routine besteht in einem Verkürzen des Datenstacks und der Ablage des Y-Registers dort. Das Prüfbyte stammt also entweder aus der Vorbesetzung oder aus YESC.

Beispiele

Die vorstehend beschriebenen Befehle sollen durch eine Reihe von Beispielen verdeutlicht werden. Ihnen allen ist gemein, daß der Peripherieport A mit der Adresse A001 abgefragt wird und daß entsprechend Bit 7 unterschiedliche Aktionen eingeleitet werden. Da gleichzeitig auch die Vergleichsbefehle dargestellt werden, ergibt sich eine Form, die den Inhalt des Bytes von A001 immer auf Wortlänge erweitert.

Im Beispiel 1 wird über den IF-Befehl entweder der Buchstabe T (Bit 7=0) oder F (bit 7=1) ausgegeben. Im Beispiel 2 bleibt die Kontrolle solange in der BEGIN-UNTIL-Schleife, bis Bit 7=1 wird. Im Beispiel 3 wird solange der Buchstabe T ausgegeben, bis Bit 7=1 wird. Das letzte Beispiel stellt eine DO-Schleife vor, die die Summe aller Zahlen von 1 bis 3 ermittelt.

Zusammenfassung

Mit diesen Darstellungen für eine einfache FORTH-Implementation hofft der Verfasser einen Weg gewiesen zu haben, der diese Sprache von den Bindungen an eine bestimmt Maschine frei macht, wie sie bei einer ROM-Version eigentlich immer gegeben wären. Durch den Umfang von nur 3 Pages ergeben sich sicher viele Möglichkeiten. Implementiert sind die wichtigsten, insbesondere auch die steuernden Funktionen. Auf einen Interpreter wurde verzichtet.

Anmerkungen des Herausgebers: Diese eigenständige Implementierung weicht in einigen Punkten ten vom üblichen FIG-FORTH ab. Dazu die Hinweise: *) Auch Prüfbytes oder Flags haben im FIG FORTH 2 Byte, wobei 0=FALSE, ungleich 0=TRUE. 1) FIG-FORTH läßt auch viele dazwischenliegende Wortadressen zu. Man beachte auch die weiteren Hinweise des Autors.

```
---------------------------------------------------------
TO R  TO RETURN STACK
          0405        0317
0405 0704    0318 TOR    .WOR 0+2
0407 B501    0319        LDA 1,X        ;BYTE VOM DATA-STACK HOLEN
```

```
040A 48      0321        PHA             ;UND BYTE IM RETURN-STACK ABLEGEN
040B 4C3002  0322        JMP NEXT

-------------------------------------------------------------
FROM R FROM RETURN STACK
040E 1004    0324 FROMR  .WOR *+2
0410 68      0325        PLA             ;BYTE VOM R-STACK HOLEN
0411 CA      0326        DEX DATA-STACK VERKUERZEN
0412 9501    0327        STA 1,X         ;UND BYTE IM DATA-STACK ABLEGEN
0414 4C3002  0328        JMP NEXT

-------------------------------------------------------------
BEGIN SCHLEIFENANFANG
             0417         0330
0417 1904    0331 BEGIN  .WOR *+2
0419 A582    0332        LDA PC          ;SYMBOLISCHEN PROGRAMMCOUNTER
041B 48      0333        PHA             ;AUF DEM RETURN-STACK ABLEGEN
041C A583    0334        LDA PC+1
041E 48      0335        PHA
041F 4C3002  0336        JMP NEXT

-------------------------------------------------------------
AGAIN UNENDLICHE WIEDERHOLUNG
             0422         0338 AGAIN

-------------------------------------------------------------
REPEAT WIEDERHOLUNG
             0422         0340
0422         0341
0422 2404    0342 REPEAT .WOR *+2
0424 68      0343        PLA             ;PC AUS DEM STACK ABRUFEN
0425 8583    0344        STA PC+1
0427 68      0345        PLA
0428 8582    0346        STA PC
042A 4C1904  0347        JMP BEGIN+2     ;PC WIEDER AUF STACK ABLEGEN

-------------------------------------------------------------
UNTIL BEDINGTE SCHLEIFE
             042D         0349
042D 2F04    0350 UNTIL  .WOR *+2
042F E8      0351        INX             ;STACK VERKUERZEN
0430 B500    0352        LDA 0,X         ;VERGLEICHSBYTE VOM STACK LADEN
0432 F0F0    0353        BEQ AGAIN+2     ;BEI FALSE SCHLEIFE WIEDERHOLEN
0434 68      0354        PLA             ;BEI TRUE RETURN-STACK VERKUERZEN
0435 68      0355        PLA
0436 4C3002  0356        JMP NEXT

-------------------------------------------------------------
WHILE BEDINGTE SCHLEIFE
             0439         0358
0439 3B04    0359 WHILE  .WOR *+2
043B E8      0360        INX             ;STACK VERKUERZEN
043C B500    0361        LDA 0,X         ;VERGLEICHSBYTE LADEN
043E F07A    0362        BEQ NSKP        ;BEI FALSE EINFACH WEITERGEHEN
0440 68      0363        PLA             ;BEI TRUE RETURN-STACK VERKUERZEN
0441 68      0364        PLA
0442 A904    0365        LDA #$04        ;UND 4 BYTE -AUSFUEHRUNG UND REPEAT-
0444 D06B    0366        BNE SKPX        ;UEBERSPRINGEN

-------------------------------------------------------------
DO SCHLEIFENANWEISUNG
             0446         0368
0446 4804    0369 DO     .WOR *+2
0448 B504    0370        LDA 4,X         ;ENDWERT VOM DATENSTACK AUF DEN
```

```
044A 48      0371        PHA             ;RETURN-STACK DUPLIZIEREN
044B B503    0372        LDA 3,X
044D 48      0373        PHA
044E B502    0374        LDA 2,X         ;EBENSO ANFANGSWERT DER ZUR LAUFVARIABLE WIRD
0450 48      0375        PHA
0451 B501    0376        LDA 1,X
0453 48      0377        PHA
0454 E8      0378        INX             ;DATENSTACK VERKUERZEN UM ENTNOMMENE PARAMETER
0455 E8      0379        INX
0456 E8      0380        INX
0457 E8      0381        INX
0458 4C1904  0382        JMP BEGIN+2     ;UND JETZT PC AUF STACK

-------------------------------------------------------------------
LOOP SCHLEIFEN-ENDANWEISUNG
             045B         0384
045B 5D04    0385 LOOP   .WOR *+2
045D 8680    0386        STX XSAV        ;X-REGISTER ALS DATENSTACK-POINTER SICHERN
045F BA      0387        TSX             ;POINTER UEBERTRAGEN
0460 FE0301  0388        INC $103,X      ;SCHLEIFENVARIABLE ERHOEHEN
0463 D003    0389        BNE *+5
0465 FE0401  0390        INC $104,X
0468 BD0301  0391        LDA $103,X      ;AUF ENDWERT UEBERPRUEFEN
046B DD0501  0392        CMP $105,X
046E D014    0393        BNE RLOOP       ;UNGLEICH WEITER SCHLEIFEN
0470 BD0401  0394        LDA $104,X
0473 DD0601  0395        CMP $106,X
0476 D00C    0396        BNE RLOOP       ;WIE OBEN WEITER SCHLEIFEN
0478 E8      0397        INX             ;ENDWERT ERREICHT, RETURN-STACK VERKUERZEN
0479 E8      0398        INX
047A E8      0399        INX
047B E8      0400        INX
047C E8      0401        INX
047D E8      0402        INX
047E 9A      0403        TXS             ;POINTER ZURUECKUEBERTRAGEN
047F A680    0404        LDX XSAV
0481 4C3002  0405        JMP NEXT
0484         0406 RLOOP  *=*             ;WEITER SCHLEIFEN
0484 A680    0407        LDX XSAV
0486 4C2404  0408        JMP REPEAT+2    ;JETZT PC HOLEN UND WIEDER IN STACK ABLEGEN

-------------------------------------------------------------------
IND LAUFVARIABLE AUF DEN DATENSTACK UEBERNEHMEN
          .  0409         0410
0489 8B04    0411 IND    .WOR *+2
048B 8680    0412        STX XSAV
048D BA      0413        TSX
048E BD0301  0414        LDA $103,X
0491 BC0401  0415        LDY $104,X
0494 A680    0416        LDX XSAV
0496 CA      0417        DEX
0497 CA      0418        DEX
0498 9501    0419        STA 1,X
049A 9402    0420        STY 2,X
049C 4C3002  0421        JMP NEXT

-------------------------------------------------------------------
IF BEDINGTE BEFEHLSAUSFUEHRUNG
             049F         0423
049F A104    0424 IF     .WOR *+2
04A1 E8      0425        INX             ;STACK VERKUERZEN UM PRUEFBYTE
```

```
04A2 B500    0426        LDA 0,X         ;PRUEFBYTE LADEN
04A4 48      0427        PHA             ;UND AUF RETURN-STACK ABLEGEN
04A5 D008    0428        BNE SKP2        ;BEI UNGLEICH = FALSE FOLGEBEFEHL UEBERSPRINGEN
04A7 F011    0429        BEQ NSKP        ;BEI GLEICH = TRUE FOLGEBEFEHL AUSFUEHREN

-------------------------------------------------------------
ELSE AUSFUEHRUNG DER FALSE-BEDINGUNG
             04A9         0431
04A9 AB04    0432 ELSE   .WOR *+2
04AB 68      0433        PLA             ;PRUFEBYTE VOM RETURN-STACK ENTNEHMEN
04AC 48      0434        PHA             ;UND WIEDER ABLEGEN
04AD D00B    0435        BNE NSKP        ;BEI UNGLEICH = FALSE FOLGEBEFEHL AUSFUEHREN
04AF A902    0436 SKP2   LDA #$02        ;FUER SPRUNG UEBER 2 BYTE VORBEREEITEN
04B1 18      0437 SKPX   CLC             ;FUER JEDEN SPRUNG VORBEREITEN
04B2 6582    0438        ADC PC          ;NIEDERES PC-BYTE ADDIEREEN
04B4 8582    0439        STA PC          ;UND WIEDER ABLEGEN
04B6 9002    0440        BCC *+4         ;BEI CARRY HOEHERES BYTE
04B8 E683    0441        INC PC+1        ;ERHOEHEN
04BA 4C3002  0442 NSKP   JMP NEXT

-------------------------------------------------------------
THEN BEDINGUNGSSENDE
             04BD         0444
04BD BF04    0445 THEN   .WOR *+2
04BF 68      0446        PLA             ;PRUEFBYTE IN JEDEM FALL DEM RETURN-STACK ENTNEHMEN
04C0 4C3002  0447        JMP NEXT        ;UND FORTFAHREN

-------------------------------------------------------------
EQ PRUEFEN AUF GLEICH
             04C3         0449
04C3 C504    0450 EQ     .WOR *+2
04C5 A001    0451        LDY #1          ;UNGLEICH-KENNZEICHEN LADEN
04C7 B503    0452        LDA 3,X         ;LOW-BYTE VERGLEICHEN
04C9 D501    0453        CMP 1,X
04CB D008    0454        BNE NOTC        ;UNGLEICH VERZWEIGEN
04CD B504    0455        LDA 4,X         ;HIGH-BYTE VERGLEICHEN
04CF D502    0456        CMP 2,X
04D1 D002    0457        BNE NOTC
04D3 A000    0458 YESC   LDY #0          ;LADEN BEI BEDINGUNG ERFUELLT
04D5 E8      0459 NOTC   INX             ;IN JEDEM FALL STACK VERKUERZEN
04D6 E8      0460        INX
04D7 E8      0461        INX
04D8 9401    0462        STY 1,X         ;BEDINGUNGSERGEBNISS AUF STACK ABLEGEN
04DA 4C3002  0463        JMP NEXT

-------------------------------------------------------------
GE GREATER EQUAL
             04DD         0465
04DD DF04    0466 GE     .WOR *+2
04DF A001    0467        LDY #1          ;FALSE-ERGEBNISS LADEN
04E1 B503    0468        LDA 3,X         ;LOW-BYTE VERGLEICHEN
04E3 D501    0469        CMP 1,X
04E5 B504    0470        LDA 4,X
04E7 F502    0471        SBC 2,X
04E9 90EA    0472        BCC NOTC BEDINGUNG NICHT ERFUELLT
04EB B0E6    0473        BCS YESC        ;BEDINGUNG ERFUELLT

-------------------------------------------------------------
LT LESS THAN
             04ED         0475
04ED EF04    0476 LT     .WOR *+2
04EF A001    0477        LDY #1 FALSE-ERGEBNISS LADEN
04F1 B503    0478        LDA 3,X         ;LOW-BYTE VERGLEICHEN
```

```
04F3 D501   0479        CMP 1,X
04F5 B504   0480        LDA 4,X         ;HIGH-BYTE VERGLEICHEN
04F7 F502   0481        SBC 2,X
04F9 90D8   0482        BCC YESC BEDINGUNG ERFUELLT
04FB B0D8   0483        BCS NOTC        ;BEDINGUNG NICHT ERFUELLT

------------------------------------------------------------
RUN-PACKAGE
04FD        0485        *=$800          ;BEISPIEL 1
0800 4B02   0486        .WOR TCALL
0802 1908   0487        .WOR A001
0804 0503   0488        .WOR PUSHB      ;STACK UM 1 BYTE VERLAENGERN FUER VERGLEICH
0806 FF     0489        .BYT $FF
0807 1703   0490        .WOR PUSHW      ;VERGLEICHSWORT 8000=7FFF+1 AUF STACK BRINGEN
0809 0080   0491        .WOR $8000
080B ED04   0492        .WOR LT         ;(A001)<8000
080D 9F04   0493        .WOR IF
080F 2308   0494        .WOR JA         ;BIT 7=0
0811 A904   0495        .WOR ELSE
0813 2C08   0496        .WOR NEIN       ;BIT 7=1
0815 BD04   0497        .WOR THEN
0817 6302   0498        .WOR TRET
0819        0499 A001   *=*             ;HOLEN INHALT VON ADRESSE A001
0819 4B02   0500        .WOR TCALL
081B 1703   0501        .WOR PUSHW
081D 01A0   0502        .WOR $A001
081F 3403   0503        .WOR FETB
0821 6302   0504        .WOR TRET
0823        0505 JA     *=*             ;BUCHSTABE T AUSGEBEN
0823 4B02   0506        .WOR TCALL
0825 0503   0507        .WOR PUSHB
0827 54     0508        .BYT $54
0828 FA03   0509        .WOR EMIT
082A 6302   0510        .WOR TRET
082C        0511 NEIN   *=*             ;BUCHSTABE N AUSGEBEN
082C 4B02   0512        .WOR TCALL
082E 0503   0513        .WOR PUSHB
0830 46     0514        .BYT $46
0831 FA03   0515        .WOR EMIT
0833 6302   0516        .WOR TRET
0835        0517        *=$900          ;BEISPIEL 2
0900 4B02   0518        .WOR TCALL
0902 1704   0519        .WOR BEGIN
0904 1908   0520        .WOR A001
0906 0503   0521        .WOR PUSHB
0908 FF     0522        .BYT $FF
0909 1703   0523        .WOR PUSHW
090B 0080   0524        .WOR $8000
090D DD04   0525        .WOR GE
090F 2D04   0526        .WOR UNTIL
0911 6302   0527        .WOR TRET
0913        0528        *=$A00          ;BEISPIEL 3
0A00 4B02   0529        .WOR TCALL
0A02 1704   0530        .WOR BEGIN
0A04 1908   0531        .WOR A001
0A06 0503   0532        .WOR PUSHB
0A08 FF     0533        .BYT $FF
0A09 1703   0534        .WOR PUSHW
0A0B FF7F   0535        .WOR $7FFF
```

```
0A0D C304   0536        .WOR EQ
0A0F 3904   0537        .WOR WHILE
0A11 2308   0538        .WOR JA
0A13 2204   0539        .WOR REPEAT
0A15 6302   0540        .WOR TRET
0A17        0541        *=$B00          ;BEISPIEL 4
0B00 4B02   0542        .WOR TCALL
0B02 1703   0543        .WOR PUSHW      ;RECHENFELD AUF STACK BRINGEN
0B04 0000   0544        .WOR $0000
0B06 1703   0545        .WOR PUSHW      ;ENDWERT
0B08 0400   0546        .WOR $0004
0B0A 1703   0547        .WOR PUSHW
0B0C 0000   0548        .WOR $0000      ;ANFANGSWERT
0B0E 4604   0549        .WOR DO
0B10 B904   0550        .WOR IND        ;LAUFVARIABLE HOLEN
0B12 D102   0551        .WOR PLUS       ;ZU RECHENFELD ADDIEREN
0B14 5B04   0552        .WOR LOOP
0B16 B603   0553        .WOR DOT        ;AUSGABE RECHENFELD
0B18 6302   0554        .WOR TRET
0B1A        0555        .END
```

#

Hans-Georg Lange, 1000 Berlin 30

LISP

Eine Sprache wird wiederentdeckt

LISP wurde zum ersten Mal im Jahre 1959 auf einem IBM-Rechner 7090 implementiert. Eine Untermenge dieser Sprache war auch auf einer PDP1 installiert. LISP gehört somit zu den Veteranen der höheren Programmiersprachen und war lange Zeit in der Versenkung verschwunden. In neuerer Zeit ist jedoch das Stichwort 'künstliche Intelligenz' (AI) verstärkt im Zusammenhang mit selbstlernenden Maschinensteuerungen oder der Spracherkennung aufgetreten. Da außerdem inzwischen die Implementierung von LISP auf mehreren 6502-Rechnern gelungen ist, tritt diese Sprache wieder in den Vordergrund.

Zwei Eigenschaften unterscheiden LISP von allen anderen höheren Programmiersprachen:

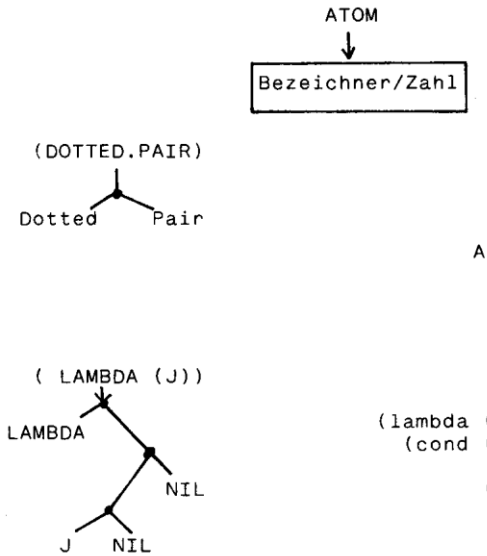
> LISP-Strukturen sind grundsätzlich Baumstrukturen nach dem Prinzip: linker Ast/rechter Ast usw..
>
> Es wird generell nicht zwischen Programmen und Daten unterschieden. Dadurch ist es wie in keiner anderen Sprache möglich, Programme durch Programme zu erzeugen und diese auch gleich auszuführen.

Wer von einer anweisungs- und schleifenorientierten Programmiersprache her kommt, ist von LISP zunächst enttäuscht: Die einzig (zunächst) vorhandenen Datentypen sind die Zeichenkette und die ganze Zahl. Von den bekannten Kontrollstrukturen existiert nur ein GO-Statement, das Einrichten einer Funktion ist relativ aufwendig.

Nach Einarbeitung werden auch die Vorteile deutlich: LISP-Bezeichner können für beliebig komplexe Strukturen stehen, die je nach Bedarf wachsen oder schrumpfen können. Die Zuweisungsfunktion SETQ ist das allgemeinste Assignment, das denkbar ist. Programme und Daten können beliebig miteinander vertauscht werden. Listen können nicht nur immer wieder sich selbst erzeugen, auch mutuelle Rekursion und das Erstellen einer zyklischen Liste ist mit einer Handvoll Befehlen möglich.

Das Editieren in LISP ist zunächst auch ungewohnt: Der Editor ist selbst eine Liste. Jeder Editierbefehl stellt einen Eingriff in eine aktuelle Listenstruktur dar. Fehler bei der Punktuation erzeugen eine völlig veränderte Struktur. Der Editor ist jedoch mit einem allgemeinen 'Pretty Print'-Algo-

rithmus ausgestattet, der die tatsächlich (wenn auch vielleicht nicht beabsichtigte) LISP-Struktur darstellt. Nachfolgend einige Beispiele für einfache LISP-Strukturen und als Beispiel für LISP-Programme die rekursive Berechnung von n! (n Fakultät).




```
(lambda (n)
  (cond ((eq n 0)
         1)
        (t (* n (fak (sub n 1))))))
```

#

Horst Brettin, 1000 Berlin 44

RENAME Disk 4040

Das Programm dient dem Ändern des Disketten-Namens auf der Commodore Floppy 4040. Es dürfte sich weitgehend selbst erklären. Zu beachten ist, daß an den angegebenen Stellen in den Zeilen 200 und 270 SHIFT-SPACES zu verwenden sind und dahinter 3x CURSOR LEFT. Die Konstante NOP$ besteht aus 16x SHIFT SPACE. Beendet wird das Programm mit RETURN ohne eine Eingabe, die Shifted Spaces in 200 und 270 helfen diesen Fall zu erkennen.

Es existiert ein ähnliches Programm, das nicht nur die Änderung des Namens, sondern - angeblich auch der ID -erlaubt. Vor diesem Programm (es kursiert unter dem Namen ID-Wechsler) möchte ich warnen: Es zeigt sich zwar hinterher im Catalog die neue ID, aber nur dort! Die Floppy erkennt weiterhin nur die alte ID. Das Ändern des Disk-Namens ist dagegen vollkommen ungefährlich. Wer das Programm mit der 3030 oder der 8050 verwenden möchte, muß die Zeile 230 entsprechend ändern.

```
100 REM RENAME DISK 4040
110 REM ----------------
120 OPEN 1,8,15:PRINT"■RENAME DISK■■
130 NOP$=CHR$(160):NOP$="                "
200 INPUT"DRIVE   ■■■■";D$:IF D$="0" OR D$="1" GOTO 230
205 REM          ↑ SHIFT-SPACE CRSR ←←←
210 IF D$="" GOTO 400
220 PRINT"■":GOTO 200
230 PRINT#1,"I"D$:C$=CHR$(144)+CHR$(65+VAL(D$))
240 PRINT#1,"M-R" C$ CHR$(20)
250 INPUT#1,N$
```

```
 260 PRINT"ALTER DISKNAME : "N$
 270 INPUT"NEUER DISKNAME     ";N$
 275 REM                    ↑ SHIFT-SPACE CRSR ←←←
 280 IF N$=NP$ GOTO 400
 290 PRINT#1,"N-W" CHR$(16)LEFT$(N$+NOP$,16)
 300 OPEN 2,8,2,"#":CLOSE 2
 310 GOTO 200
 400 CLOSE 1
 410 PRINT"ENDE
READY.
(C) by Horst Brettin                                              #
```

Horst Brettig, 1000 Berlin 44

Graph/Text für CBM 3001

Dieses Interruptprogramm schaltet für die obere Bildhälfte des CBM den Graphik- und für die untere Bildhälfte den Text-Modus der Zeichenwiedergabe ein. Es wird mit SYS 941 aktiviert und mit SYS 949 ausgeschaltet. Durch Erhöhen (Vermindern) der Timervorgabe um 512 wandert der Umschaltpunkt eine TV-Zeile tiefer (höher). Das Low-Byte der Timervorgabe sollte nicht verändert werden .Die Anpassung für andere Betriebssysteme ist sicher möglich.

```
                        GRAFIK/TEXT
                        UMSCHALTEN IM INTERRUPT

                ZEIT    EQU 10351           (C) 1982 BY
                IRQV    EQU $90            HORST BRETTIN
                IRQEND  EQU $E6E4

                        VIA 6522
                        --------
                T2L     EQU $E848
                T2H     EQU T2L+1
                ACR     EQU $E84B
                PCR     EQU ACR+1
                IFR     EQU PCR+1
                IER     EQU IFR+1

                        ORG $3AD

03AD  A9A0    ON        LDA#%10100000              SYS 941
03AF  A2D5              LDX#IRQIN
03B1  A003              LDY#IRQIN/256       ;HI-BYTE
03B3  D00E              BNE SET

03B5  AD4CE8  OFF       LDA PCR                    SYS 949
03B8  29FD              AND#%11111101
03BA  8D4CE8            STA PCR
03BD  A920              LDA#%00100000
03BF  A22E              LDX#$2E
03C1  A0E6              LDY#$E6

03C3  78      SET       SEI
03C4  8D4DE8            STA IFR             ;CLR-FLAG
```

```
03C7  8D4EE8          STA IER          ;SET/CLR ENABLE
03CA  A690            STX IRQV
03CC  8491            STY IRQV+1
03CE  A900            LDA#0
03D0  8D4BE8          STA ACR
03D3  58              CLI
03D4  60              RTS


03D5  AD4CE8  IRQIN   LDA PCR
03D8  2C4DE8          BIT IFR          ;INTERRUPT-QUELLE?
03DB  3012            BMI TIMER
03DD  29FD            AND#%11111101    ;GRAFIK-MODE
03DF  8D4CE8          STA PCR
03E2  A96F            LDA#ZEIT
03E4  A228            LDX#ZEIT/256
03E6  8D48E8          STA T2L
03E9  8E49E8          STX T2H          ;START TIMER 2
03EC  4C2EE6          JMP $E62E

03EF  0902    TIMER   ORA#%10          ;TEXT-MODE
03F1  8D4CE8          STA PCR
03F4  2C48E8          BIT T2L          ;CLR FLAG
03F7  4CE4E6          JMP IRQEND                                    #
```

Dr. A. Schnell, 5100 Aachen

Fernsteuerung eines Tonbandgerätes

Interruptgesteuertes Zählen externer Impulse bis ca. 10 kHz am CBM

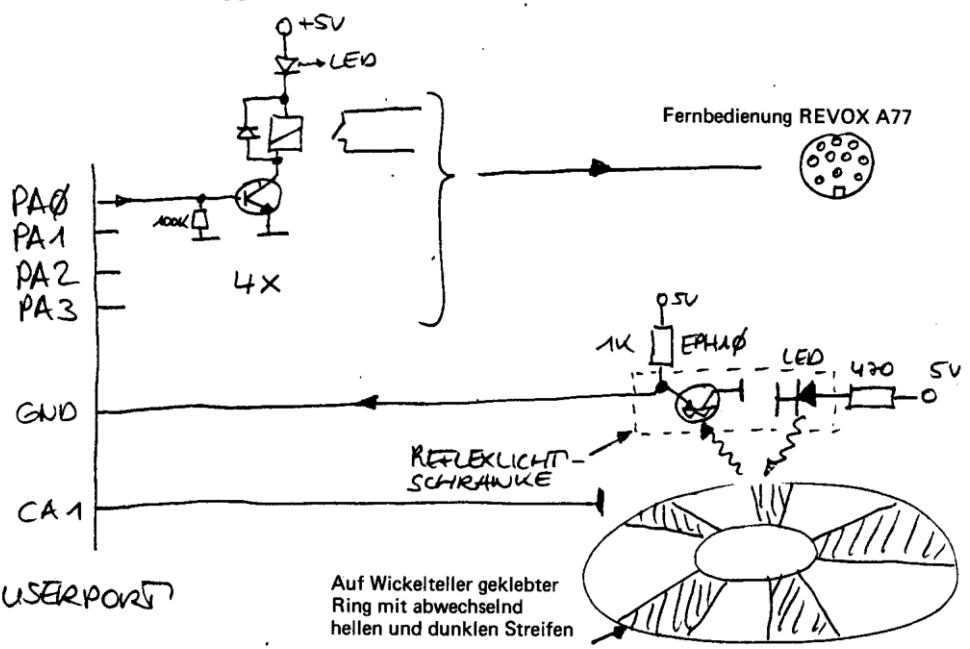
Der vorliegende Beitrag beschreibt die Möglichkeit, ein fernsteuerbares Tonbandgerät (z.B. Revox A77) von einem CBM unter Programmkontrolle zu steuern. Da Programm ist dabei nur ein Grundbaustein. Es läßt sich soweit erweitern, daß Titel automatisch gesucht werden können bzw. eine vorprogrammierte Reihenfolge von Musiktiteln vom Tonbandgerät abgerufen werden kann. Dies wird dadurch ermöglicht, daß der CBM-Rechner elektronisch in einer Interruptroutine über die jeweile Bandposition Buch führt. Hierzu muß allerdings an einem Bandteller im Tonbandgerät eine Lichtschranke angebracht werden, so daß bei laufendem Band dem Rechner einzelne Zählimpulse übermittelt werden können. Die weiterhin notwendige Hardware zeigt Abb. 1.

Über den Userport werden einzelne Relais für schnellen Vor- und Rücklauf, Play und Stop angesteuert. Von der Lichtschranke an einem der beiden Bandteller kommt eine Impulsleitung, die über den Eingang CA1 jeweils einen Interrupt auslöst. Um eine möglichst gute Ortsauflösung zu erhalten, sollte die Lichtschranke pro Bandtellerumdrehung mehrere Impulse liefern. Die Interruptroutine sorgt nun dafür, daß die beiden Speicherzellen 1000 und 1001 des Rechners zu jeder Zeit die aktuelle Bandposition wiedergeben. Es wird aufwärts oder abwärts gezählt, je nachdem ob vorher ein Vor-, Rücklauf oder PLAY-Kommando übermittelt wurde. Damit erübrigt sich eine zusätzliche Abfrage über die Wickelrichtung des Bandtellers. Wichtig ist, daß der Bandzählerstand im Rechner wegen der Interruptauslösung immer aktualisiert wird, unabhängig davon, was das im Vordergrund ablaufende BASIC-Programm ausführt. Der Bandzähler kann jederzeit z.B. mit der BASICzeile 180 zur Anzeige gebracht werden.

Mit RUN 1000 wird zunächst das kleine Interruptprogramm in den 2. Kassettenbuffer geladen (Zeilen 1000-1030). Zeile 1040 stellt den internen Zähler auf 0 und legt die Laufrichtung fest (hochzählen). Die Zeilen 1050 und 1070 sind insofern interessant, als hiermit von der BASIC-Ebene der Interruptvektor umgeschaltet wird. Dies wird normalerweise in Maschinenspracheebene durchgeführt, nach einem SEI-Befehl. Da im Normalzustand Interrupts im CBM nur vom Timer 1

des VIA 6522 ausgelöst werden, werden in der Zeile 1050 keine Interrupts mehr vom 6522 zugelassen, indem das entsprechende Interrupt-Enableregister mit 0 geladen wird. Dann kann ungestört von einem Interrupt in BASICzeile 1060 der Interruptvektor IRQ2 auf die Bandzählroutine umgeleitet werden (03BD). Die Zeile 1070 enabled im VIA 6522 dann wieder den normalen Timer 1-Interrupt und den CA1-Interrupt. Das in den DATA-Statements enthaltene kleine Maschinenprogramm testet zunächst, ob der Interrupt vom CA1-Eingang ausgelöst wurde. Nur dann wird abhängig von der Speicherzelle 03EA (1002) aufwärts oder abwärts gezählt. Falls der Interrupt nicht durch CA1 ausgelöst wurde, sondern vom Timer 1, wird zu der normalen Interruptroutine gesprungen.

Nach der Initialisierung wird das eigentliche BASIC-Vordergrundprogramm angesprungen. Dies erwartet eine Kommandoeingabe von der Tastatur und steuert das Tonbandgerät über den Userport entsprechend. Außerdem wird fortlaufend die aktuelle Bandposition angezeigt. Dieser Programmteil kann und sollte, wie anfangs angedeutet, erweitert werden, um z.B. eine programmierte Musiktitelfolge zu ermöglichen. Insofern soll das vorliegende Programm lediglich als Anregung zu eigener Softwareerstellung geben.




```
100 rem revox a77 steuerprogramm
110 print"comando > ,<,p,s,r
120 poke59459,255:rem userport = output
130 get a$
140 ifa$=">"then200:rem forwind
150 ifa$="<"then210:rem rewind
160 ifa$="s"then220:rem stop
170 ifa$="p"then230:rem play
```

```
180 ifa$="r"then240:rem reset counter
190 goto130
200 poke59471,1:fori=1to100:next:poke59471,0:poke1002,0:goto130
210 poke59471,2:fori=1to100:next:poke59471,0:poke1002,1:goto130
220 poke59471,4:fori=1to100:next:poke59471,0:goto130
230 poke59471,8:fori=1to100:next:poke59471,0:poke1002,0:goto130
240 poke1000,0:poke1001,0:goto130
999 rem-----------------------------------------------------------------
1000 data173,77,232,74,74,144,33,173,65,232,173,234,3,208,11
1010 data238,232,3,208,3,238,233,3,76,228,230
1020 data173,232,3,208,3,206,233,3,206,232,3,76,228,230,76,46,230
1030 fori=957to999:reada:pokei,a:next
1040 poke1002,0:rem  zaehlrichtung
1050 poke59470,0:rem irq disable
1060 poke145,3:poke144,189:rem irq2 auf zaehlroutine
1070 poke59470,130:rem t1 & ca1 irq enable
1080 stop
1085 rem-----------------------------------------------------------------
1090 poke59470,0:poke145,230:poke144,46:poke59470,128:rem reset irq vect

.-  03bd  ad 4d e8    lda  e84d
.-  03c0  4a          lsr  .a
.-  03c1  4a          lsr  .a
.-  03c2  90 21       bcc  03e5
.-  03c4  ad 41 e8    lda  e841
.-  03c7  ad ea 03    lda  03ea
.-  03ca  d0 0b       bne  03d7
.-  03cc  ee e8 03    inc  03e8
.-  03cf  d0 03       bne  03d4
.-  03d1  ee e9 03    inc  03e9
.-  03d4  4c e4 e6    jmp  e6e4
.-  03d7  ad e8 03    lda  03e8
.-  03da  d0 03       bne  03df
.-  03dc  ce e9 03    dec  03e9
.-  03df  ce e8 03    dec  03e8
.-  03e2  4c e4 e6    jmp  e6e4
.-  03e5  4c 2e e6    jmp  e62e
```

#

Aus der Branche - Produkte

Die Commodore Büromaschinen GmbH hat ihr Schulungsprogramm für das Winterhalbjahr 1982/83 erweitert. Es werden folgende Kurse angeboten: Commodore BASIC für Anfänger/für Fortgeschrittene/Floppy/Assembler für Anfänger/für Fortgeschrittene/IEEE/FORTRAN/APL/Strukturiertes Programmieren/Textverarbeitung/Finanzbuchhaltung. Adresse: Lyoner Str. 38 in 6000 Frankfurt 71, Herr Berlipp, Tel.: 0611-6638-182.

Das EXBASIC Level II für Commodore 2000/4000/8000 (8K EPROM) von Andreas Dripke im Vertrieb des Verlages Interface Age ist durch drei 'Softmodule Packs' wesentlich erweitert worden, so daß man wohl sagen kann, daß mit ihnen CBM-Rechner die größte zusammenhängende Erweiterung an Systemroutinen erhalten. Dieses Softwarepaket traf kurz vor Redaktionsschluß ein und konnte mit seinen vielfältigen Leistungsmöglichkeiten noch nicht im einzelnen ausgetestet werden, es ist gleichwohl imponierend: PAC1 bietet Scroll des Programm-Listings vorwärts und rückwärts, ein erweitertes Renumber, Replace von Textausdrücken, Programmver- und Entschlüsselung, Spoolbetrieb u.a.m und ein Paket zur 1/4 Grafik einschl. Bildschirmfenstern. - Das PAC2 bedeckt vorwiegend mathematische Funktionen, mehrfach genaue Arithmetik, Matrizenoperationen, verdichtete Abspeicherung von Variablen und einen Quicksort, den wohl am schnellsten ausführenden Sortieralgorrithmus. - PAC3 enthält erweiterte Funktionen bei der Stringeingabe, Stringersetzung, Formatierung von Ausgabestrings (erweitertes Print Using) und Datensicherung. Hier ist auch das Turbo-Programm enthalten, das den Diskettenzugriff wesentlich beschleuningt, ebenso das beschleunigte Handling der Datasette. Allein für die letztgenannten Dienstleistungen werden in Anzeigen anderswo Preise verlangt, die den Preis aller drei Pacs zusammen übersteigen. Ohne Verbindlichkeit: Jedes Pac kostet DM 88 plus ggfs. DM 56 für den Datenträger. Dabei ist jeweils ein 75-95-seitiges Anleitungsbuch enthalten, das auch das ggfs. nachzutippende BASIC-Programm für jedes einzelne Modul enthält. Die Module können in beliebiger Zusammensetzung an das EXBASIC II angehängt werden, das in jedem Fall Voraussetzung für den Betrieb ist. Info/Bezug: Interface Age Verlag GmbH, Vohburgerstr. 1, 8000 München 21.

Ein Fig-FORTH für alle Commodore-Rechner mit 32 KB RAM wird jetzt auch von der phs/SLS in Hannover geliefert. Es kam bei Redaktionsschluß hier an und konnte sofort in Betrieb genommen werden. Es erlaubt die Abarbeitung unter FORTH im Tischrechnerbetrieb und die Kompilierung aus dem mitgelieferten Editor heraus. Quelltext kann auf Diskette abgespeichert und von dort zurückgeladen werden, für eine Editierung oder für eine Kompilierung. Quelltext kann aber auch direkt von der Diskette her in das Vokabular hineinkompiliert werden. Es sind etwa 400 FORTH-Befehle bereits implementiert, darunter auch alle Gleitkommaoperationen, zu denen das BASIC der Commodore-Rechner fähig ist. Enthalten sind auch die von dort her bekannten höheren mathematischen Funktionen wie Sinus, LOG usw.. Die Sprache kann damit als gut ausgebaut bezeichnet werden. Stringoperationen sind in der hier vorliegenden Version noch nicht implementiert, sie können nach der Vorlage in Heft 25 jedoch leicht nachgetippt werden.

Ein BASIC-Compiler für CBM ist für November 1982 angekündigt. Die Gesellschaft für mathemat. Beratung & Software Dr. August C. Quint GmbH, Kaiser-Friedrich-Ring 55 in 6200 Wiesbaden bietet demnächst einen Compiler an, der aus einem BASIC-Programm diverse Files in Assembler-Quelltext generiert, in den zum Zwecke der Optimierung noch hineineditiert werden kann. Angestrebt sind im weiteren Verlauf Code-Optimierungen, Integerartimetik und Dezimalarithmetik. Zum Compiler gehört ein etwa 150-seitiges Handbuch.

Mikroprozessorplatine 6504.1 der Firma Michael Nowak (mino). Der Herausgeber erhielt den Prospekt für eine Karte, die in verschiedenen Einsatzfällen von Interesse sein kann: CPU 6504 (sie kann 4 KB adressieren), eine VIA 6502, 1 serieller Baustein 6551 ACIA mit eigenem Quarz, 2K RAM, 2K, 4K oder 8K EPROM Steckmöglichkeit (letztere werden nur teilweise ausgeschöpft). Daneben eine Lochrasterfläche für freie Verdrahtung. Daten-, Adreßbus- und I/O-Leitungen sind auf eine 64polige Steckverbindung VG 41612B herausgeführt. Die Karte sollte damit für Stand Alone-Systeme geeignet sein, insbesondere auch als Schnittstellenumsetzer (serielles Interface).

Speicherprogrammierbares Steuerungssystem LOGREG = Prozeßtechnik mit Microcomputern (CBM). Es gibt eine Vielzahl von speicherprogrammierbaren Steuerungen. Für den Einstieg in diese Technologie (fußend auch auf der entspr. Artikelserie in dieser Zeitschrift) wurde LOREG als ein Lern- und Testsystem entwickelt, und zwar für CBM mit Floppy und Drucker. Es können Logikprogramme ohne (und auch mit) Peripherie erstellt und ausgetestet werden. Damit ist LOREG auch für Ausbildungszwecke in Schule und Betrieb interessant. SPS-Programme werden im bildschirm- und zeilenorientierten Editor des CBM eingegeben und mit RUN compiliert. Es wird nach Funktionsplan programmiert. Funktionsblöcke können einzeln ausgetestet werden. Programme können im Eingabe- und im SPS-Format gelistet werden. Es gibt auch ein LOREG E/A-Magazin, das über den Userport des CBM angesprochen wird. Info/Bezug: Dipl-Ing. Wulf Dietmar Hein, Kollenrodtstr. 17, 3000 Hannover 1, Tel.: 0511 - 62 80 80.

Neue Komponenten zu Rockwell-Systemen. Nach den hier vorliegenden Pressemitteilungen sind folgende Module für Systeme mit dem RM 65-Bus jetzt lieferbar: Analog I/O-Modul RM65-5303E mit 16 Eingängen (bzw. 8 differentiellen) und 2 Ausgängen, 12 Bit Auflösung, 10 Bit Genauigkeit. Der RM65-5101 ist ein intelligenter Floppy Disk Controller für 5 oder 8 Zoll-Laufwerke (bis zu vier), einfache und doppelte Schreibdichte, ein- oder doppelseitig. Ein auf der Europaplatine enthaltenes ROM enthält die Unterprogramme für die Ansteuerung der Disketten. Das Modul kann zur Ansteuerung der meisten gebräuchlichen Laufwerke benutzt werden. Es unterstützt den interruptgetriebenen oder parallel abgefragten Betrieb. Mit Schaltern erfolgt die Zuweisung zu Speicherbanken, der Festwertspeicher kann deaktiviert werden. Ebenfalls neu ist der Direct Memory Accesss Controller (DMAC, RM65-5104E), der Datentransporte zwischen I/O und Memory mit bis zu 500 Kilobyte/sek durchführen kann. Auch Wechselbetrieb mit der CPU ist möglich.

Rockwell International hält Seminare zum Einsatz von Board-Level-Mikrokomputern am 7., 14., bzw. 15 Dezember 1982 in Stuttgart, Wien bzw. Zürich ab.

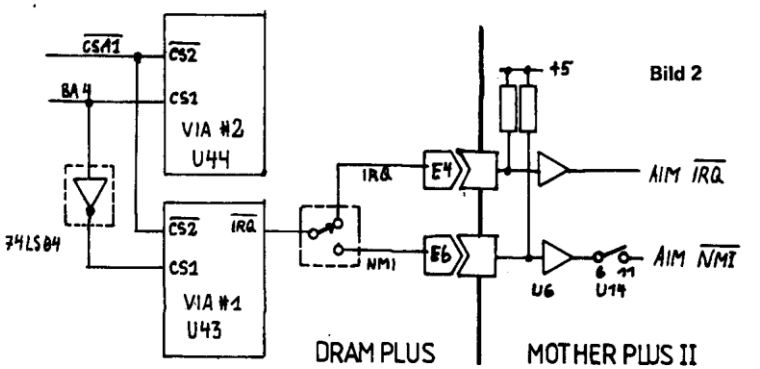
Bei der Firma System-Kontakt in Bad Friedrichshall u.a. Distributor für Rockwell-Produkte, ist Dipl.—ing Franz Leitl als Nachfolger des Geschäftsführers Dr. Jörg Langer eingetreten.

Motorola's Marketing Gruppe für Mikroprozessoren (Vertrieb, technische Unterstützung, Training) ist umgezogen nach Arabellastraße 17, Postfach 81 06 20 in 8000 München 80. Tel.: 089-92 72-0. — Motorola und Valvo/Philips veranstalteten am 6.Okt. 1982 in Frankfurts Plaza ein von etwa 120 Personen besuchtes **Anwenderforum für die 68000** mit verschiedenen Vorträgen von Anwendern. Am interessantesten waren wohl die Ausblicke auf die weitere Entwicklung der Familie 68000 (CPU-Versionen und Peripherie). So soll z.B. bis Ende 1982 die 10 MHz-Version L10 lieferbar sein und die L12 als Prototyp vorliegen. Die 68010 ist eine Maschine mit virtueller Speicheransprache für Mehrbenutzer. Ab Ende 1983 soll die Version 68020 in Mustern lieferbar sein, die mit 20 MHz arbeitet, 32 Pin Adreß- und 32 Pin Datenbus hat, 32 Bit-Befehle, auch mit der Adressierungsart bis 32 Bit Offset zum Programmzähler. Sie hat Befehle zur direkten BCD zu ASCII-Umwandlung, ist co-prozessorfähig (68881 Floating Point Prozessor), hat ein Hash memory und kann, ebenso wie die 68010 bei einem Bus-Error aus dem gestackten Microcode den letzten Befehl wiederholen. Sie kann ferner Block Moves ausführen. — Wenn man sich diese angestrebten Spezifikationen vor Augen hält und die weitere Entwicklung, daß größere Speicher weniger Platz benötigen und noch immer billiger werden, so sieht man, daß die Leistungsfähigkeit der Mikros schnell in weitere Bereiche fortschreitet. #

Wolfgang Radeloff, 2080 Pinneberg

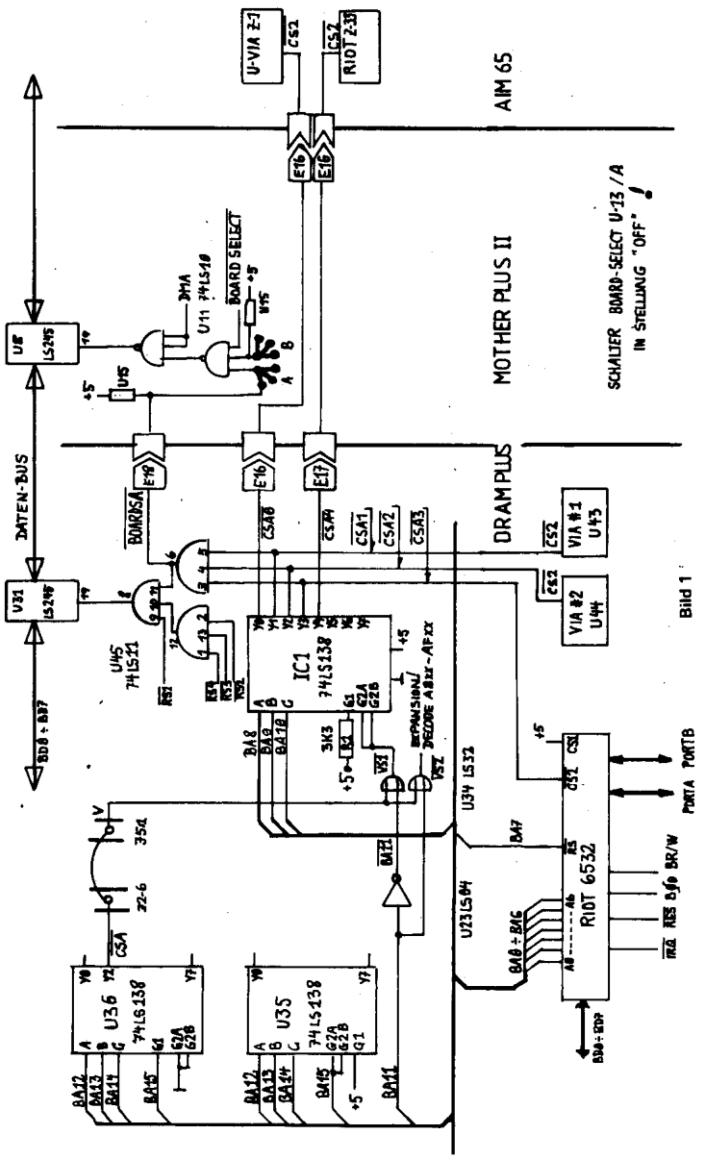
Decodierung des Axxx-Bereiches im AIM 65

Betreiber eines AIM mit der Speichererweiterung DRAM PLUS und der Busplatine MOTHER PLUS II können mit minimalem Aufwand den Adressenbereich Axxx besser decodieren. Die beiden VIA's des DRAM+ und weitere periphere Schaltungen können mit der Schaltung nach Bild 1 im genannten Bereich betrieben werden. Benötigt wird lediglich ein 3zu8 Encoder 74LS138 und ein Widerstand 3K3.



Der Umbau läßt sich leicht der Schaltung entnehmen. Außer der Installation von IC-1 sind die Anschlüsse an U-5 teilweise umzulegen. Das so freiwerdende 3fach-UND wird mit den Signalen CSA1, CSA2 und CSA3 belegt und erzeugt das Enable-Signal für die Datenbustreiber U-31 und U-8. Der Board-Select-Schalter für den A-Bereich auf der Busplatine bleibt in Stellung OFF. Das Signal BOARDS A schaltet für die Daten den Weg zum Prozessor selber frei.

Die Programmentwicklung für den EMUF-Einplatinen-Computer kann durch einen zusätzlich installierten RIOT-Chip unterstützt werden. Bild 2 zeigt eine Decodierung der beiden DRAM+ -VIAs im Bereich A1xx mit Hilfe eines der nicht benutzten Inverter auf dem Board. Die Basisadressen sind dann A100 und A110. - Wird das VIDEO PLUS II betrieben, sollte bei dieser Gelegenheit der Cold-Reset-Taster eingebaut werden (siehe MICRO MAG Nr. 10, Jg. 1979). #



Bernhard Kokula, 6700 Ludwigshafen

Lesen typ-verschiedener EPROMs

Auf dem 24-poligen Steckplatz der für einen bestimmten EPROM-Typ decodiert ist, kann man durchaus ein typverschiedenes EPROM auslesen. Man findet dann aber häufig die Information in einem versetzten Adressenbereich Die Zusammenhänge zeigt nebenstehende nützliche Tabelle. In der ersten Spalte der Kästchen steht jeweils 1KB-weise die Speicheradresse des lesenden Computers. Die zweite Spalte zeigt, welches x-te 1K-Segment des EPROMs man dort ausliest.

#

EPROM: TYP → / AUF PLATZ ↓	2716	2532	2732(A)
2716	=	1. ← 1. k 2. ← 2.	1. ← 3. k 2. ← 4.
2532	1. ← 1. k 2. ← 2. 3. — 4. —	=	1. ← 3. k 2. ← 4. 3. — 4. —
2732(A)	1. — 2. — 3. ← 1. k 4. ← 2.	1. — 1. k 2. — 2. 3. ← 1. k 4. ← 2.	=

Editorial

Die Herausgabe einer Zeitschrift 8xxx MICRO MAG wurde in Heft 25 als Projekt angesprochen. Es wurde einstweilen angehalten. Dabei spielten Hardwareschwierigkeiten (Reparaturen) mit, die die Zeitpläne zerwarfen und die bekannten Entwicklungen in Wirtschaft und Politik, deren Auswirkungen noch einige Zeit beobachtet werden sollten. Es bleibt auch zu beobachten, ob für die Prozessorfamilien nicht letztlich die gleichen Lösungswege (Algorithmen) darzustellen sind. Als prozessorspezifisch sind die Befehlssätze und damit die jeweiligen Assemblersprachen anzusehen, ebenso die Eigenschaften der Interfacebausteine, die in einer Programmierung zu berücksichtigen sind.

Eigentlich nur für diesen Bereich interessiert eine spezifische Fachzeitschrift. Und da ist nun festzustellen, daß auf uns eine große Zahl von Computern mit gemixter Hardware zukommt: 6502 mit Z80, mit 6809 oder mit 68000. Damit wird sich das Interesse der Betreiber auf mehrere prozessorspezifische Sektoren richten müssen. Auch in einer solchen Umgebung kann man den für einen Prozessor gelisteten Algorithmus einer Dienstleistung für einen anderen anpassen, wenn man die unterschiedliche Leistungsfähigkeit und die nötige Verwaltung der Maschinenregister berücksichtigt.

Die Hantierung der vielen Computer kann nicht Gegenstand einer Fachzeitschrift sein, hier ist auf die jeweiligen Handbücher hinzuweisen.

Das gemeinsame Band für alle Computerbetreiber bilden jedoch dann wieder die höheren Sprachen. BASIC ist eine Selbstverständlichkeit für jeden Computer. Diese Sprache darf mit ihren Möglichkeiten als gut dokumentiert gelten, gleichwohl treten immer wieder interessante Lösungen auf, auch in Verbindung mit Assemblerprogrammen, die darzustellen sind. Die Leser seien nach wie vor zu solchen Beiträgen ermuntert, auch zu solchen in Assemblersprache.

Von PASCAL muß man vermuten, daß diese Sprache auf Mikrocomputern nur wenig benutzt wird. So sind hier in den vergangenen Jahren keine Beiträge unter PASCAL vorgelegt worden.

Demgegenüber ist festzustellen, daß FORTH schon jetzt eine hohe Akzeptanz findet. Wie der Inhalt dieses Heftes wieder zeigt, ist es ein sehr modulares Werkzeug für die Softwareentwicklung, und es gibt es für AIM, CBM (demnächst wohl auch für den JUNIOR), für 68er Systeme und natürlich auch für Z80 u.a.m..

LISP, im Zusammenhang mit 'artificial intelligence' (AI) ist jetzt auch für den 6502 verfügbar, und zwar für den Apple und für CBM. Auf CBM wurde dem Herausgeber kürzlich eine sehr leistungsfähige Implementierung des Softwarebüros H-G. Lange in Berlin vorgeführt, die demnächst auch hier eingesetzt und besprochen werden soll. LISP gibt es natürlich auch unter dem Betriebssystem CP/M.

Der Gebrauch von Sprachen wie FORTH und LISP ist für unseren Sprachraum in Büchern und in anderen Zeitschriften noch viel zu wenig abgehandelt worden, so daß hier Aufgaben für eine anpassungsfähige Zeitschrift wie diese liegen, wenn sie eine Mehrzahl von Computern betreuen will. Und das betrifft auch z.B. die Beschreibung einfacher Datenbanksysteme, Taskverwaltung u.a.m.. Es gibt damit einige Gründe, nicht unbedingt den Weg der prozessorspezifischen Fachzeitschriften weiterzugehen, sondern im vorhandenen Rahmen die allgemeinen Instrumente darzustellen.

Wesentlich ist natürlich auch die Erwartungshaltung der Leser. Ihre Meinung und ihre Anregungen zu darzustellenden Themen sind hier stets willkommen.

Den Heften im Direktversand ab hier ist ein Prospekt für das 65xx MICRO MAG beigefügt. Wenn er auch ein nützliches Gesamt-Inhaltsverzeichnis darstellt, so werden die Leser doch um empfehlende Weitergabe in ihrem Bekanntenkreis gebeten. #

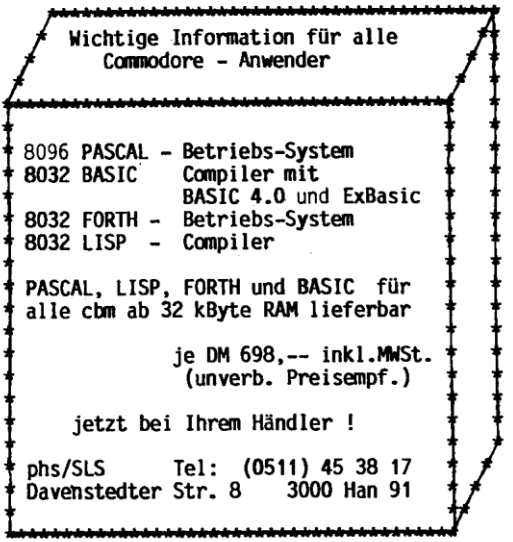
Kleinanzeigen

Bufferplatine für Daten und Adressen des AIM 65 lieferbar. Einbau ohne zu löten zwischen CPU 6502 und CPU-Fassung. Für alle 6502-Systeme verwendbar (KIM, SYM usw.). In Industriequalität m. Schaltplan/Beschreibung DM 21,80 inkl. Porto: B. Kokula, Wredestr. 17, 6700 Ludwigshafen.

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

LISP Compiler/Interpreter für CBM 3032, 4032, 8032 inkl. Macro-Erweiterung, Editor und deutscher Anleitung. Info bei H.-G. Lange; Goltzstraße 18, 1000 Berlin 30.

AIM Screen Editor, voll Cursor- und bildschirmorientiert Groß und Kleinschreibung, Autorepeat, Scroll up/down, paging, Tabulator, Block-Move/Copy/Kill, Randausgleich, voll kompatibel mit AIM-Editor etc. 2K/4K-EPROM, DM 90,-/180,-. Handbuch DM 10,-. Info gegen Freiumschlag: Kurt Peter, Kölnerstr. 6, 6053 Obertshausen, Tel.: 06104 - 717 89. #



Wir liefern Compiler für CBM

PASCAL
Vollcompiler nach ISO-Normierung
weitgehend UCSD-kompatibel
voller Jensen/Wirth Standard
11-stellige Arithmetik
PEEK/POKE
Vietelpunktgraphik über PLOT (x,y,boolean)
virtuelles RAM bis 1 Megabyre mit cbm 8050
Dateidirektzugriff
jegliche Peripherie wie gewohnt ansprechbar (IN- und OUT !)
lauffähig mit 32 kByte RAM und Doppelfloppy
Sonderversion mit 64 kByte für 8096 lieferbar

BASIC
Vollcompiler mit BASIC 4.0 und ExBasic Level II
automatische Programmoptimierung
darüberhinaus erweiterte Standard-BASIC-Funktionen
dynamische RAM-Verwaltung
verschiebbarer Code
Linker für Assemblerprogramme

FORTH
vollständiges FORTH-Betriebssystem
Diskhandling
Macro-Assembler
398 Befehle bereits implementiert
Linker für Assemblerprogramme

Bei allen Compilern handelt es sich um ausgereifte Systeme, die gerade für den kommerziellen Nutzer alle Möglichkeiten der Commodore-Geräte ohne Einschränkungen nutzbar lassen (auch fremde Hard- oder Softwareerweiterungen).

Die Compiler werden auf Diskette geliefert, deshalb bei allen Bestellungen unbedingt Rechner- und Floppytype angeben!

Unsere Preise- Jeder Compiler DM 698,-- inkl. 13% MWSt. bei Kauf
DM 113,-- inkl. 13% MWSt. bei 14 Tagen Miete
(nur PASCAL und BASIC I)

phs/SLS EDV-Beratung

Davenstedter Str. 8 **3000 Hannover 91** **Tel.: 0511 - 45 38 17**

Im INTERFACE AGE Vertrieb Produkte der Unternehmensberatung A. DRIPKE:

SOFTMODULE™

Die ideale Ergänzung zu EXBASIC LEVEL II

Die Leistungsfähigkeit von EXBASIC LEVEL II ist enorm - und trotzdem noch um ein Vielfaches steigerbar mit SOFTMODULEN. Ein SOFTMODUL wird einfach von Diskette oder Kassette in den Computerspeicher eingeladen und hängt sich automatisch an EXBASIC LEVEL II an. Die neuen Befehle des SOFTMODULS stehen dann zusätzlich zur Verfügung. Es können beliebig viele SOFTMODULE gleichzeitig aktiv sein. Auf diese Weise können Sie Ihren Commodore auf den Anwendungsgebieten stark machen, auf den Sie ihn einsetzen. Es gibt keine andere Sprache im Mikrocomputerbereich als EXBASIC LEVEL II, die Sie so flexibel an Ihre Bedürfnisse anpassen können.

Jedes einzelne SOFTMODUL-Pac bietet eine geradezu unglaubliche Fülle an neuen Funktionen, die Ihnen durch ihre Leistungsfähigkeit das Programmieren um ein Vielfaches einfacher und komfortabler machen. Für Programmteile, die zu programmieren Sie früher Tage oder Wochen gekostet hat, nehmen Sie jetzt ein einziges SOFTMODUL, das dieselbe Funktion mit einer Anweisung und sogar besser erfüllt. SOFTMODULE zu EXBASIC LEVEL II sind für jeden, der seinen Computer ernsthaft programmiert ebenso unerläßlich wie EXBASIC LEVEL II selbst.

Jedes SOFTMODUL-Pac beinhaltet eine leicht verständliche Einführung in die grundsätzliche Benutzung von SOFTMODULEN, genaue Erklärungen aller Befehle und Funktionen, die die einzelnen SOFTMODULE zur Verfügung stellen mit zahlreichen Beispielen sowie die SOFTMODULE selbst in Form von Basic-Listings. Der Preis für ein solches SOFTMODUL-Pac beträgt DM 88,- inkl. MwSt.*

Vor der ersten Benutzung müssen Sie die SOFTMODULE in den Computer eingeben, um sie zur weiteren Verwendung auf Kassette oder Diskette abzuspeichern. Wenn Sie sich das Eintippen ersparen möchten, so können Sie sich zusätzlich fertige Kassetten oder Disketten zulegen. Eine solche Diskette oder Kassette mit den SOFTMODULEN des jeweiligen Pacs kommt auf zusätzlich DM 56,- inkl. MwSt.* Geben Sie bei Bestellung einer Diskette bitte den Typ Ihrer Floppystation an.

* unverbindliche Preisempfehlung.

Name: _____ Str./Nr.: _____

Plz/Ort: _____ Ich bestelle hiermit:

... Exemplar(e) EXBASIC LEVEL II (Bitte geben Sie Ihren Computertyp an!)

... Exempl. SOFTMODUL-Pac Nummer ... (1/2/3) mit/ohne Kassette/Diskette

... Exempl. SOFTMODUL-Pac Nummer ... (1/2/3) mit/ohne Kassette/Diskette

... Exempl. Kombi-Pac mit allen drei SOFTMODUL-Pacs mit/ohne Kass./Disk.

... Exempl. PHS-EXBASIC-COMPILER für DM 698,- inkl. MwSt. Voll kompatibler Compiler für Commodore Basic und EXBASIC LEVEL II. Gerne senden wir Ihnen auch ausführliche Informationen über unsere Compiler für Commodore.

Bitte angeben: Computertyp _ _____, Floppydisk (falls vorhanden) _____

Zahlungsart (Bitte ankreuzen): ... Verrechnungsscheck liegt bei, ... per Nachnahme

Einsenden: INTERFACE AGE, Vohburgerstr. 1, D-8000 München 21, Tel. 089/5806702

Datum: _____ Unterschrift: _____

Pac 1: Schwerpunkte Graphik und Hilfsfunktionen

- GRAPHIC 1. Komplexes Modul für Viertelcursorgraphik mit 13 neuen Befehlen zum Zeichnen von Funktionen. SCALE, XAXIS, YAXIS, PENUP, PENDOWN, PEN, POINT, POS, PLOT, MOVE, DRAW, IMOVE, IDRAW.
- GRAPHIC 2. Komplexes Modul zur Definition und Manipulation von Bildschirmfenstern. Zeile/Spalte/Fenster invertieren, Fensterbereich rollen rechts/links und hoch/runter, Abspeichern auf Diskette u.v.a.m. Insgesamt 10 verschiedene Befehle.
- SCROLL. Das Programmlisting wird zu einem Band zusammengefaßt und kann über den Bildschirm vor und zurück gerollt werden. Ständiges manuelles Listen "von - bis" entfällt. Steuerung mit 'Cursor hoch/runter'.
- RENUMBER. Blockweises Umnumerieren eines Programmteiles.
- REPLACE. Wie "Search & Replace" bei Textprogrammen. Es werden beliebige Programmteile automatisch durch neue ersetzt (Befehle, Variablen, Texte etc.).
- CODE/DECODE. Verschlüsseln von Programmen mit einem Codewort aus bis zu 255 Buchstaben auf Diskette oder Kassette. Wer das Codewort nicht kennt, kann das Programm nicht benutzen oder listen.
- SPOOLING. Erlaubt die Ausgabe von Programmen und Dateien von Diskette auf Drucker, während gleichzeitig das Zentralgerät benutzt werden kann.
- UNNEW. Macht versehentliches NEW wieder rückgängig.

Pac 2: Schwerpunkte Mathematik und Matrixmanipulation

- MEHRFACHGENAUE ARITHMETIK. Alle vier Grundrechenarten können mit bis zu 255 Stellen Genauigkeit durchgeführt werden. Komplexes Modul.
- PACK/UNPACK. Speichert Zahlen platzsparend im 5-Byte oder 2-Byte Format in Stringvariablen ab. Ideal für große Datenmengen (auch z.B. auf Diskette).
- MAX/MIN. Sucht das Maximum bzw. Minimum aus einem Variablenfeld beliebiger Dimension in Sekundenschnelle. Bei einer Matrix mit optionaler Spaltenangabe.
- QSORT. Sortiert ein- oder zweidimensionale Variablenfelder (numerisch real oder integer und Strings) nach schnellstem bekannten Verfahren (Quicksortalgorithmus).
- MATRIZENRECHNUNG. Komplexes Modul mit allen Grundfunktionen der Matrizenrechnung. ADD, SUB, MUL, INV, DET und viele weitere Möglichkeiten.

Pac 3: Schwerpunkte Stringmanipulation und Peripheriegeräte

- FORMAT. Formatierungsfunktion ähnlich PRINTUSING, mit Zuweisungsmöglichkeit an Variable. FORMAT kann absolut alles, was hierbei denkbar ist: String - und Zahlenformatierung, Exponentialdarstellung, deutsche Kaufmannsnorm u.v.a.m.
- MID$=. Innenstringmanipulation, erlaubt einen String an beliebiger Stelle direkt in einen anderen einzusetzen und erspart somit komplizierte RIGHT$-, LEFT$- und herkömmliche MID$-Befehle.
- CAPITAL/SMALL. Wandelt alle Zeichen eines Strings in Groß- bzw. Kleinbuchstaben um. Das ist immer dann wichtig, wenn Strings unabhängig von Groß-/Kleinschreibung miteinander verglichen werden sollen.
- GETSTR$#. Liest bis zu 255 Zeichen von Diskette unabhängig von irgendwelchen Trennungszeichen. Länge und Abbruchkriterium sind frei wählbar. Ideal zum kompakten Abspeichern von Daten auf Diskette.
- PLIST/SLIST/RLIST/FILE. Listet Programme und alle Dateiarten von Diskette auf Bildschirm oder Drucker, ohne das Programm im Speicher zu zerstören. FILE gibt eine Aufstellung aller aktiven OPEN-Kanäle mit Adressen.
- SECURE/UNSECURE. Schützt Programme und Dateien auf Diskette vor unbeabsichtigtem Löschen oder Überschreiben. Dadurch hohe Datensicherheit.
- DMOD. Ein mit DMOD auf Diskette gespeichertes Programm kann zwar ausgeführt, aber nicht gelistet, geändert oder sonstwie manipuliert werden. Das DMOD-Programm kann von jedem Computer auch ohne EXBASIC LEVEL II eingelesen werden.
- FAST TAPE 8000. Schnelle Kassettenoperationen mit fünffacher Geschwindigkeit für CBM 8000. Mit den Befehlen LOAD, SAVE, VERIFY und MERGE.
- HARDCOPY. Mit Angabe eines Bildschirmbereiches "von - bis" und Druckwegoptimierung.

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

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

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

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

Nachliefermöglichkeiten: Hefte 1-13 sollten als Buch nachbezogen werden (s. Anzeige unten). Solange Vorrat reicht, können auch noch einzelne Hefte der Nos. 7-13 geliefert werden. Hefte 14-26 sind unbeschränkt nachlieferbar zu DM 7,80/Stück + DM 2,50 je Sendung.

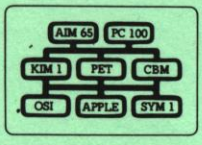
Private Besteller werden um Überweisung/Scheck (auch Auslandsschecks) zusammen mit der Bestellung gebeten. Konto Roland Löhr, Nr. 654 70-202 Postscheckamt Hamburg, BLZ 200 100 20.

Leser-Service des Herausgebers

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

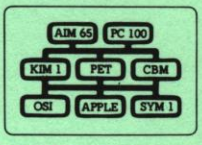
Thermokopf (Printerplatte) für AIM 65 und PC 100
mit Anleitung für den leichten Einbau. Auffrischung des Druckes — **DM 28,--**

Das Buch 1-6 des 65.. MICRO MAG



230 Seiten, DM 26,--

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

FORTH User's Guide
Rockwell-Handbuch für das Fig-FORTH des AIM 65. Mit der Erklärung des Befehlssatzes auch für andere FORTH-Betrieber geeignet. Ca. 300 S., engl. — DM 24,--

Cross-Assembler für MC6805 und 6809

lauffähig unter FORTH des AIM 65, erzeugt EPROM-fähigen Zielcode in beliebigem Speicherbereich. Komfortable Assembler, strukturiert, auch mit Labeln arbeitend, mit Warnings und Fehlermeldungen. Lieferbar ab Ende Oktober 1982. Handbuch DM 10,--, Cross6805-EPROM ($D000) DM 320,,-, Cross6809 ($C000-DFFF) EPROMs DM 380,-.

Alle Preise inkl. MWSt, zuzüglich DM 2,50/Sendung + ggfs. Nachnahme + DM 1,70