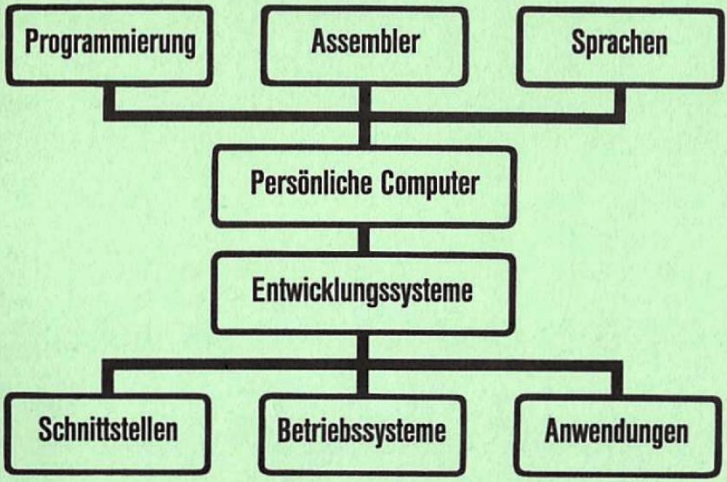
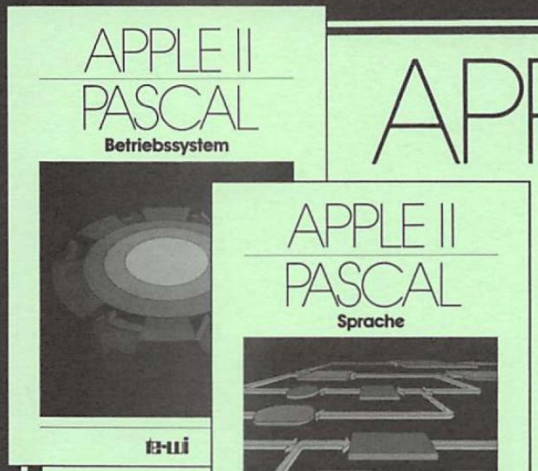
MICRO MAG

DM 9,50 Nr. 41 Februar 1985



Inhaltsverzeichnis

Programmierung der 65802/816 3
Dis- und Reassembler für alle 65xxx-CPU's 14
Konzepte der CPU's ' 320xx 30
Development Board DB 16000 von NS 32
Der LASER 3000 34
C-64 am IEEE 488 Bus - Erweiterungen 37
COPY 4 42
C-64 Disk-Monitor (2) 43
Dump to Apple 50
Verbesserte Disk-Befehle CBM 8032/4032 52
Bücher 41, 56
Editorial 56
Aus der Branche 57
DISMOS V1.0 58



APPLE II PASCAL

Betriebssystem und Sprache

Erstes deutsches Referenzwerk sämtlicher Befehle und Systemroutinen von Apple II Pascal – mit Addendum einschließlich Version Pascal 1.2!

Gültig für Apple II, II Plus, IIe einschließlich der 128K/80 Zeichen-Konfiguration.

Betriebssystem kommentiert ausführlich und in Deutsch Funktion und Benutzung der fast 60 Systemroutinen des Apple II Pascal Betriebssystems.

Sprache ist das vollständige, deutsche Referenzwerk der „Apple Pascal"-Programmiersprache mit u. a. Informationen über professionelle Pascal-Programmierung, Turtlegraphics, Programmbibliothek etc.

In Vorbereitung: Addendum Pascal 1.2, ein Zusatz zum Buch „Betriebssystem" für 1.2-Benutzer in Deutsch.

„Nach Unterlagen von Apple Deutschland hergestellt"

Apple II Betriebssystem,
272 Seiten, DM 49,–

Apple II Sprache,
216 Seiten, DM 39,–

Pascal 1.2 Addendum, etwa 100 Seiten,
DM 36,– (1. Quartal '85)

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

te-wi

Roland Löhr

Programmierung der 65802/816

Zur Übersicht

Nach der Implementierung eines entsprechenden Assemblers für die neuen CPU-Typen von GTE hat hier die weitere Erprobung und das Studium zweckmäßiger Programmiertechniken begonnen. Die dabei gemachten Beobachtungen sind interessant und sollen hier in einer losen Folge dargestellt werden. Man beachte auch die Hinweise in den Heften 38 und 40.

Benutzte neue Assembler-Direktiven

.A08 veranlaßt den Assembler, Direktoperanden für den Akku wie bisher als solche von 1 Byte Länge zu betrachten. Bei .A16 sollen solche Operanden 2 Byte lang sein. .X08 und .X16 sind entsprechende Anweisungen für die CPU-Register X und Y. .R08 und .R16 sind ein entsprechender zusammenfassender Ausdruck für alle CPU-Register.

Die Betriebsweise der CPU-Register während der Ausführungszeit wird durch die Status-Bits M und X bestimmt. Das geschieht mit den Befehlen REP # und SEP # (Reset und Set Bits im Statusregister P). Sie sind als Befehl für die CPU zu codieren, wo eine der vorstehenden Assembler-Direktiven benutzt wird, sonst ist das Programm mangels Gleichschrittes absturzgefährdet.

Beispiel 20: Einrichten eines Pointers in der Zero Page

In bisheriger Technik wurde ein Pointer durch nachstehende Befehlsabfolge eingerichtet: LDA #LOW, STA POINTER, LDA #HIGH, STA POINTER+1.

Stattdessen bringen wir jetzt den Akku mit .A16 und REP #$20 in den 16 Bit-Modus, laden ihn mit einem Direktwert und speichern ihn in den Pointer. Nach Ablauf des Beispieles 20 finden wir $FF in Zelle 20 und hex 34 in Zelle 21. Es werden also in diesem Modus auch 2 Bytes abgespeichert.

Beispiel 21: Direct Page Pointer beliebig im RAM

Das Direct Register (oder wie wir es auch nannten: Direct Page Register) gestattet es, die Direct Page für die verkürzte Adressierungsform Zero Page beliebig ins RAM zu legen, auch an 'krumme' Adressen. Im Beipiel legen wir die Direct Page ab Adresse hex 2001. Das geschieht mit dem Befehl TCD, der den Inhalt des Akku in dieses Register überträgt. Die spätere Ablage des Akku-Inhaltes mit STA $20 (in Zeile 13) erfolgt relativ zu dieser Basis. Tatsächlich finden wir in den Zellen hex 2021/2022 die Bytes FF und 34 (LOW/HIGH).

```
*******************************************************************************
EINRICHTEN EINES POINTERS IN DER ZERO PAGE
000000                  0001        *=$4000
004000  18              0002        CLC             ;SET EMULATION BIT=0 FOR NATIV
E MODE
004001  FB              0003        XCE             ;EXECUTE
004002                  0004        .A16            ;DIRECTIVE FUER ASSEMBLER: AKK
U 16 BIT
004002  C2 20           0005        REP #$20        ;UMSCHALTUNG DER CPU
004004  A9 FF 34        0006        LDA #$34FF      ;DIREKTOPERAND
004007  85 20           0007        STA $20         ;ABLAGE ZUR KONTROLLE
004009                  0008
004009  E2 20           0009        SEP #$20        ;CPU AUF 8 BIT ZURUECKHOLEN
00400B  38              0010        SEC             ;PREPARE
00400C  FB              0011        XCE             ;RESET TO 6502 MODE
00400D  4C 82 E1        0012        JMP $E182       ;COMIN
004010                  0013        .END
 ERRORS=0000

*******************************************************************************
BEISPIEL 21:DIRECT PAGE POINTER BELIEBIG IM RAM
000000                  0002 ;INITIALISIERUNG WIE ZUVOR
004004                  0009        .OPT LIS
```

```
004004  A9 01 20      0010          LDA #$2001
004007  5B             0011          TCD                 ;LADE DAS DIRECT PAGE REGISTER
 ZU $2001
00400B  A9 FF 34       0012          LDA #$34FF          ;DIREKTOPERAND
00400B  85 20          0013          STA $20             ;ABLAGE ZUR KONTROLLE
00400D                 0014 ;FOLGEBEFEHLE WIE GEHABT

*********************************************************************************
BEISPIEL 22: INKREMENTIERE POINTER
000000                 0002 ;INITIALISIERUNG WIE ZUVOR
004004                 0009          .OPT LIS
004004  0B             0010          PHD                 ;RETTE DIRECT PAGE REGISTER
004005  A9 01 20       0011          LDA #$2001
004008  5B             0012          TCD                 ;LADE DAS DIRECT PAGE REGISTER
 ZU $2001
004009  A9 FF 34       0013          LDA #$34FF          ;DIREKTOPERAND
00400C  85 20          0014          STA $20             ;ABLAGE ZUR KONTROLLE
00400E  E6 20          0015          INC $20             ;INKREMENTIERE DEN POINTER
004010  2B             0016          PLD                 ;HOLE ALTE DIRECT PAGE ZURUECK
004011                 0017 ;FOLGEBEFEHLE WIE GEHABT
```

Beipiel 22: Inkrementieren eines Pointers im RAM

Es kommt nun eine angenehme Überraschung: Wenn das M-Bit im Status gelöscht ist (Akku auf 16 Bit Breite eingerichtet), dann wirken auch die auf den Speicher gerichteten Befehle INC Pointer und DEC Pointer in dieser Breite. Im Beispiel wird der Pointer zunächst mit hex 34FF geladen. Es folgt in Zeile 15 ein INC Pointer, und wir finden dann ab Adresse $2021 die Bytes 00 und 35 zum Beweise, daß INC mit einem Befehl auf 2 Byte wirkt. Demgegenüber hatten wir früher die übliche Reihenfolge INC Pointer, BNE WEITER, INC Pointer+1, WEITER ....

Ein analoges Ergebnis stellt sich ein, wenn man den Befehl DEC Adresse gibt. Und ebenso wirken INC A und DEC A natürlich auf den Akku in 16 Bit Breite.

Wichtiger Hinweis zum Direct Page Register

Man tut bei Experimenten mit diesem Register gut daran, es vor einer Veränderung zu retten und es vor der Rückkehr in den Monitor wieder in den ursprünglichen Zustand zu bringen, damit der Monitor evtl. benutzte Pointer an der gewohnten Stelle wiederfindet. Dem dienen die Befehle PHD und PLD im Beispiel 22 - bringe dieses Register auf den Stack und hole den Inhalt von dort zurück.

Beispiel 23: Pointer wie Indexregister benutzt

Bereits bei der CMOS-CPU R65C02 wurde für die arithmetischen und logischen Befehle die neue Adressierungsart Befehl (indirekte Adresse) eingeführt. Sie gestattet es, einen Pointer wie ein Indexregister zu benutzen, ohne daß eines der Indexregister für die Bildung der effektiven Adresse eingeschaltet wird.

Wenn man nun zusätzlich die Befehle INC Pointer und DEC Pointer aus Beispiel 22 in Betracht zieht, so wird erkennbar, daß man eine große Zahl von Pointern im RAM ohne ohne Umstände wie ein Indexregister benutzen kann. Man beachte dabei, daß INC und DEC Pointer nur dann auf 16 Bit wirken, wenn der Akku im 16 Bit-Modus ist. Der Zustand des X-Bits im Status für die Registerbreite von X und Y wirkt nicht auf diese Befehle.

Wenn man den Akku nun indirekt über die Pointeradresse lädt, so werden 2 Byte geladen. Beim Überstreichen eines Datenfeldes muß man nun zweimal INC bzw. DEC Pointer geben, um auf das nächste Element eines Feldes aufzusetzen. Im Beispiel 23 führen wir eine dezimale Addition über 3 Worte=6 Byte aus, wobei ab Adresse FELD der Operand in absteigender Wertigkeit der dezimalen Bytes per Assembler (.DBY) abgelegt worden ist. Er soll verdoppelt und an alter Stelle wieder abgelegt werden.

In der Initialisierung laden wir wieder das Direct Page Register mit hex 2001 und den Pointer mit der Adresse FELD+4, damit er auf den niederwertigen Teil des Operanden weist. Die Variable ZAEHLR wird für 3 Schleifendurchläufe vorgeladen. Es wird der Dezimalmodus hereingerufen. Es ergibt sich:

Operand per Assembler 10 11 22 23 34 35
Verdoppelung per Programm 20 22 44 46 68 70

Wenn wir den Dezimalmodus (SED) fortlassen, so ergibt sich das ebenfalls richtige binäre Ergebnis:

20 22 44 46 68 6A

Die Programmierung kann im 16 Bit Modus also nach bekannten Prinzipien fortgesetzt werden. Das Beispiel 23 möge weiterhin zeigen: Mit den neuen Adressierungsarten ist es möglich, ohne Benutzung eines wirklichen Indexregisters eine 'indizierende' Adressierung über Hilfsregister im Speicher der Direct Page zu erreichen. Und da die Direct Page in der Speicherbank 0 beliebig eingerichtet werden kann, hat man im Endeffekt eine sehr große Zahl von Hilfsregistern. - Das Konzept dieser Memory indirekten Adressierung entspricht modernen CPU-Designs, wie wir sie z.B. in Heft 40 beim MC68020 erwähnten. In einer solchen Umgebung kann man mehrere Aufgaben oder Benutzer ohne komplizierte Umladevorgänge bedienen.

```
*******************************************************************************
BEISPIEL 23: ADDITION EINES FELDES INDIREKT (POINTER)
000000                0002 POINTR =$20              ;POINTER IN DER DIRECT PAGE
000000                0003        *=$5000           ;PARAMETERFELD
005000  10 11         0004 FELD   .DBY $1011,$2223,$3435 ;HOEHERWERTIGE BYTES LIN
KS
005002  22 23         0004
005004  34 35         0004
005006                0005 ;INITIALISIERUNG WIE ZUVOR
004005                0013        .OPT LIS
004005  A9 01 20      0014        LDA #$2001
004008  5B            0015        TCD               ;LADE DAS DIRECT PAGE REGISTER
 ZU $2001
004009  F8            0016        SED               ;DEZIMALMODUS RUFEN
00400A  A2 03         0017        LDX #3            ;INITIALISIERUNG EINES ZAEHLER
S
00400C  A9 04 50      0018        LDA #FELD+4       ;UEBERTRAGE ADRESSE NACH POINT
R
00400F  85 20         0019        STA POINTR        ;MIT 2 BYTE INITIALISIERT
004011  18            0020        CLC               ;ADDITIONSVORBEREITUNG
004012  B2 20         0021 ADLOOP LDA (POINTR)
004014  72 20         0022        ADC (POINTR)      ;VERDOPPELUNG
004016  92 20         0023        STA (POINTR)      ;ABLAGE AN ALTER STELLE
004018  C6 20         0024        DEC POINTR
00401A  C6 20         0025        DEC POINTR        ;ERNIEDRIGUNG UM 2 ADRESSEN
00401C  CA            0026        DEX
00401D  D0 F3         0027        BNE ADLOOP
00401F  D8            0028        CLD               ;BNIAERMODUS ZURUECK
004020                0029 ;FOLGEBEFEHLE WIE GEHABT
```

Beispiel 24: Pointeradressierung in eine andere Bank

Ein vergleichbares Beispiel wie in Nr. 23 wollen wir jetzt für eine Subtraktion geben. Es wird eine neue Adressierungsart des 65816 benutzt, nämlich die Schreibweise Befehl (Pointer,L). In diesem Fall hat der Pointer 3 Byte in folgender Anordnung von niederen zu höheren Adressen:

Adresse LOW Adresse HIGH Bankadresse

Das Beispiel 24 soll auch wieder ohne Zuhilfenahme von Indexregistern arbeiten. Wir gehen davon aus, daß die Operanden in FELD1 und FELD2 jeweils vollständig in einer Bank liegen, so daß bei der laufenden Veränderung der Pointeradressen nicht die Zweifelsfrage auftritt, ob auch die Bankadresse im Pointer verändert werden muß. FELD2 soll von FELD1 subtrahiert und das Ergebnis in FELD1 abgelegt werden. Wir benötigen dann 2 Pointer, nämlich POINT1 und POINT2. Zur Vereinfachung nehmen wir einmal an, daß die Pointer bei Programmbeginn bereits mit Adressen (3 Bytes) geladen sind (hier werden sie per Assembler geladen). Die Programmierung ist umständlicher als in den folgenden Beispielen, weil zwei Pointer und ein Zähler dekrementiert werden.

```
*******************************************************************************
BEISPIEL 24: SUBTRAKTION ZWEIER FELDER (POINTER)
000000                  0003          *=$20             ;ZUORDNUNG DER ZEROPAGE
000020  04 50            0004 POINT1 .WOR FELD1+4       ;GERINGWERTIGEN TEIL ADRESSIER
EN
000022  20              0005          .BYT $20          ;BANKADRESSE IM 3. BYTE
000023  04 06           0006 POINT2 .WOR FELD2+4
000025  23              0007          .BYT $23          ;BANKADRESSE VON FELD2
000026  03 00           0008 ZAEHLR .WOR 3              ;FUER 3 WORTE JE FELD
00002B                  0009          *=$205000         ;PARAMETERFELD IN BANK $20
205000  10 11           0010 FELD1  .DBY $1011,$2223,$3435 ;WERT DES MINUENDEN
205002  22 23
205004  34 35           0010
205006                  0011          *=$230600         ;PARAMETERFELD IN BANK $23
230600  00 03           0012 FELD2  .DBY $3,$2120,$0 ;SUBTRAHEND
230602  21 20           0012
230604  00 00           0012
230606                  0013 ;INITIALISIERUNG WIE ZUVOR
004004                  0020          .OPT LIS
004004  F8              0021          SED               ;DEZIMALMODUS HEREINRUFEN
004005  38              0022          SEC               ;SUBTRAKTIONSVORBEREITUNG
004006  A7 20           0023 SBLOOP LDA (POINT1,L)    ;PER POINTER MIT BANK
004008  E7 23           0024          SBC (POINT2,L)    ;SUBTRAHEND
00400A  87 20           0025          STA (POINT1,L)    ;ABLAGE AN ALTER STELLE
00400C  C6 20           0026          DEC POINT1        ;POINTER UM 2 ERNIEDRIGEN
00400E  C6 20           0027          DEC POINT1
004010  C6 23           0028          DEC POINT2        ;AUCH HIER
004012  C6 23           0029          DEC POINT2
004014  C6 26           0030          DEC ZAEHLR        ;ZAEHLER ABBAUEN
004016  D0 EE           0031          BNE SBLOOP
004018  D8              0032          CLD               ;BINAERMODUS ZURUECK
004019                  0033 ;FOLGEBEFEHLE WIE GEHABT
```

Beispiel 25: Subtraktion zweier Felder (indirekt,L),Y

Eleganter als im letzten Beispiel lassen sich Datenfelder nach wie vor bei Benutzung eines Indexregisters zur Adressierung und Zählung bearbeiten. Zu den Indexregistern des 65816 heißt es im Datenblatt, daß sie in die Folgebank hineingreifen, wenn die Summe aus Basisadresse+Index einen Wert von größer 65535 (hex FFFF) ergibt. Sofern wir jetzt die Indexregister auch in 16 Bit Breite laufen lassen, können bei diesen Adressierungsarten mächtige Datenfelder überstrichen werden.

Wir führen wie in Nr. 24 eine Subtraktion durch und gewöhnen uns daran, daß wir zu Anfang der Routine das benutzte Indexregister auf den Stack retten (PHX bzw. PHY) und es am Ende von dort zurückholen (PLX bzw. PLY). So haben wir immer saubere Verhältnisse, und dieses bißchen Ordnung kostet uns kaum eine Anstrengung. - Die Adressierungsart Adresse,L enthält im 4. Byte die Adresse einer angesprochenen Bank, sie greift also in beliebige Bänke. L steht dabei mnemonisch für 'lang'.

```
*******************************************************************************
BEISPIEL 25: SUBTRAKTION ZWEIER FELDER (INDIREKT,L),Y
000000                  0003          *=$20             ;ZUORDNUNG DER ZEROPAGE
000020  04 50            0004 POINT1 .WOR FELD1+4       ;GERINGWERTIGEN TEIL ADRESSIER
EN
000022  20              0005          .BYT $20          ;BANKADRESSE IM 3. BYTE
000023  04 06           0006 POINT2 .WOR FELD2+4
000025  23              0007          .BYT $23          ;BANKADRESSE VON FELD2
000026  03 00           0008 ZAEHLR .WOR 3              ;FUER 3 WORTE JE FELD
00002B                  0009          *=$205000         ;PARAMETERFELD IN BANK $20
205000  10 11           0010 FELD1  .DBY $1011,$2223,$3435 ;WERT DES MINUENDEN
205002  22 23
205004  34 35           0010
205006                  0011          *=$230600         ;PARAMETERFELD IN BANK $23
230600  00 03           0012 FELD2  .DBY $3,$2120,$0 ;SUBTRAHEND
230602  21 20           0012
230604  00 00           0012
230606                  0013 ;INITIALISIERUNG WIE ZUVOR
004002                  0018          .OPT LIS
```

```
004002                  0019        .R16              ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0020        REP'#$30          ;UMSCHALTUNG DER CPU
004004  F8              0021        SED               ;DEZIMALMODUS HEREINRUFEN
004005  38              0022        SEC               ;SUBTRAKTIONSVORBEREITUNG
004006  5A              0023        PHY               ;RETTE Y-REGISTER
004007  A0 04 00        0024        LDY #4            ;ADRESSIERE GANZ NACH RECHTS
00400A  B7 20           0025 SBLOOP LDA (POINT1,L),Y
00400C  F7 23           0026        SBC (POINT2,L),Y
00400E  88              0027        DEY
00400F  10 F9           0028        BPL SBLOOP
004011  7A              0029        PLY               ;Y WIEDDERHERSTELLEN
004012  D8              0030        CLD               ;BINAERMODUS ZURUECK
004013                  0031 ;FOLGEBEFEHLE ANALOG

*******************************************************************************
BEISPIEL 26: SUBTRAKTION MIT DATENUEBERGABE AUF DEM ST
000000                  0003        *=$205000         ;PARAMETERFELD IN BANK $20
205000  10 11           0004 FELD1  .DBY $1011,$2223 ;WERT DES MINUENDEN
205002  22 23           0004
205004                  0005        *=$230600         ;PARAMETERFELD IN BANK $23
230600  00 03           0006 FELD2  .DBY $3,$2120     ;SUBTRAHEND
230602  21 20           0006
230604                  0007 ;INITIALISIERUNG WIE ZUVOR
004002                  0012        .OPT LIS
004002                  0013        .R16              ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0014        REP #$30          ;UMSCHALTUNG DER CPU
004004  F8              0015        SED               ;DEZIMALMODUS HEREINRUFEN
004005  38              0016        SEC               ;SUBTRAKTIONSVORBEREITUNG
004006  DA              0017        PHX               ;RETTE X-REGISTER
004007  BA              0018        TSX               ;RETTE ALTEN STACKPOINTER
004008  AF 02 50 00     0019        LDA FELD1+2,L     ;HOLE MIT BANKADRESSE
00400C  48              0020        PHA
00400D  AF 00 50 00     0021        LDA FELD1,L       ;NOCH MINUEND
004011  48              0022        PHA
004012  AF 02 06 00     0023        LDA FELD2+2,L     ;SUBTRAHEND LOW WORD
004016  48              0024        PHA
004017  AF 00 06 00     0025        LDA FELD2,L
00401B  48              0026        PHA               ;ENDE HINTRANSPORT
00401C  20 35 40        0027        JSR SUBTRA        ;AUSFUEHREN SUBTRAKTION
00401F                  0028
00401F  A3 05           0029        LDA 5,S           ;HOEHERWERTIGER TEIL, STACK-RE
LATIV
004021  8F 00 50 00     0030        STA FELD1,L       ;AN DIE STELLE DES MINUENDEN
004025  A3 07           0031        LDA 7,S           ;STACK-RELATIV NIEDERWERTIGER
TEIL
004027  8F 02 50 00     0032        STA FELD1+2,L
00402B  9A              0033        TXS               ;ALTEN STACKPOINTER EINSTELLEN
00402C  FA              0034        PLX               ;AUCH X ZURUECK
00402D                  0035 ;PROGRAMMFORTSETZUNG HIER
004035                  0045        .OPT LIS
004035                  0046 ;MAN BEACHTE, DA> JSR DIE RETURN-ADRESSE AUF DEN STACK
 LEGTE
004035  38              0047 SUBTRA SEC               ;ALLG. UNTERPROGRAMM SUBTRAKTI
ON
004036  A3 09           0048        LDA 9,S           ;NIEDERWERTIGER TEIL MINUEND
004038  E3 05           0049        SBC 5,S           ;AUCH SUBTRAHEND STACK-RELATIV
00403A  83 09           0050        STA 9,S           ;ERGEBNISABLAGE
00403C  A3 07           0051        LDA 7,S           ;HOEHERWERTIGER TEIL
00403E  E3 03           0052        SBC 3,S
004040  83 07           0053        STA 7,S
004042  60              0054        RTS
004043                  0055        .END
```

Beispiel 26: Subtraktion mit Datenübergabe über den Stack

Wir sollten gedanklich unterscheiden zwischen der Übergabe von Daten, an denen gearbeitet wird, und der Übergabe von Adressen, die auf Daten weisen. Eine 'anonyme' Übergabe ist dabei der Tranport auf den Stack. Für sie haben wir die neue stack-relative Adressierungsart Offset,S. Der Offset darf dabei höchstens 255 sein.

Zunächst zur Datenübergabe: Ein allgemeines Unterprogramm SUBTRA soll Minuend (FELD1) und Subtrahend (FELD2) auf dem Stack vorfinden und die Differenz in FELD1 ablegen. Beide Felder sind 4 Byte breit.

Der Liste für Beispiel 26 ist zu entnehmen, daß diese Art der linearen Programmierung sehr aufwendig ist. Wir sollten uns daher bessere allgemeine Lösungen überlegen.

Beispiel 27: Addition in gleicher Bank mit Adressenübergabe per PEA

Wenn es um die Bearbeitung von Datenfeldern geht, tut man gut daran, auf vermeidbare Transporte von Daten zu verzichten. Damit kommt man auch einfacher zu allgemeinen Lösungen. Statt der Daten werden jetzt die Zeiger auf die Daten auf den Stack gebracht. Wir haben folgende Datenfeldbezeichnungen: FELD1 und FELD2 für die beiden Operanden, die addiert werden sollen und SUMME für das Ergebnis. Die Addition soll in einem Unterprogramm erfolgen, das veränderliche Feldbreiten (BREITE) erlaubt. Damit wird es allgemein benutzbar. Dieses Beispiel geht noch davon aus, daß alle Operanden in der gleichen Datenbank liegen, die beliebig gewählt werden kann. Später werden wir auch einen Mix zwischen Datenbanken zulassen.

```
*******************************************************************************
BEISPIEL 27: ADDITION MIT ADRESSENUEBERGABE DURCH PEA
000000                  0003          *=$5000           ;PARAMETERFELD
005000  10 11          0004 FELD1    .DBY $1011,$2223  ;WERT DES 1. OPERANDEN
005002  22 23          0004
005004  00 03          0005 FELD2    .DBY $3,$2120     ;2. OPERAND
005006  21 20          0005
005008                  0006 SUMME    *=*+4             ;ERGEBNISFELD
00500C                  0007 BREITE   =2                ;ARBEITSBREITE
00500C                  0008 ;INITIALISIERUNG WIE ZUVOR
004002                  0013          .OPT LIS
004002                  0014          .R16              ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0015          REP #$30          ;UMSCHALTUNG DER CPU
004004  F8              0016          SED               ;DEZIMALMODUS HEREINRUFEN
004005  18              0017          CLC               ;ADDITIONSVORBEREITUNG
004006  5A              0018          PHY               ;RETTE Y-REGISTER
004007  BA              0019          TSX               ;RETTE ALTEN STACKPOINTER
00400B  F4 00 50        0020          PEA FELD1         ;BRINGE FELDADRESSE AUF DEN ST
ACK
00400B  F4 04 50        0021          PEA FELD2         ;DITO
00400E  F4 08 50        0022          PEA SUMME         ;
004011  A0 02 00        0023          LDY #BREITE       ;UEBERNAHME DER ARBEITSBREITE
004014  20 21 40        0024          JSR ADDIT         ;FUEHRE ADDITION AUS
004017                  0025
004017  9A              0026          TXS               ;ALTEN STACKPOINTER EINSTELLEN
004018  7A              0027          PLY               ;AUCH Y ZURUECK
004019                  0028 ;PROGRAMMFORTSETZUNG HIER
004021                  0038          .OPT LIS
004021                  0039 ;MAN BEACHTE, DA> JSR DIE RETURN-ADRESSE AUF DEN STACK
 LEGTE
004021  18              0040 ADDIT    CLC               ;ALLG. UNTERPROGRAMM ADDITION
004022  B3 07           0041 ADLOOP   LDA (7,S),Y       ;1. OPERAND
004024  73 05           0042          ADC (5,S),Y       ;2. OPERAND STACK-RELATIV
004026  93 03           0043          STA (3,S),Y
00402B  88              0044          DEY               ;ZAEHLER UND ADDRESSER
004029  88              0045          DEY               ;2X VERMINDERN
00402A  10 F6           0046          BPL ADLOOP
00402C  60              0047          RTS
00402D                  0048          .END
```

Es wird die stack-relative Adressierung (Offset,S),Y benutzt. Der so auf dem Stack angesprochene Pointer hat 2 Byte. Er weist auf eine Datenbasis. Diese wird per Indexregister Y nachindiziert. Wir benutzen zum Transport der Datenbasis auf den Stack den Befehl PEA = Push Effective Address.

Zu beachten ist wiederum, daß bei 65xxx der Stapelzeiger (Stackpointer) auf die nächste freie Stapeladresse weist. Das erste Nutzbyte oder -wort liegt daher stack-relativ zu 1,S. Wenn wir jedoch in einem Unterprogramm sind, so liegt es zu 3,S, wenn der Aufruf mit JSR erfolgte und zu

4,S wenn wir mit JSL (Jump Subroutine Long) in eine andere Programmbank umgestiegen sind, weil diese Aufrufe ja eine Rückkehradresse vor die Parameter auf den Stack legen.

Wir kommen nun zu allgemeinen eleganten Lösungen, die als Vorbild dienen mögen. Es wird eine beliebige Datenbank angesprochen. Nach der Programmausführung wird die ursprüngliche Datenbank wiederhergestellt. Für das Umladen der Datenbank stehen zwar nicht alle erwünschten Register-Transportbefehle zur Verfügung. Mit dem Umweg über den Akku und Stack geht es gleichwohl, siehe auch das Schema in Heft 40, Seite 14.

Beispiel 28: Der Inhalt eines Adreßpointers wird auf den Stack kopiert

In Beispiel 27 wurden die Adressen der Symbole FELD1, FELD2 und SUMME auf den Stack gebracht, wo sie als Pointer dienen konnten. Das war schon eine recht allgemeine Lösung für die Parameterübergabe an ein Unterprogramm. Noch allgemeiner wird man mit dem Befehl PEI zur Parameterübergabe. Er kopiert den *Inhalt* eines Pointers (in der Direct Page) auf den Stack. Erfahrungsgemäß hat man bei Bearbeitungen seine Pointer bereits so geladen, daß sie schon auf die derzeit zu bearbeitenden Daten zeigen. Wenn man Additionen usw. nun immer über gleichartige Sätze von Pointern steuert, so wird man sich zur Übergabe der Pointerinhalte mit PEI entschließen, besonders wenn mehrere Aufgaben gleichzeitig auf dem Rechner laufen.

Die Logik in Nr. 28 folgt derjenigen von Nr. 27. Hier sind die Pointer durch den Assembler bereits so vorgeladen, daß sie auf FELD1, FELD2 und SUMME zeigen.

```
*******************************************************************************
BEISPIEL 28: ADRESSPOINTER MIT PEI KOPIEREN
000000                  0003          *=$20             ;POINTER IN DER DIRECT PAGE
000020  00 50           0004 POINT1 .WOR FELD1 ,ZEIGT AUF 1. OPERANDEN
000022  04 50           0005 POINT2 .WOR FELD2
000024  08 50           0006 POINT3 .WOR SUMME
000026                  0007          *=$5000           ;PARAMETERFELD
005000  10 11           0008 FELD1  .DBY $1011,$2223 ;WERT DES 1. OPERANDEN
005002  22 23           0008          .
005004  00 03           0009 FELD2  .DBY $3,$2120     ;2. OPERAND
005006  21 20           0009
005008                  0010 SUMME  *=*+4             ;ERGEBNISFELD
00500C                  0011 BREITE =2                ;ARBEITSBREITE 2 WORTE
00500C                  0012 ;INITIALISIERUNG WIE ZUVOR
004002                  0017          .OPT LIS
004002                  0018          .R16            ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0019          REP #$30        ;UMSCHALTUNG DER CPU
004004  F8              0020          SED             ;DEZIMALMODUS HEREINRUFEN
004005  18              0021          CLC             ;ADDITIONSVORBEREITUNG
004006  5A              0022          PHY             ;RETTE Y-REGISTER
004007  BA              0023          TSX             ;RETTE ALTEN STACKPOINTER
004008  D4 20           0024          PEI POINT1      ;BRINGE INHALT DES POINTERS AU
F DEN STACK
00400A  D4 22           0025          PEI POINT2
00400C  D4 24           0026          PEI POINT3
00400E  A0 02 00        0027          LDY #BREITE     ;UEBERNAHME DER ARBEITSBREITE
004011  20 1E 40        0028          JSR ADDIT       ;FUEHRE ADDITION AUS
004014                  0029
004014  9A              0030          TXS             ;ALTEN STACKPOINTER EINSTELLEN
004015  7A              0031          PLY             ;AUCH Y ZURUECK
004016                  0032 ;PROGRAMMFORTSETZUNG HIER
00401E                  0042          .OPT LIS
00401E                  0043 ;MAN BEACHTE, DA> JSR DIE RETURN-ADRESSE AUF DEN STACK
 LEGTE
00401E  18              0044 ADDIT  CLC               ;ALLG. UNTERPROGRAMM ADDITION
00401F  B3 07           0045 ADLOOP LDA (7,S),Y       ;1. OPERAND
004021  73 05           0046          ADC (5,S),Y     ;2. OPERAND STACK-RELATIV
004023  93 03           0047          STA (3,S),Y
004025  88              0048          DEY             ;ZAEHLER UND ADDRESSER
004026  88              0049          DEY             ;2X VERMINDERN
004027  10 F6           0050          BPL ADLOOP
004029  60              0051          RTS
00402A                  0052          .END
```

Beispiel 29: Pointerübergabe mit PER

Wie in Heft 40 bereits erwähnt, hat PER zwei Gesichter: Zur Assemblierungszeit berechnet er wie BRL einen Abstand (Offset) zwischen dem Programmzähler, der schon auf den Folgebefehl zeigt, und einem Symbol (Adresse). Zur Laufzeit des Programms dreht sich das um: Die CPU berechnet aus PC und Offset eine effektive Adresse und bringt diese auf den Stack, wo sie als Pointer dienen kann. Die Wirkung ist damit verwandt zu PEA, und wir können das Beispiel 27 entsprechend abwandeln. Gleichwohl gibt es Unterschiede, denn PER eignet sich für positions-unabhängigen Code, während PEA zur Zeit der Assemblierung und der Ausführung immer auf dieselben Adressen führt.. Ein Programm-Modul mit PER, BRA und BRL statt des Befehles JMP kann in beliebigen Speicherbereichen zur Ausführung gebracht werden.

Beispiel 30: Durchgriff für Datenfelder durch mehrere Banken

Für Operanden, die in verschiedenen Banken liegen, haben wir schon folgende Adressierungsarten benutzt: (Pointer,L) und (indirekt,L),Y. Nun gibt es keine Pointeradressierung relativ zum Stapelzeiger, die dort einen 3 Byte breiten Pointer zu benutzen gestattet, etwa in der Schreibweise (Offset,SL),Y. Um für die Bearbeitung von Datenfeldern allgemeine Unterprogramme zu entwikkeln, die in veränderlicher Breite arbeiten und die vollständige Adreßparameter vorfinden (einschl. Bankadresse) und diese verwerten, muß man noch etwas trickreicher vorgehen. Der einfachste Weg scheint die Übergabe eines Zeigers an das Unterprogramm zu sein, der auf die jeweils aktuellen Pointer in der Direct Page hinweist. Und dieser Zeiger liegt in der Veränderung des Direct Page Registers.

Man mag fragen, wann solche Übergaben notwendig sind. Die Antwort heißt, daß jede Task, daß jeder Benutzer und jede Interruptquelle möglichst reservierte Speicherbereiche benutzen können soll, gleichwohl leichten Durchgriff zu allgemein verwertbaren Programmteilen und Daten haben soll.

```
*******************************************************************************
BEISPIEL 29: PDINTERUEBERGABE MIT PER
000000                  0003        *=$5000          ;PARAMETERFELD
005000  10 11           0004 FELD1  .DBY $1011,$2223 ;WERT DES 1. OPERANDEN
005002  22 23           0004
005004  00 03           0005 FELD2  .DBY $3,$2120    ;2. OPERAND
005006  21 20           0005
00500B                  0006 SUMME  *=*+4            ;ERGEBNISFELD
00500C                  0007 BREITE =2               ;ARBEITSBREITE 4 BYTE
00500C                  0008 ;INITIALISIERUNG WIE ZUVOR
004002                  0013        .OPT LIS
004002                  0014        .R16             ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0015        REP #$30         ;UMSCHALTUNG DER CPU
004004  F8             0016        SED              ;DEZIMALMODUS HEREINRUFEN
004005  18             0017        CLC              ;ADDITIONSVORBEREITUNG
004006  5A             0018        PHY              ;RETTE Y-REGISTER
004007  BA             0019        TSX              ;RETTE ALTEN STACKPOINTER
004008  62 F5 0F        0020        PER FELD1        ;BERECHNE OFFSET
00400B  62 F6 0F        0021        PER FELD2
00400E  62 F7 0F        0022        PER SUMME
004011  A0 02 00        0023        LDY #BREITE      ;UEBERNAHME DER ARBEITSBREITE
004014  20 21 40        0024        JSR ADDIT        ;FUEHRE ADDITION AUS
004017                  0025
004017  9A             0026        TXS              ;ALTEN STACKPOINTER EINSTELLEN
004018  7A             0027        PLY              ;AUCH Y ZURUECK
004019                  0028 ;PROGRAMMFORTSETZUNG HIER
004021                  0038        .OPT LIS
004021                  0039 ;MAN BEACHTE, DA> JSR DIE RETURN-ADRESSE AUF DEN STACK
 LEGTE
004021  18             0040 ADDIT  CLC              ;ALLG. UNTERPROGRAMM ADDITION
004022  B3 07          0041 ADLOOP LDA (7,S),Y      ;1. OPERAND
004024  73 05          0042        ADC (5,S),Y      ;2. OPERAND STACK-RELATIV
004026  93 03          0043        STA (3,S),Y
004028  88             0044        DEY              ;ZAEHLER UND ADDRESSER
004029  88             0045        DEY              ;2X VERMINDERN
00402A  10 F6          0046        BPL ADLOOP
00402C  60             0047        RTS
00402D                  0048        .END
```

Diese wohl einfachste Übergabe von Zeigern per Direct Register an ein Unterprogramm erfordert lediglich, daß Zeiger z.B. für Addition innerhalb eines Zeigerfeldes die gleiche Anordnung haben, die gleiche Adressendistanz. Und bei dieser Art der Lösung tritt auch nicht die Frage auf, ob man per JSR oder JSL in das Unterprogramm gelangte, was ja einen Einfluß auf die stack-relative Lage von Parametern hat.

Dem Autor scheint, daß speziell für 65802 und /816 die Parameterübergabe über den Stack nicht mehr zu empfehlen ist, außer es handelt sich um Kleinigkeiten. Der Verwaltungsaufwand ist zu groß, wie wir an den früheren Beispielen gesehen haben. Man sollte besser den Weg gehen, einen Zeiger auf Zeiger zur Verfügung zu stellen, und das geschieht wie in Beispiel 30 durch Parameterübergabe für die Direct Page. Bei diesem Vorgehen braucht man in einem Betriebssystem oder in einer Sprache nur noch eine oder zwei Routinen für jede Grundleistung der Addition, Subtraktion usw. Eine, wenn man nur binär arbeitet, zwei, wenn man auch dezimal arbeitet. Die Bearbeitungsbreite wird in einem Register übergeben, bevorzugt Y. Man hat dann Durchgriff durch alle

```
*******************************************************************************
BEISPIEL 30: DURCHGRIFF DURCH MEHRERE BANKEN
000000                  0003          *=$333              ;ZEIGERFELD IN DER SPAETEREN D
IREKT PAGE
000333  00 50           0004 POINTR .WOR FELD1           ;ADRESSABLAGE
000335  23              0005          .BYT $23           ;BANK VON FELD1
000336  50 55           0006          .WOR FELD2
000338  25              0007          .BYT $25           ;BANK FELD2
000339  50 58           0008          .WOR SUMME
00033B  26              0009          .BYT $26           ;BANK DER SUMME
00033C                  0010          *=$235000          ;PARAMETERFELD  IN BANK 23
235000  10 11           0011 FELD1    .DBY $1011,$2223   ;WERT DES 1. OPERANDEN
235002  22 23           0011
235004                  0012          *=$25550           ;PARAMETERFELD IN BANK 25
025550  00 03           0013 FELD2    .DBY $3,$2120      ;2. OPERAND
025552  21 20           0013
025554                  0014          *=$265850          ;ERGEBNISFELD IN BANK 26
265850                  0015 SUMME    *=*+4              ;ERGEBNISFELD
265854                  0016 BREITE   =2                 ;ARBEITSBREITE
265854                  0017 ;INITIALISIERUNG WIE ZUVOR
004002                  0022          .OPT LIS
004002                  0023          .R16               ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0024          REP #$30           ;UMSCHALTUNG DER CPU
004004  F8              0025          SED                ;DEZIMALMODUS HEREINRUFEN
004005  18              0026          CLC                ;ADDITIONSVORBEREITUNG
004006  5A              0027          PHY                ;RETTE Y-REGISTER
004007  0B              0028          PHD                ;RETTE DIRECT PAGE REGISTER
004008  A9 33 03        0029          LDA #POINTR        ;BEGINN DES ZEIGERFELDES
00400B  A0 02 00        0030          LDY #BREITE        ;UEBERNAHME DER ARBEITSBREITE
00400E  20 1B 40        0031          JSR ADDIT          ;FUEHRE ADDITIOM AUS
004011                  0032
004011  2B              0033          PLD                ;DIRECT PAGE REGISTER ZURUECK
004012  7A              0034          PLY                ;Y ZURUECK
004013                  0035 ;PROGRAMMFORTSETZUNG HIER
00401B                  0045          .OPT LIS
00401B  5B              0046 ADDIT    TCD                ;LADE DIRECT PAGE REGISTER FUE
R DIESEN AUFRUF
00401C  18              0047          CLC                ;ALLG. UNTERPROGRAMM ADDITION
00401D  B7 00           0048 ADLOOP LDA (0,L),Y          ;1. OPERAND
00401F  77 03           0049          ADC (3,L),Y        ;2. OPERAND STACK-RELATIV
004021  97 06           0050          STA (6,L),Y        ;ERGEBNIS NACH SUMME
004023  88              0051          DEY                ;ZAEHLER UND ADDRESSER
004024  88              0052          DEY                ;2X VERMINDERN
004025  10 F6           0053          BPL ADLOOP
004027  60              0054          RTS
004028                  0055          .END
 ERRORS=0000


ADDIT   00401B  #0046  ADLOOP  00401D  #0048  BREITE  000002  #0016
FELD1   005000  #0011  FELD2   005550  #0013  POINTR  000333  #0004
SUMME   005850  #0015
```

Bänke für die Operanden und kann Integers, doppelt genaue Zahlen und andere Datentypen mit einem Unterprogramm 'erschlagen'.

Das gilt entsprechend auch für die Übergabe von Daten (also nicht nur von Pointeradressen). Wir brauchen das Direct Register nur auf die Daten zeigen zu lassen, um sie zu übergeben. Ein Transport in Briefkästen, an vereinbarte Stellen, ist dann nicht mehr nötig. Und per Direct Register haben wir überall in der Bank 00 die verkürzte und schneller ausführende Adressierung in der Zero Page. Insgesamt haben wir mit 65802/816 eine sparsamere Programmierung.

Verschiebebefehle (Beispiel 32)

In der 16 Bit breiten Betriebsweise (.A16) des Akkus werden nicht nur die schon erwähnten Befehle INC und DEC entsprechend betroffen, sondern auch die Verschiebebefehle ASL, LSR, ROL und ROR, und zwar sowohl bei Verschiebungen im Akku, als auch bei solchen im Speicher (Read/Modify/Write). Es tritt nun eine Besonderheit auf, die zwar logisch in Ordnung ist, die aber beachtet werden muß: Das höherwertige Byte eines Wortes wird beim 65816 an der jeweils höheren Adresse abgespeichert. Wenn wir jetzt wie in Beispiel 32 das Carry-Bit in ein Datenwort einziehen, so erscheint es zunächst in der Adresse $5001. Nach dem Programmlauf finden wir dort das Ergebnis $40. Wenn wir nun weitere Verschiebungen folgen ließen, so würde das eingezogene Bit schließlich im niederwertigen Teil des Wortes erscheinen, im Byte unter der Adresse 5000. Das widerspricht etwas der Vorstellung bei einer Verschiebung z.B. 'von links nach rechts'.

```
*******************************************************************************
BEISPIEL 32: VERSCHIEBEBEFEHLE
000000                0003 ;INITIALISIERUNG WIE ZUVOR
004004                0010         .OPT LIS
004004  DA            0011         PHX              ;RETTE X-REGISTER
004005  A2 06 00      0012         LDX #6           ;FUER 11 DURCHLAEUFE
004008  A9 00 00      0013         LDA #0
00400B  38            0014         SEC              ;WIR WOLLEN DIE WANDERUNG
00400C  6A            0015 VLOOP   ROR A            ;DIESES BITS BEOBACHTEN
00400D  CA            0016         DEX
00400E  D0 FC         0017         BNE VLOOP
004010  8D 00 50      0018         STA $5000
004013  FA            0019         PLX              ;X ZURUECK
004014                0020 ;PROGRAMMFORTSETZUNG HIER
```

Beispiel 31: Multiplikation 16*16 Bit

Die eben erwähnte Besonderheit werden wir bei der nun folgenden binären Multiplikation in der Anordnung der Operanden berücksichtigen müssen. Wenn wir den ersten Faktor, FAKT1 mit .DBY $0300 erkären, so hat er den Wert 3 (niedriges/hohex Byte). Wir hätten ihn auch als .WOR 3 erklären können, was vielleicht einleuchtender ist, weil diese Erklärung den Wert besser wiedergibt. Entsprechend für FAKT2. Auch das Summenfeld ist jetzt so orientiert, daß im Wort bei Adresse SUMME der niederwertige Teil des Ergebnisses steht und bei SUMME+2 der höherwertige.

Der Algorithmus der binären Multiplikation ist denkbar einfach: Es wird die wiederholte Addition ausgeführt, wenn ein Bit im ersten Faktor '1' ist, danach folgt die stellenrichtige Ergebnisverschiebung. Wenn das Bit '0' ist, dann erfolgt nur eine Verschiebung.

Zur Abkürzung des Ablaufes haben wir eine Besonderheit programmiert, ähnlich wie im Beispiel 30: Das Direkt Page Register wird auf das Datenfeld gerichtet. Das geschieht mit den Befehlen LDA #FAKT1 und TCD (Transfer in das Direct Register). Der Akku dient danach der Aufnahme des höherwertigen Zwischenergebnisses, bis er schließlich nach SUMME+2 abgespeichert wird. - Um per Assembler auch eine Direct Page-Adressierung zu erhalten, werden alle Symbole auf den Stand des Direct Page Registers bezogen, hier auf FAKT1.

In den Bytes ab Adresse hex 5000 finden wir später 09 00 00 00.

Zu den Verschiebebefehlen und INC und DEC ist zu bemerken, daß für sie im 65802/816 keine Erweiterung der Adressierungsarten stattgefunden hat, wie sie für die arithmetischen und logischen Befehle vorgenommen wurde. Um sie in allen Bänken zu benutzen, wird man den Weg über eine Veränderung des Datenbankregisters gehen müssen.

```
*******************************************************************************
BEISPIEL 31: MULTIPLIKATION 16*16 BIT
000000                  0003          *=$5000          ;PARAMETERFELD
005000  03 00           0004 FAKT1    .DBY $300        ;1. FAKTOR
005002  03 00           0005 FAKT2    .DBY $300        ;2. FAKTOR
005004                  0006 ;ERGEBNISFELD
005004                  0007 SUMME    *=*+4            ;ERGEBNISFELD
00500B                  0008 ;INITIALISIERUNG WIE ZUVOR
004002                  0013          .OPT LIS
004002                  0014          .R16             ;DIRECTIVE FUER ASSEMBLER: ALL
E REGISTER 16 BIT
004002  C2 30           0015          REP #$30         ;UMSCHALTUNG DER CPU
004004  9C 04 50        0016          STZ SUMME        ;ERGEBNISFELD LOESCHEN
004007  DA              0017          PHX              ;RETTE X-REGISTER
004008  0B              0018          PHD              ;RETTE DIRECT PAGE REGISTER
004009  A9 00 50        0019          LDA #FAKT1       ;BEGINN DES ARBEITSFELDES
00400C  5B              0020          TCD              ;IN DAS DIRECT PAGE REGISTER
00400D  A2 10 00        0021          LDX #16          ;FUER 16 DURCHLAEUFE
004010  A9 00 00        0022          LDA #0           ;HIER WIRD DAS HIGH BYTE GEFUE
HRT
004013  46 00           0023 MULOOP LSR FAKT1-FAKT1    ;STELLE UM STELLE MULTIPLIZIER
EN
004015  90 03           0024          BCC MUL1
004017  18              0025          CLC              ;ADDIERE OHNE UEBERTRAG
004018  65 02           0026          ADC FAKT2-FAKT1
00401A  4A              0027 MUL1     LSR A            ;PRODUKT RECHTS SCHIEBEN
00401B  66 04           0028          ROR SUMME-FAKT1
00401D  CA              0029          DEX
00401E  D0 F3           0030          BNE MULOOP
004020  85 06           0031          STA SUMME+2-FAKT1
004022  2B              0032          PLD              ;DIRECT PAGE REGISTER ZURUECK
004023  FA              0033          PLX              ;AUCH X ZURUECK
004024                  0034 ;PROGRAMMFORTSETZUNG HIER
```

Beispiel 33: Interfacebausteine im 16 Bit-Modus

Wo es nicht zu Betriebsstörungen führen kann, dürfen wir auch benachbarte Register in Interfacebausteinen mit einem 16 Bit-Befehl ansprechen. Wie das Beispiel zeigt, können zwei Ports einer VIA 6522 mit einem Schreibvorgang in der Datenrichtung (hier Ausgang) und mit einem weiteren mit den Ausgangssignalen eingerichtet werden. Auch beim Timer 1 können wir so verfahren. Bekanntlich soll man beim Laden zunächst das Low Order Latch beschreiben. Wenn man danach in den Counter High schreibt, dann startet der Timer. Nun wird das Low Order Latch (Vorspeicher) sowohl an der Adresse VIA+4 als auch bei VIA+6 angesprochen. Wenn wir jetzt sagen STA VIA+4, so wird zunächst das Low Order Latch geladen. Das im nächsten Zyklus bei VIA+5 landende Byte startet dann den Timer. Wenn wir den Zähler des Timers mit dem nächsten Befehl wieder auslesen, so finden wir wir ihn bereits um 3 vermindert.

```
*******************************************************************************
BEISPIEL 33: VIA 6522 IM 16 BIT MODUS
000000                  0003 VIA      =$A000           ;BEIM AIM 65
000000                  0004 PORTB    =VIA             ;PORTREGISTER B UND A
000000                  0005 DDRB     =VIA+2           ;DATENRICHTUNGSREGISTER
000000                  0006 T1CL     =VIA+4           ;TIMER 1 COUNTER LOW/HIGH
000000                  0007 ;INITIALISIERUNG WIE ZUVOR
004004                  0014          .OPT LIS
004004  A9 FF FF        0015          LDA #$FFFF       ;ALLE PINS ALS AUSGANG
004007  8D 02 A0        0016          STA DDRB         ;DATENRICHTUNGSREGISTER
00400A  A9 CC CC        0017          LDA #$CCCC       ;DAS SEI DAS AUSGANGSSIGNAL
00400D  8D 00 A0        0018          STA PORTB
004010  8D 04 A0        0019          STA T1CL         ;TIMER 1 COUNTER & LATCH LOW
004013  AD 04 A0        0020          LDA T1CL         ;LADEN ZUR KONTROLLE
004016  8D 00 50        0021          STA $5000
004019                  0022 ;PROGRAMMFORTSETZUNG HIER
```

Der BIT-Befehl

Der BIT-Befehl weist eine Besonderheit auf, die dem Autor bei der CPU R65C02 noch gar nicht aufgegangen war, die dort evtl. auch nicht besteht. Bei der Adressierungsart BIT # (immediate)

mit Opcode 89 Wird das V-Bit im Status (Overflow) nicht beeinflußt, auch wenn es nach dem Direktoperanden eigentlich sein 'müßte'. Bei der Adressierung von Interfaces und Speichern reagiert der BIT-Befehl natürlich wie normal. Nun muß man allerdings auch bemerken, daß die Beeinflussung des V-Flags überflüssig wäre, kennt man doch seinen Direktoperanden!

Die Befehle PHA, PHX usw. und PHP/PLP

Im vollen16 Bit Modus (.R16) werden bei PHA, PHX usw. und bei den vom Stack zurückholenden Befehlen PLA, PLX ... natürlich immer 2 Byte transportiert. Eine Ausnahme machen PHP und PLP, sie transportieren immer nur 1 Byte. Man muß sich das vor Augen halten, wenn man (in seltenen Fällen) einmal den Status via Akku in einer Speicherstelle aufbewahren will.

Block Move (Beispiel 35)

Nachdem die Befehle MVN und MVP im Assembler implementiert sind, sollen sie auch in ihrer Schreibweise einmal dokumentiert werden. Das X-Register soll für diese Befehle den Quellbereich des Datentransportes enthalten, das Y-Register die Adresse des Zielbereiches und der Akku die Menge (hex) der zu transportierenden Bytes, er dient also als Zähler. CPU-intern scheint eine Schleife für diese Befehle eingebaut zu sein: Erniedrige den Programmzähler um 3 (um die Länge von MVN und MVP), solange der Akku noch keinen Unterlauf nach hex FFFF hat. Der Akku wird natürlich bei jedem Schleifendurchlauf um 1 erniedrigt. Zu den Bytes des Befehles ist zu sagen, daß auf den Opcode die Zielbank folgt und dahinter die Quellbank. Wenn wir also schreiben MVN $4455, so wird im Speicher zunächst die 55 abgelegt, dahinter die 44, so daß von Bank 44 nach Bank 55 transportiert wird. Mnemonisch haben wir damit: MVN Quell-/Zielbank.

MVN erhöht mit jedem Schleifendurchlauf die Adressen in X und X. Der Befehl faßt damit ein Datenfeld von vorne an. MVP erniedrigt laufend die Adressen in den Indexregistern und faßt ein Feld damit von hinten an. Wie die Befehle beim Überschreiten einer Bankgrenze wirken, das bleibt noch zu studieren.

```
*******************************************************************************
BEISPIEL 35: BLOCK-MOVE
004004                 0009          .OPT LIS
004004  A9 FF 0F      0010          LDA #$FFF       ;HEX 1000 BYTE TRANSPORTIEREN
004007  A2 FF FF      0011          LDX #$FFFF      ;NEHMEN WIR DAS F-ROM
00400A  A0 FF 6F      0012          LDY #$6FFF      ;KOPIEREN ES VON 6000-6FFF
00400D  44 55 44      0013          MVP $4455       ;VON BANK $44 NACH BANK$55
004010  8D 00 03      0014          STA $300
004013  8E 02 03      0015          STX $302
004016  8C 04 03      0016          STY $304
```

Roland Löhr

Dis- und Reassembler

für alle 65xxx-CPU's

Nachdem in der 65xxx-Familie jetzt mehrere unterschiedliche Befehlssätze implementiert sind, so für 6502, R65C02 und 65802/816, ist es an der Zeit, einen Disassembler vorzulegen, der wahlweise für jeden Typ disassemblieren kann. Der Autor hat dabei ein von Grund auf eigenes Programm entwickelt um nicht mühsam und unübersichtlich an etwa bestehende Vorlagen anzuflicken. Es sollte vielmehr ein klarer und gedanklich nachvollziehbarer Aufbau errreicht werden. Die Grundideen sollten ferner als eine Vorlage für ähnliche Disassembler dienen können, wenn man die mnemonischen Bezeichnungen der Befehle und die Aufschlüsselung der Adressierungsarten berücksichtigt. Auch sollte die Textausgabe so gestaltet werden, daß sie unmittelbar wieder als Quelltext für einen Assembler dienen kann (Funktion als Re-Assembler). Operanden werden in einem solchen Falle nicht mit einem vorangestellten '$'-Zeichen gelistet, sondern mit vorangestelltem 'H' als Kunstlabel. Das gilt auch für den Programmzähler. - Unbekannte Opcodes werden

immer als .BYT $ ausgegeben, mit der nachfolgenden Interpretation als druckbares ASCII-Zeichen, wenn der Code das hergibt.

Wenn man die abweichenden drei Befehlssätze berücksichtigen will, die im Initialisierungsteil mit einem Menue ausgewählt werden, so wird der Programmablauf schon ganz schön kompliziert, denn es gibt viele Sonderfälle, die bei den neueren CPU's durch immer weniger symmetrische Architekturen bedingt sind. Die Lücken in der ursprünglichen Opcode-Matrix wurden zwar mit nützlichen Befehlen gefüllt, aber z.T. eben unregelmäßig. Man denke hier z.B. an die Befehle BIT und STZ oder an die Gruppen JMP und JSR. - Beim 65802/816 ist bei der Befehlslänge ferner zu berücksichtigen, daß die üblichen Befehle für Direktoperanden je nach Betriebsweise der Register zwei und drei Byte lang sein können, auch gemischt für die Register Akku bzw. X und Y.

Wegen dieser komplizierten Aufgabenstellung entschloß sich der Autor, das hier schon oft vertretene Prinzip 'ON GOTO' in Maschinensprache zu verwirklichen, um an den Aufsetzpunkten möglichst klar zu wissen, welche Befehle oder Befehlsgruppen dorthin vektoriert wurden. Es zeigte sich wiederum, daß man mit diesem Prinzip eine klare Programmierung mit wenig 'Reibungsverlusten' erreicht.

Zur Verwirklichung: Nach der Initialisierung tritt man in die Gruppierungsschleife BEFPAR ein, die den Opcode mit AND und EOR absucht, und zwar per Tabelle MASKE. Je nach CPU tritt man mit einer anderen Vorladung in X nach BEFPAR ein. Mit dem höchsten X wird für 6502 geladen, weil hier die Opcode-Matrix die größten Löcher hat. Es kommt damit zur möglichst frühen Ausschaltung 'verbotener' Codes, die auf das Label BYTE vektorieren (Ausgabe als 1 Byte). Das wiederholt sich entsprechend am Eintrittspunkt für die CPU R65C02. Bei ihr ist zu berücksichtigen, daß Opcodes mit x7 und xF eine andere Bedeutung als beim 65816 haben, so daß solche möglichst früh abgefangen werden müssen. Danach läuft BEFPAR auf den Einsatzpunkt für 65816 zu, wo einerseits die Gemeinsamkeiten aller CPUs abgehandelt werden und andererseits natürlich die exklusiven Befehle der 65816.

Sobald eine Befehlsgruppe erkannt ist, wird bei FOUND indirekt mit dem in X erreichten Zählerstand auf die Sprungleiste JMPTBL vektoriert und von dort weiter auf den mittleren Programmabschnitt, dessen Label mit 'GR' beginnen. Das ist das 'ON GOTO-Prinzip', hier noch mit dem Befehlssatz der 6502 verwirklicht. - An den Ankunftsstellen werden Besonderheiten abgefragt, speziell auch die Indizierung. Im Akku wird die Adressierungsart übergeben (BEFMOD), in Y die Befehlslänge (BEFLEN) und in X der Zeiger auf eine Tafel TXTADR, in der die Beginne der menmonischen Texte T0 bis TF enthalten sind. Die Übergabe der drei Parameter erfolgt in PARPAS. Diese Technik mag interessant sein, wenn Texttabellen länger als 256 Bytes sind.

Alle Zwischenstufen springen direkt oder indirekt zu ZEILE, zur Zeilenausgabe. Bei der Fülle der Adressierungsarten hatte der Autor die Erwartung, daß das Erzeugen einer vernünftigen Ausgabezeile eine wohl logisch komplizierte Aufgabe sei. Es hat sich gleichwohl eine einfache Codesequenz zwischen den Labeln ZEILE und WEITER ergeben, auch in der Option, Operanden und Programmzähler mit vorangestelltem 'H' als Labels für Quelltextausgabe zu erzeugen. Beim PER-Befehl wird als Kommentar sogar die effektive Adresse als Kommentar nachgeschickt. Maßgeblich für die Abfragesequenz ist neben gelegentlich INDEX das Byte BEFMOD. Die Bedeutung seiner einzelnen Bits ist am Beginn der Programm-Liste erklärt.

Zusätzliche Hinweise: Dieser Disassembler läuft natürlich auch auf 65802/816. Dort möchte man jedoch in beliebigen Bänken disassemblieren können. Dazu sind nur ganz geringe Veränderungen notwendig. In der Initialisierung muß man die Variable BANK interaktiv mit der Zielbank füllen. Die zu analysierenden Bytes eines Programmes werden an wenigen Stellen mit LDA (VIRTPC),Y geholt, Opcode B1 10. Wenn man auf der 65816 stattdessen schreibt LDA (VIRTPC,L),Y, so entsteht der Opcode B7 10. Jetzt wird der 3 Byte breite Pointer VIRTPC inkl. BANK ausgewertet. Das sind schon alle Änderungen!

Zur Hantierung und Anpassung

Im Anschluß an den Start finden wir eine Befehlsleiste, die auf den AIM 65 angepaßt ist. OUTALL gibt ein ASCII-Byte im Akku auf die aktive Ausgabe aus. Das kann auch ein Textsystem sein. Man wird ggfs. noch einen Ausgabeverteiler vom Typ WHEREO (Bestimmung der Ausgabe) in die Initialisierung einbauen. NUMA gibt den Akku-Inhalt als 2 Hexbytes aus. FROM erfragt die Startadresse der Disassemblierung, die sich danach in ADDR befindet. TO entsprechend für die Endadresse-1. RCHEK fragt die ESC-Taste ab. Wenn betätigt, Rückkehr zum Monitorprogramm. GETKEY wartet auf einen Tastendruck. Das Zeichen wird im Akku übergeben. - Eine Anpassung dieser Sprünge auf andere Systeme sollte einfach sein, weil es solche Routinen überall gibt.

Die Benutzerführung im Menue erfolgt interaktiv. Viele Abfragen können einfach mit RETURN beantwortet werden, so daß man schnell in die 'normale' Disassemblierung gelangt. - Disassemblierung heißt häufig, daß man sich einen Bereich zunächst einmal auf dem Bildschirm ansieht, um sich dabei Notizen zu machen, wo eine besondere Aufschlüsselung notwendig ist. Da kommen ASCII-Texte vor und auch Anweisungen wie .WOR. Dieses Programm erlaubt es in der interaktiven Abfrage, auch solche Ausgaben zu erzeugen.

Mit dem Grundstock dieses Programmes sind Anpassungen an weitere Wünsche des Anwenders leicht möglich. So wird z.B. ein EPROM auf einen Sockel gesteckt, aus einem Adreßbereich heraus disassembliert, der nicht dem dem normalen Standort entspricht. In diesem Falle ist die Routine PCOUT um einen Offset zu korrigieren, auch der VIRTPC bei Verzweigungsbefehlen und PER. Die notwendigen Änderungen sind geringfügig.

Zusammenfassung

In Hinsicht auf klare Linien ist das nachstehende Programm zwar umfangreich. Speicherbedarf ist künftig aber nicht mehr kritisch. Es kam auf ein Systemprogramm an, das für einige Zeit zukunftssicher auf verschiedenen Rechnern implementiert werden kann und das für alle jetzigen CPU's nützliche Dienste tut. Es ist im Code des 6502 geschrieben, in der kleinsten gemeinsamen Menge des Befehlssatzes. Der Verzicht auf Kunstgriffe und Verdichtungen erlaubt es, aus diesem Programm Disassembler für den Code anderer CPU's zu entwickeln.

```
000000                0000 ;DISASSEMBLER FUER 6502, 65C02 65816
000000                0001 ;COPYRIGHT 1985 BY ROLAND LOEHR
000000                0002 ;DATUM UND VERSION 09.01.85 V1
000000                0003 ;VARIABLE IM RAM
000000                0004       *=$10
000010                0005 VIRTPC *=*+2            ;VIRTUELLER PC
000012                0006 BANK   *=*+1            ;BANKADRESSE FUER 65816, INTER
AKTIV ABFRAGEN!
000013                0007 MODE   *=*+1            ;0=6502 $40=65C02 $80=65816
000014                0008 BREITE *=*+1            ;0=ALLES 8 BIT, $40=INDICES/$8
0=AKKU 16 BIT
000015                0009 INDEX  *=*+1            ;0=UNINDIZIERT $FF=INDIZIERTER
 BEFEHL
000016                0010 SPRUNG *=*+2            ;POINTER FUER INDIR SPRUNG
000018                0011 OPTION *=*+1            ;0=UEBLICHE AUSGABE, $FF= '$'
ALS 'H' KEIN HEX
000019                0012 FORMAT *=*+1            ;0=DISASSEMBLE, $40=.BYT-, $80
=.WOR-AUSGABE
00001A                0013 TEMP   *=*+2            ;ZWISCHENSPEICHER & TEXTBASIS
00001C                0014 BEFLEN *=*+1            ;BEFEHLSLAENGE INSGESAMT
00001D                0015 BEFMOD *=*+1            ;ADRESSIERUNGSART DES BEFEHLES
00001E                0016 ;LEGENDE: BIT7=1 AKKU-MODUS BEI IMPLIED
00001E                0017 ;         BIT6=1 BRANCH, OPERAND RELATIV AUSGEBEN
00001E                0018 ;         BIT5=1 INDIREKT, '(' UND ')' AUSGEBEN
00001E                0019 ;         BIT4+BIT3+BIT2=0 IST ABSOLUT, OHNE INDEX
0001E                 0020 ;            =001 IST ',X' =010 IST ',Y' =011 IST ',S'
00001E                0021 ;             =100 IST ',L' =101 IST '#'
00001E                0022 ;         BIT1+BIT0=00 AUF ')' KLAMMER ZU FOLGT NICHTS
00001E                0023 ;            =10 ES FOLGT DORT ',Y'  =11 ES FOLGT ',L
'
```

```
00001E                  0024 ;     BYTE=$7F =PER-BEFEHL
00001E                  0025 ;     BYTE=$DF =GRUPPE BRANCH ON BIT SET/RESET
00001E                  0026 ;SYSTEMADRESSEN AIM 65
00001E                  0027 OUTALL =$E9BC          ;AUSGABE VON ASCII-ZEICHEN AUF
 AKTIVE EINHEIT
00001E                  0028 NUMA   =$EA46          ;AUSGABE AKKU=2 HEXBYTES
00001E                  0029 FROM   =$E7A3          ;ABFRAGE STARTADRESSE
00001E                  0030 TO     =$E7A7          ;ENDADRESSE
00001E                  0031 GETKEY =$EC40          ;TASTATUREINGABE
00001E                  0032 RCKEK  =$E907          ;ESC-TASTE GEDRUECKT?
00001E                  0033 MON    =$E1B2          ;RUECKSPRUNGADRESSE MONITOR, A
IM
00001E                  0034 ;MONRAM-ADRESSEN AIM
00001E                  0035 ADDR   =$A41C
00001E                  0036        *=$1000         ;START DES DISASSEMBLERS
001000  20 A3 E7        0037        JSR FROM        ;STARTADRESSE
001003  AD 1C A4        0038        LDA ADDR        ;PARAMETERTRANSPORT
001006  85 10           0039        STA VIRTPC
001008  AD 1D A4        0040        LDA ADDR+1
00100B  85 11           0041        STA VIRTPC+1
00100D  20 A7 E7        0042        JSR TO          ;ENDADRESSE
001010  A0 00           0043        LDY #0          ;ABFRAGE DER CPU
001012  84 14           0044        STY BREITE      ;DEFAULT 8 BIT
001014  84 18           0045        STY OPTION      ;NORMALE AUSGABE/DEFAULT
001016  84 19           0046        STY FORMAT      ;NORMALE DISASSEMBLIERUNG
001018  20 BC 14        0047        JSR TXTOUT
00101B  20 40 EC        0048        JSR GETKEY      ;WELCHE TASTE?
00101E  C9 0D           0049        CMP #$0D        ;RETURN=6502
001020  D0 06           0050        BNE ANDERE
001022  A9 00           0051        LDA #0
001024  85 13           0052        STA MODE
001026  F0 32           0053        BEQ START2
001028  C9 47           0054 ANDERE CMP #'G'
00102A  D0 2A           0055        BNE ROCKW       ;IST R65C02
00102C  A9 80           0056        LDA #$80
00102E  85 13           0057        STA MODE
001030  A0 25           0058        LDY #TEXT3-TEXTE
001032  20 BC 14        0059        JSR TXTOUT
001035  20 40 EC        0060        JSR GETKEY
00103B  C9 0D           0061        CMP #$0D        ;RETURN?
00103A  D0 04           0062        BNE ANDRE1
00103C  A9 C0           0063        LDA #$C0        ;ALLES 16 BIT. BREIT
00103E  D0 11           0064        BNE ANDRE4
001040  C9 41           0065 ANDRE1 CMP #'A'
001042  D0 04           0066        BNE ANDRE2
001044  A9 80           0067        LDA #$80        ;A=16, X=8 BREIT
001046  D0 09           0068        BNE ANDRE4
001048  C9 58           0069 ANDRE2 CMP #'X'
00104A  D0 03           0070        BNE ANDRE3
00104C  A9 40           0071        LDA #$40        ;A=8, X=16 BREIT
00104E  2C              0072        .BYT $2C        ;SKIP OVER INSTRUCTION
00104F  A9 00           0073 ANDRE3 LDA #0          ;ALLE REGISTER 8 BIT
001051  85 14           0074 ANDRE4 STA BREITE
001053  4C 5A 10        0075        JMP START2
001056  A9 40           0076 ROCKW  LDA #$40
001058  85 13           0077        STA MODE
00105A  A0 9E           0078 START2 LDY #TEXT9-TEXTE
00105C  20 BC 14        0079        JSR TXTOUT
00105F  20 40 EC        0080        JSR GETKEY      ;AUSGABEOPTION?
001062  C9 0D           0081        CMP #$0D        ;RETURN?
001064  F0 02           0082        BEQ FORMO
001066  C6 18           0083        DEC OPTION
001068  A0 C1           0084 FORMO  LDY #TEXT10-TEXTE ;AUSGABE .BYT ODER .WDR?
00106A  20 BC 14        0085        JSR TXTOUT
00106D  20 40 EC        0086        JSR GETKEY
001070  C9 0D           0087        CMP #$0D        ;CR?
001072  F0 10           0088        BEQ START3
001074  C9 42           0089        CMP #'B'
001076  D0 04           0090        BNE FORM1
001078  A9 40           0091        LDA #$40        ;FUER .BYT
00107A  D0 06           0092        BNE FORM3
```

```
00107C  C9 57        0093 FORM1  CMP #'W'
00107E  D0 E8        0094        BNE FORM0          ;FEHLBEDIENUNG
001080  A9 80        0095        LDA #$80
001082  85 19        0096 FORM3  STA FORMAT
001084  A9 0D        0097 START3 LDA #$0D
001086  20 BC E9     0098        JSR OUTALL
001089  38           0099        SEC
00108A  A5 10        0100        LDA VIRTPC         ;OBERGRENZE ERREICHT?
00108C  ED 1C A4     0101        SBC ADDR
00108F  A5 11        0102        LDA VIRTPC+1
001091  ED 1D A4     0103        SBC ADDR+1
001094  90 03        0104        BCC START5
001096  4C B2 E1     0105        JMP MON            ;ABBRUCH
001099  20 07 E9     0106 START5 JSR RCKEK          ;ABBRUCH?
00109C  20 AB 14     0107        JSR PCOUT          ;AUSGABE VIRTPC
00109F  A0 00        0108        LDY #0
0010A1  84 15        0109        STY INDEX          ;INIT NICHT INDIZIERT
0010A3  84 1D        0110        STY BEFMOD         ;DEFAULT UNINDEXED
0010A5  B1 10        0111        LDA (VIRTPC),Y     ;OPCODE
0010A7  24 19        0112        BIT FORMAT         ;WELCHE AUSGABEART?
0010A9  10 03        0113        BPL START6
0010AB  4C 12 11     0114        JMP WORD
0010AE  50 03        0115 START6 BVC START7
0010B0  4C DA 10     0116        JMP BYTE
0010B3  29 10        0117 START7 AND #$10           ;HALBBYTE UNGERADE?
0010B5  F0 02        0118        BEQ EVEN
0010B7  C6 15        0119        DEC INDEX
0010B9  B1 10        0120 EVEN   LDA (VIRTPC),Y
0010BB  24 13        0121        BIT MODE           ;WELCHE CPU
0010BD  30 0A        0122        BMI GTE816
0010BF  70 04        0123        BVS R65C02
0010C1  A2 71        0124        LDX #113           ;TABELLENELEMENTE X2-1
0010C3  D0 06        0125        BNE ALLCPU         ;ALWAYS
0010C5  A2 5D        0126 R65C02 LDX #93            ;# ELEMENTE
0010C7  D0 02        0127        BNE ALLCPU
0010C9  A2 47        0128 GTE816 LDX #71            ;#ELEMENTE
0010CB  D8           0129 ALLCPU CLD
0010CC  B1 10        0130 BEFPAR LDA (VIRTPC),Y     ;GRUPPIERUNGSSCHLEIFE
0010CE  3D E1 14     0131        AND MASKE,X
0010D1  5D E0 14     0132        EOR MASKE-1,X
0010D4  F0 68        0133        BEQ FOUND          ;OPCODE GRUPPIERT
0010D6  CA           0134        DEX
0010D7  CA           0135        DEX
0010D8  10 F2        0136        BPL BEFPAR         ;PARSE WEITER
0010DA               0137 BYTE                      ;GIB CODE ALS .BYT AUS
0010DA  A2 00        0138        LDX #0
0010DC  BD 97 17     0139 BYTE1  LDA BYTTXT,X       ;AUSGABE '.BYT $'
0010DF  20 BC E9     0140        JSR OUTALL
0010E2  E8           0141        INX
0010E3  E0 06        0142        CPX #6
0010E5  90 F5        0143        BCC BYTE1
0010E7  A0 00        0144        LDY #0
0010E9  B1 10        0145        LDA (VIRTPC),Y
0010EB  48           0146        PHA
0010EC  20 46 EA     0147        JSR NUMA
0010EF  68           0148        PLA
0010F0  29 7F        0149        AND #$7F           ;GGFS DRUCKBAR MACHEN
0010F2  C9 7F        0150        CMP #$7F           ;FILTER FUER DRUCKBAR
0010F4  B0 16        0151        BCS BYTE2
0010F6  C9 21        0152        CMP #$21
0010F8  90 12        0153        BCC BYTE2
0010FA  20 C7 14     0154        JSR SPACE
0010FD  A9 27        0155        LDA #$27           ;EIN
0010FF  20 BC E9     0156        JSR OUTALL
001102  B1 10        0157        LDA (VIRTPC),Y
001104  20 BC E9     0158        JSR OUTALL
001107  A9 27        0159        LDA #$27
001109  20 BC E9     0160        JSR OUTALL
00110C  C8           0161 BYTE2  INY                ;Y=1
00110D  84 1C        0162        STY BEFLEN
00110F  4C E1 12     0163        JMP WEITER
001112               0164 ;---------------------------------
```

```
001112  A2 00        0165 WORD   LDX #0
001114  BD 9D 17     0166 WORD1  LDA WORTXT,X
001117  20 BC E9     0167        JSR OUTALL
00111A  E8           0168        INX
00111B  E0 05        0169        CPX #5
00111D  90 F5        0170        BCC WORD1
00111F  24 18        0171        BIT OPTION
001121  10 03        0172        BPL WORD2
001123  A9 48        0173        LDA #'H'
001125  2C           0174        .BYT $2C
001126  A9 24        0175 WORD2  LDA #'$'
001128  20 BC E9     0176        JSR OUTALL
00112B  A0 02        0177        LDY #2
00112D  84 1C        0178        STY BEFLEN
00112F  88           0179        DEY
001130  B1 10        0180        LDA (VIRTPC),Y
001132  20 46 EA     0181        JSR NUMA
001135  88           0182        DEY
001136  B1 10        0183        LDA (VIRTPC),Y
001138  20 46 EA     0184        JSR NUMA
00113B  4C E1 12     0185        JMP WEITER
00113E               0186 ;--------------------------------
00113E  BD 53 15     0187 FOUND  LDA JMPTBL,X      ;VEKTORIERE
001141  85 17        0188        STA SPRUNG+1
001143  CA           0189        DEX
001144  BD 53 15     0190        LDA JMPTBL,X
001147  85 16        0191        STA SPRUNG
001149  B1 10        0192        LDA (VIRTPC),Y    ;OPCODE
00114B  4A           0193        LSR A             ;OBERES HALBBYTE ADRESSIERT MN
EMONIC
00114C  4A           0194        LSR A
00114D  4A           0195        LSR A
00114E  4A           0196        LSR A
00114F  85 1A        0197        STA TEMP
001151  06 1A        0198        ASL TEMP          ;X2
001153  18           0199        CLC
001154  65 1A        0200        ADC TEMP          ;AKKU JETZT X3
001156  85 1A        0201        STA TEMP          ;ABLAGE FUER SPAETER
001158  98           0202        TYA               ;A=0
001159  C8           0203        INY               ;Y=1, DEFAULT BEFLEN
00115A  6C 16 00     0204        JMP (SPRUNG)      ;SPRUNG AUF ZWISCHENSTUFE MIT
GRX
00115D               0205 ;-----------------------------------------------
00115D               0206 ;ZUSAMMENFUEHRUNG DES PRGRAMMES AUS DEN STUFEN GRX
00115D  20 78 14     0207 ZEILE  JSR PARPAS        ;PARAMETERUEBERGABE AUSGABE DE
R ZEILE
001160  A5 1B        0208        LDA OPTION
001162  30 20        0209        BMI ZEIL3         ;KEIN OPCODE, WENN LABELAUSGAB
E
001164  A0 00        0210        LDY #0            ;FUER AUSGABE OPCODES
001166  B1 10        0211 ZEIL1  LDA (VIRTPC),Y
001168  20 46 EA     0212        JSR NUMA          ;HEXBYTE AUSGEBEN
00116B  20 C7 14     0213        JSR SPACE         ;ZWISCHENRAUM
00116E  C8           0214        INY
00116F  C4 1C        0215        CPY BEFLEN        ;FERTIG?
001171  D0 F3        0216        BNE ZEIL1
001173  C0 04        0217 ZEIL2  CPY #4            ;KOLONNEN FORMATIEREN
001175  F0 0D        0218        BEQ ZEIL3         ;OPCODES FERTIG
001177  20 C7 14     0219        JSR SPACE         ;F. 1. HEXBYTE
00117A  20 C7 14     0220        JSR SPACE         ;F. 2. HEXBYTE
00117D  20 C7 14     0221        JSR SPACE         ;F. ZWISCHENRAUM
001180  C8           0222        INY
001181  4C 73 11     0223        JMP ZEIL2
001184  20 6B 14     0224 ZEIL3  JSR MNEMDR        ;MNEMONICS AUSGEBEN
001187  E0 0A        0225        CPX #$0A          ;GRUPPE SMB/RMB?
001189  D0 03        0226        BNE MNEM1
00118B  4C 6A 12     0227 MNEM2  JMP SMBBEF
00118E  E0 14        0228 MNEM1  CPX #$14          ;GRUPPE BBS/BBC?
001190  F0 F9        0229        BEQ MNEM2
001192  20 C7 14     0230        JSR SPACE
001195  A4 1C        0231        LDY BEFLEN        ;BEFEHLSLAENGE
001197  88           0232        DEY
```

```
00119B  F0 22        0233          BEQ INHER       ;INHERENT
00119A  A5 1D        0234          LDA BEFMOD      ;ADRESSIERUNGSART
00119C  F0 2F        0235          BEQ DOLLAR      ;NORMAL
00119E  C9 7F        0236          CMP #$7F        ;PER-BEFEHL?
0011A0  D0 03        0237          BNE MNEMD2
0011A2  4C B8 12     0238          JMP PERBEF
0011A5  C9 14        0239 MNEMD2   CMP #$14        ;DIREKTOPERAND?
0011A7  F0 1F        0240          BEQ KREUZ       ;JA
0011A9  C9 40        0241          CMP #$40        ;BRANCH?
0011AB  D0 03        0242          BNE MNEMD3
0011AD  4C CB 12     0243          JMP BRANCH
0011B0  29 20        0244 MNEMD3   AND #$20        ;INDIREKT?
0011B2  F0 19        0245          BEQ DOLLAR      ;NEIN
0011B4  A9 28        0246          LDA #'('        ;FUER INDIREKT
0011B6  20 BC E9     0247          JSR OUTALL
0011B9  4C CD 11     0248          JMP DOLLAR
0011BC  A5 1D        0249 INHER    LDA BEFMOD
0011BE  10 05        0250          BPL AKK1        ;NICHT 'A'
0011C0  A9 41        0251 AKKU     LDA #'A'
0011C2  20 BC E9     0252          JSR OUTALL      ;AKKU-MODUS
0011C5  4C E1 12     0253 AKK1     JMP WEITER
0011C8  A9 23        0254 KREUZ    LDA #'#'
0011CA  20 BC E9     0255          JSR OUTALL
0011CD  20 CD 14     0256 DOLLAR   JSR DOLOUT
0011D0  20 D8 14     0257          JSR HEXOUT
0011D3  A5 1D        0258          LDA BEFMOD
0011D5  D0 03        0259          BNE HEX1
0011D7  4C E1 12     0260 HEX0     JMP WEITER      ;FERTIG
0011DA  C9 14        0261 HEX1     CMP #$14        ;ERLEDIGT AUCH FUER DIREKTOPER
AND
0011DC  F0 F9        0262          BEQ HEX0
0011DE  C9 20        0263          CMP #$20        ;NUR INDIREKT?
0011E0  D0 08        0264          BNE KOMMA
0011E2  A9 29        0265          LDA #')'        ;NUR NOCH KLAMMER ZU
0011E4  20 BC E9     0266 HEX2     JSR OUTALL
0011E7  4C E1 12     0267          JMP WEITER
0011EA  29 20        0268 KOMMA    AND #$20        ;NOCH INDIREKT?
0011EC  F0 47        0269          BEQ DIREKT      ;NEIN
0011EE  A5 1D        0270          LDA BEFMOD
0011F0  29 1C        0271          AND #$1C        ;NOCH INDICES IN DER KLAMMER?
0011F2  F0 1C        0272          BEQ POSTIN      ;INDIZIERUNG AUSSERHALB KLAMME
R
0011F4  C9 04        0273          CMP #$04        ;MIT X?
0011F6  D0 04        0274          BNE KOMMA1
0011F8  A9 58        0275          LDA #'X'
0011FA  D0 0A        0276          BNE KOMMA2
0011FC  C9 0C        0277 KOMMA1   CMP #$0C        ;MIT S?
0011FE  D0 04        0278          BNE KOMMA2
001200  A9 53        0279          LDA #'S'
001202  D0 02        0280          BNE KOMMA2
001204  A9 4C        0281 KOMMA2   LDA #'L'
001206  48          0282 KOMMA2   PHA
001207  A9 2C        0283          LDA #','
001209  20 BC E9     0284          JSR OUTALL
00120C  68          0285          PLA
00120D  20 BC E9     0286          JSR OUTALL
001210  A9 29        0287 POSTIN   LDA #')'        ;KLAMMER NOCH ZU
001212  20 BC E9     0288          JSR OUTALL
001215  A5 1D        0289          LDA BEFMOD
001217  29 03        0290          AND #$03        ;FOLGT NACHINDIZIERUNG?
001219  D0 03        0291          BNE POST1
00121B  4C E1 12     0292          JMP WEITER      ;NEIN
00121E  48          0293 POST1    PHA
00121F  A9 2C        0294          LDA #','        ;KOMMA FOLGT
001221  20 BC E9     0295          JSR OUTALL
001224  68          0296          PLA
001225  C9 02        0297          CMP #$02        ;MIT Y?
001227  D0 04        0298          BNE MITL
001229  A9 59        0299          LDA #'Y'
00122B  D0 02        0300          BNE MITL1
00122D  A9 4C        0301 MITL     LDA #'L'        ;IST ( , ),L
00122F  20 BC E9     0302 MITL1    JSR OUTALL
001232  4C E1 12     0303          JMP WEITER
```

```
001235                  0304 DIREKT                ;ADRESSIERUNG DIREKT INDIZIERT
001235  A9 2C           0305         LDA #','      ;KOMMA FOLGT
001237  20 BC E9        0306         JSR OUTALL
00123A  A5 1D           0307         LDA BEFMOD
00123C  29 1C           0308         AND #$1C      ;BESTIMME INDEX X,Y,S ODER L
00123E  C9 04           0309         CMP #$04      ;X?
001240  D0 04           0310         BNE DIRE1
001242  A9 58           0311 DIREO   LDA #'X'
001244  D0 16           0312         BNE DIRE9
001246  C9 08           0313 DIRE1   CMP #$08      ;MIT Y?
001248  D0 04           0314         BNE DIRE2
00124A  A9 59           0315         LDA #'Y'
00124C  D0 0E           0316         BNE DIRE9
00124E  C9 0C           0317 DIRE2   CMP #$0C      ;MIT S?
001250  D0 04           0318         BNE DIRE3
001252  A9 53           0319         LDA #'S'
001254  D0 06           0320         BNE DIRE9
001256  24 15           0321 DIRE3   BIT INDEX     ;AUSGABE 'LX'?
001258  30 08           0322         BMI DIREX     ;JA
00125A  A9 4C           0323         LDA #'L'
00125C  20 BC E9        0324 DIRE9   JSR OUTALL
00125F  4C E1 12        0325         JMP WEITER
001262  A9 4C           0326 DIREX   LDA #'L'
001264  20 BC E9        0327         JSR OUTALL
001267  4C 42 12        0328         JMP DIREO     ;LASS X FOLGEN
00126A                  0329 SMBBEF                ;ROCKWELL-GRUPPEN
00126A  A0 00           0330         LDY #0
00126C  B1 10           0331         LDA (VIRTPC),Y ;OPCODE NOCHMALS
00126E  C8              0332         INY
00126F  4A              0333         LSR A
001270  4A              0334         LSR A
001271  4A              0335         LSR A
001272  4A              0336         LSR A         ;MSD ISOLIEREN
001273  C9 08           0337         CMP #$08      ;OBERE UNTERE BEFEHLSGRUPPE?
001275  90 02           0338         BCC SMB9
001277  29 07           0339         AND #$07
001279  09 30           0340 SMB9    ORA #$30      ;MACH ZU ASCII
00127B  20 BC E9        0341         JSR OUTALL    ;DIREKT NACH MNEMONICS
00127E  20 C7 14        0342         JSR SPACE
001281  A5 1C           0343         LDA BEFLEN
001283  C9 02           0344         CMP #2        ;SMB?
001285  D0 03           0345         BNE BBXBEF
001287  4C CD 11        0346         JMP DOLLAR
00128A  20 CD 14        0347 BBXBEF  JSR DOLOUT
00128D  B1 10           0348         LDA (VIRTPC),Y ;ADRESSE
00128F  C8              0349         INY           ;Y=2
001290  20 46 EA        0350         JSR NUMA      ;AUSGABE
001293  20 C7 14        0351         JSR SPACE
001296  20 CD 14        0352 BBX1    JSR DOLOUT
001299  20 9D 14        0353         JSR NXTBEF    ;BASIS FUER BRANCH
00129C  A9 00           0354         LDA #0
00129E  85 15           0355         STA INDEX     ;HI BYTE=0
0012A0  B1 10           0356         LDA (VIRTPC),Y ;OFFSET
0012A2  10 02           0357 BBX2    BPL ADDI      ;VORWAERTSVERZWEIGUNG
0012A4  C6 1B           0358         DEC TEMP+1
0012A6  18              0359 ADDI    CLC
0012A7  65 1A           0360         ADC TEMP
0012A9  48              0361         PHA
0012AA  A5 1B           0362         LDA TEMP+1
0012AC  65 15           0363         ADC INDEX     ;HI BYTE
0012AE  20 46 EA        0364         JSR NUMA
0012B1  68              0365         PLA
0012B2  20 46 EA        0366         JSR NUMA
0012B5  4C E1 12        0367         JMP WEITER
0012B8  20 CD 14        0368 PERBEF  JSR DOLOUT
0012BB  20 D8 14        0369         JSR HEXOUT
0012BE  20 C7 14        0370         JSR SPACE
0012C1  A9 3B           0371         LDA #';'
0012C3  20 BC E9        0372         JSR OUTALL
0012C6  A0 02           0373         LDY #2
0012C8  4C D1 12        0374         JMP BRANI
```

```
0012CB  A5 1C        0375 BRANCH LDA BEFLEN
0012CD  C9 02        0376        CMP #2
0012CF  F0 C5        0377        BEQ BBX1          ;OFFSET IN 1 BYTE
0012D1  20 CD 14     0378 BRAN1  JSR DOLOUT        ;PRT '$'
0012D4  20 9D 14     0379        JSR NXTBEF        ;BASIS FUER OFFSET
0012D7  B1 10        0380        LDA (VIRTPC),Y
0012D9  85 15        0381        STA INDEX         ;HI BYTE
0012DB  88           0382        DEY
0012DC  B1 10        0383        LDA (VIRTPC),Y    ;LOW
0012DE  4C A6 12     0384        JMP ADDI
0012E1               0385
0012E1               0386 OPCODE                   ;HEXBYTES AUSGEBEN
0012E1  A5 1C        0387 WEITER LDA BEFLEN
0012E3  18           0388        CLC
0012E4  65 10        0389        ADC VIRTPC
0012E6  85 10        0390        STA VIRTPC
0012E8  A5 11        0391        LDA VIRTPC+1
0012EA  69 00        0392        ADC #0
0012EC  85 11        0393        STA VIRTPC+1
0012EE  4C 84 10     0394        JMP START3
0012F1               0395 ;VERARBEITUNG JE NACH BEFEHLSGRUPPE
0012F1               0396 GR0                      ;SAMMELPUNKT VERSCH BEFEHLE
0012F1  A2 00        0397        LDX #0
0012F3  4C 5D 11     0398        JMP ZEILE
0012F6               0399
0012F6               0400 GR1                      ;( ,X) UND (),Y
0012F6  C8           0401        INY
0012F7  24 15        0402        BIT INDEX
0012F9  30 04        0403        BMI GR1A
0012FB  A9 24        0404        LDA #$24          ;FUER '( ,X)'
0012FD  D0 02        0405        BNE GR1B ,IMMER
0012FF  A9 22        0406 GR1A   LDA #$22          ;FUER '(),Y'
001301  A2 02        0407 GR1B   LDX #02
001303  4C 5D 11     0408        JMP ZEILE
001306               0409 GR2                      ;() UND SONDERBEF B16
001306  24 15        0410        BIT INDEX
001308  10 02        0411        BPL GR2B
00130A  A9 20        0412        LDA #$20          ;INDIREKT () NICHT INDIZIERT
00130C  C8           0413 GR2B   INY               ;EINSPRUNG FUER ANDERE ADRESSI
ERUNGEN
00130D  A2 04        0414 GR2C   LDX #04
00130F  4C 5D 11     0415        JMP ZEILE
001312               0416 GR3                      ;(SR) UND (SR),Y  B16
001312  C8           0417        INY
001313  24 15        0418        BIT INDEX
001315  30 04        0419        BMI GR3A          ;WENN ( ,S),Y
001317  A9 0C        0420        LDA #$0C          ;FUER  ,S
001319  D0 02        0421        BNE GR3B
00131B  A9 2E        0422 GR3A   LDA #$2E
00131D  A2 02        0423 GR3B   LDX #02
00131F  4C 5D 11     0424        JMP ZEILE
001322               0425 GR4                      ;SEHR GEMISCHT
001322  24 15        0426        BIT INDEX
001324  10 02        0427        BPL GR4A          ;DIVERSE SONDERBEFEHLE
001326  A9 04        0428        LDA #$04          ;ZERO PAGE,X
001328  C8           0429 GR4A   INY               ;EINSPRUNG FUER DIVERSE
001329  A2 06        0430        LDX #06
00132B  4C 5D 11     0431        JMP ZEILE
00132E               0432 GR5                      ;ARITHM/LOG
00132E  C8           0433        INY
00132F  24 15        0434        BIT INDEX
001331  10 02        0435        BPL GR5A          ;WENN NICHT INDIZIERT
001333  A9 04        0436        LDA #$04          ;FUER ',X'
001335  A2 02        0437 GR5A   LDX #02
001337  4C 5D 11     0438        JMP ZEILE
00133A               0439 GR6                      ;VERSCHIEBEBEF
00133A  24 15        0440        BIT INDEX
00133C  10 02        0441        BPL GR6A          ;NICHT INDIZIERT
00133E  A9 04        0442        LDA #$04          ;FUER ',X'
001340  C8           0443 GR6A   INY
001341  A2 08        0444        LDX #08
001343  4C 5D 11     0445        JMP ZEILE
```

```
001346                  0446 GR7
001346  24 15           0447        BIT INDEX
001348  30 04           0448        BMI GR7A
00134A  A9 30           0449        LDA #$30
00134C  D0 02           0450        BNE GR7B
00134E  A9 32           0451 GR7A   LDA #$32
001350  C8              0452 GR7B   INY
001351  A2 02           0453        LDX #$02
001353  4C 5D 11        0454        JMP ZEILE
001356                  0455 GR8
001356  A2 0C           0456        LDX #$C
001358  4C 5D 11        0457        JMP ZEILE
00135B                  0458 GR9
00135B  C8              0459        INY
00135C  24 15           0460        BIT INDEX
00135E  10 05           0461        BPL GR9A
001360  C8              0462        INY
001361  A9 08           0463        LDA #$08
001363  D0 0D           0464        BNE GR9C
001365  A9 14           0465 GR9A   LDA #$14
001367  24 14           0466        BIT BREITE
001369  10 01           0467        BPL GR9B
00136B  C8              0468        INY
00136C  A6 1A           0469 GR9B   LDX TEMP
00136E  E0 18           0470        CPX #24
001370  F0 05           0471        BEQ GR9D
001372  A2 02           0472 GR9C   LDX #$02
001374  4C 5D 11        0473        JMP ZEILE
001377  A2 06           0474 GR9D   LDX #6
001379  86 1A           0475        STX TEMP
00137B  4C 5D 11        0476        JMP ZEILE
00137E                  0477 GRA
00137E  A5 1A           0478        LDA TEMP
001380  C9 15           0479        CMP #21
001382  B0 08           0480        BCS GRAB
001384  C9 0F           0481        CMP #15
001386  F0 04           0482        BEQ GRAB
001388  A9 B0           0483        LDA #$B0

00138A  D0 02           0484        BNE GRAC
00138C  A9 00           0485 GRAB   LDA #0
REST
00138E  A2 0E           0486 GRAC   LDX #$0E
001390  4C 5D 11        0487        JMP ZEILE
001393                  0488 GRB
001393  A2 10           0489        LDX #$10
001395  4C 5D 11        0490        JMP ZEILE
001398                  0491 GRC
001398  88              0492        DEY
001399  84 15           0493        STY INDEX
ABC
00139B  B1 10           0494        LDA (VIRTPC),Y
00139D  4A              0495        LSR A
00139E  4A              0496        LSR A
00139F  4A              0497        LSR A
0013A0  4A              0498        LSR A
0013A1  A0 03           0499        LDY #3
0013A3  C9 05           0500        CMP #5
0013A5  D0 01           0501        BNE GRCB
0013A7  C8              0502        INY
0013A8  AA              0503 GRCB   TAX
0013A9  BD A2 17        0504        LDA TABC,X
0013AC  A2 12           0505        LDX #$12
0013AE  4C 5D 11        0506        JMP ZEILE
0013B1                  0507 GRD
0013B1  C8              0508        INY
0013B2  C8              0509        INY
0013B3  24 15           0510        BIT INDEX
0013B5  10 02           0511        BPL GRDA
0013B7  A9 04           0512        LDA #$04
0013B9  A2 02           0513 GRDA   LDX #$02
0013BB  4C 5D 11        0514        JMP ZEILE
```

```
                        ;( ,L) UND ( ,L),Y 816

                        ;WENN '( ,L),Y'
                        ;FUER  '( ,L)'

                        ;IMMER IMPLIED
                        ;TEXT T8
                        ;ALWAYS
                        ;IMMEDIATE UND ABS,Y

                        ;FUER ',Y'

                        ;FUER '#'

                        ;BREITE 2 BYTE

                        ;BIT?

                        ;TEXT 'BIT'

                        ;GEMISCHT
                        ;PRUEFE AM TEXT OB AKKU-MODE
                        ;IST PLY
                        ;GROESSER
                        ;IST ES PHY?

                        ;AKKU-MODE

                        ;RESTORE NICHT INDIZIERT FUER

                        ;IMPLIED BEIM 816

                        ;ALWAYS
                        ;GEMISCHT

                        ;INDIZIERUNG NUR PER TABELLE T

                        ;LAENGE
                        ;JMP A,L?

                        ;ADRESSIERUNGSART

                        ;ARITHM/LOG

                        ;FUER ',X'
```

```
0013BE                  0515 GRE    
0013BE  C8              0516        INY
0013BF  A5 1A           0517        LDA TEMP        ;IST ES STZ?
0013C1  C9 1B           0518        CMP #27
0013C3  D0 08           0519        BNE GRE1
0013C5  C8              0520        INY
0013C6  A9 04           0521        LDA #4
0013C8  A2 12           0522        LDX #$12
0013CA  4C 5D 11        0523        JMP ZEILE
0013CD  A9 00           0524 GRE1   LDA #0
0013CF  4C 3A 13        0525        JMP GR6
0013D2                  0526 GRF                    ;ABSOLUT LANG BEIM B16
0013D2  A9 10           0527        LDA #$10        ;FUER ',L'
0013D4  A0 04           0528        LDY #4          ;LAENGE 4 BYTES
0013D6  A2 02           0529        LDX #02
0013D8  4C 5D 11        0530        JMP ZEILE
0013DB  4C F1 12        0531 GROIMP JMP GR0         ;RTI RTS BRA
0013DE  A9 14           0532 GRLDX  LDA #$14
0013E0  24 14           0533        BIT BREITE
0013E2  50 01           0534        BVC GRLDX1
0013E4  C8              0535        INY
0013E5  4C 0C 13        0536 GRLDX1 JMP GR2B
0013E8                  0537 GRCPX                  ;#
0013E8                  0538 GRCPY                  ;#
0013E8                  0539 GRLDY                  ;#
0013E8  C8              0540        INY
0013E9  A9 14           0541        LDA #$14        ;FUER '#'
0013EB  24 14           0542        BIT BREITE
0013ED  50 01           0543        BVC GRLDY1
0013EF  C8              0544        INY
0013F0  4C F1 12        0545 GRLDY1 JMP GR0
0013F3                  0546 GRBRAN                 ;BRANCHES
0013F3  C8              0547        INY
0013F4  A9 40           0548        LDA #$40        ;FUER BRANCHES
0013F6  4C F1 12        0549        JMP GR0
0013F9                  0550 GRBRK8                 ;BREAK 8 BIT
0013F9  4C F1 12        0551        JMP GR0
0013FC                  0552 GRBRK6                 ;BREAK IM B16 MODUS
0013FC  C8              0553        INY
0013FD  4C F1 12        0554        JMP GR0
001400                  0555 GRSMB                  ;SMB UND RMB BEIM C02
001400  C8              0556        INY
001401  A5 1A           0557        LDA TEMP        ;OBERER/UNTERER BEREICH?
001403  C9 18           0558        CMP #24
001405  B0 04           0559        BCS GRSMB9      ;SMB-GRUPPE
001407  A9 00           0560        LDA #0          ;2. TEXTITEM
001409  F0 02           0561        BEQ GRSMB2
00140B  A9 03           0562 GRSMB9 LDA #3
00140D  85 1A           0563 GRSMB2 STA TEMP
00140F  A9 00           0564        LDA #0
001411  A2 0A           0565        LDX #$0A
001413  4C 5D 11        0566        JMP ZEILE
001416                  0567 GRBBS                  ;BBS UND BBC BEIM C02
001416  A0 03           0568        LDY #3
001418  A5 1A           0569        LDA TEMP
00141A  C9 18           0570        CMP #24
00141C  B0 04           0571        BCS GRBBS1
00141E  A9 03           0572        LDA #3
001420  D0 04           0573        BNE GRBBS2
001422  A9 00           0574 GRBBS1 LDA #0
001424  A9 00           0575        LDA #0
001426  85 1A           0576 GRBBS2 STA TEMP
001428  A9 DF           0577        LDA #$DF        ;FUER BRANCH ON BIT SET
00142A  A2 14           0578        LDX #$14        ;FUER TEXT TF
00142C  4C 5D 11        0579        JMP ZEILE
00142F  4C 22 13        0580 GRBIT  JMP GR4         ;FUER 6502
001432                  0581 GRJSR
001432  A0 03           0582        LDY #3
001434  4C F1 12        0583        JMP GR0
001437  A9 14           0584 GRSEP  LDA #$14
001439  4C 0C 13        0585        JMP GR2B        ;BEFEHLE REP UND SEP
00143C  A0 03           0586 GRJSL  LDY #3          ;LAENGE3+1
00143E  4C 0C 13        0587        JMP GR2B
```

```
001441  A9 7F        0588 GRPER  LDA #$7F ;MERKMAL PER
001443  A0 02        0589        LDY #2
001445  4C 0C 13     0590        JMP GR2B
00144B  A9 40        0591 GRBRL  LDA #$40 ,FUER BRANCH
00144A  C8           0592        INY
00144B  4C 0C 13     0593        JMP GR2B
00144E  C8           0594 GRMVX  INY                 ;FUER MVN MVP
00144F  4C 28 13     0595        JMP GR4A
001452  4C 28 13     0596 GRPEI  JMP GR4A
001455  C8           0597 GRPEA  INY
001456  4C 28 13     0598        JMP GR4A
001459  C8           0599 GRTRBA INY
00145A  4C 28 13     0600 GRTRB  JMP GR4A
00145D  4C 0D 13     0601 GRWDM  JMP GR2C
001460  A9 08        0602 GRLDXZ LDA #$08            ;LDX/STX ZEROPAGE,Y
001462  4C 40 13     0603        JMP GR6A
001465  A9 08        0604 GRLDXA LDA #$08            ;LDX ABS,Y
001467  C8           0605        INY
001468  4C 40 13     0606        JMP GR6A
00146B                0607 ;-----------------------------------------------
00146B  A0 00        0608 MNEMDR LDY #0              ;AUSGABE 3 ZCH MENOMICS
00146D  B1 1A        0609 MNEMD1 LDA (TEMP),Y
00146F  20 BC E9     0610        JSR OUTALL
001472  C8           0611        INY
001473  C0 03        0612        CPY #3
001475  D0 F6        0613        BNE MNEMD1          ;MEHR ZEICHEN
001477  60           0614        RTS
001478  84 1C        0615 PARPAS STY BEFLEN          ;LAENGE, EINSPRUNG LANGE LISTE
N
00147A  85 1D        0616        STA BEFMOD          ;ADRESSIERUNGSART
00147C  18           0617        CLC
00147D  BD B1 17     0618        LDA TXTADR,X        ;TEXTBEGINN
0014B0  65 1A        0619        ADC TEMP
001482  85 1A        0620        STA TEMP
001484  BD B2 17     0621        LDA TXTADR+1,X      ;HI BYTE
001487  69 00        0622        ADC #0
001489  85 1B        0623        STA TEMP+1
00148B  60           0624        RTS
00148C                0625 ;-----------------------------------------------
00148C  B9 B2 17     0626 TXTOUT LDA TEXTE,Y         ;AUSGABE BENUTZERFUEHRUNG
00148F  08           0627        PHP
001490  C8           0628        INY
001491  20 BC E9     0629        JSR OUTALL
001494  28           0630        PLP
001495  10 F5        0631        BPL TXTOUT
001497  A9 0D        0632        LDA #$0D
001499  20 BC E9     0633        JSR OUTALL
00149C  60           0634        RTS
00149D  18           0635 NXTBEF CLC                 ;BASIS FUER BRANCHES BESTIMMEN
00149E  A5 1C        0636        LDA BEFLEN
0014A0  65 10        0637        ADC VIRTPC
0014A2  85 1A        0638        STA TEMP ,ZWISCHENSPEICHER
0014A4  A5 11        0639        LDA VIRTPC+1
0014A6  69 00        0640        ADC #0
0014A8  85 1B        0641        STA TEMP+1
0014AA  60           0642        RTS
0014AB  A5 18        0643 PCOUT  LDA OPTION
0014AD  F0 05        0644        BEQ PCOUTB
0014AF  A9 48        0645        LDA #'H'
0014B1  20 BC E9     0646        JSR OUTALL
0014B4  24 13        0647 PCOUTB BIT MODE            ;CPU 65816?
0014B6  10 05        0648        BPL PCOUT1
0014B8  A5 12        0649        LDA BANK
0014BA  20 46 EA     0650        JSR NUMA
0014BD  A5 11        0651 PCOUT1 LDA VIRTPC+1        ;AUSGABE DES VIRTPC
0014BF  20 46 EA     0652        JSR NUMA
0014C2  A5 10        0653        LDA VIRTPC
0014C4  20 46 EA     0654        JSR NUMA
0014C7  A9 20        0655 SPACE  LDA #' '            ;SPACE
0014C9  20 BC E9     0656        JSR OUTALL
0014CC  60           0657        RTS
```

```
0014CD  A9 24        0658 DOLOUT LDA #'$'          ;AUSGABE
0014CF  24 18        0659        BIT OPTION
0014D1  10 02        0660        BPL DOL1
0014D3  A9 48        0661        LDA #'H'          ;AUSGABE IN LABEL-FORM
0014D5  4C BC E9     0662 DOL1   JMP OUTALL
0014D8  B1 10        0663 HEXOUT LDA (VIRTPC),Y    ;GET OPERAND
0014DA  20 46 EA     0664        JSR NUMA          ;AUSGABE
0014DD  88           0665        DEY
0014DE  D0 F8        0666        BNE HEXOUT        ;NUR DIE BYTES DES OPERANDEN!
0014E0  60           0667        RTS
0014E1               0668 ;-------------------------------------------------
0014E1               0669 MASKE                     ;EOR- UND AND-OPERATOREN F=R B
EFEHLSGRUPPEN
0014E1  01 0F        0670        .DBY $010F,$020F,$22FF,$62FF,$B2FF,$42FF ;LSD1/
2 JSL PER
0014E3  02 0F        0670
0014E5  22 FF        0670
0014E7  62 FF        0670
0014E9  B2 FF        0670
0014EB  42 FF        0670
0014ED  03 0F        0671        .DBY $030F,$040F,$050F,$060F,$070F ;LSD 3-7
0014EF  04 0F        0671
0014F1  05 0F        0671
0014F3  06 0F        0671
0014F5  07 0F        0671
0014F7  44 EF        0672        .DBY $44EF,$D4FF,$F4FF ;MVX PEI PEA
0014F9  D4 FF        0672
0014FB  F4 FF        0672
0014FD  08 0F        0673        .DBY $080F,$090F,$0A0F,$0B0F,$0C0F,$0D0F,$0E0F,
$0F0F ;8-F
0014FF  09 0F        0673
001501  0A 0F        0673
001503  0B 0F        0673
001505  0C 0F        0673
001507  0D 0F        0673
001509  0E 0F        0673
00150B  0F 0F        0673
00150D  00 1F        0674        .DBY $001F,$E0FF,$C0FF,$A2FF,$A0FF ;0-IMPL CPX/
Y LDX/Y #
00150F  E0 FF        0674
001511  C0 FF        0674
001513  A2 FF        0674
001515  A0 FF        0674
001517  96 DF        0675        .DBY $96DF,$BEFF ;LDX/STX,Y
001519  BE FF        0675
00151B  C2 DF        0676        .DBY $C2DF,$1CFF,$14FF
00151D  1C FF        0676
00151F  14 FF        0676
001521  80 FF        0677        .DBY $80FF,$101F,$20FF,$00FF ;BRAN JSR BRK ;EIN
SATZ /816
001523  10 1F        0677
001525  20 FF        0677
001527  00 FF        0677
001529  5C FF        0678        .DBY $5CFF,$DCDF,$D4DF,$021F,$A2FF ;L=CKEN USW.
 C02
00152B  DC DF        0678
00152D  D4 DF        0678
00152F  02 1F        0678
001531  A2 FF        0678
001533  00 FF        0679        .DBY $00FF,$070F,$0F0F,$030F,$0B0F ;BRK S/RMB 2
VERBOTENE
001535  07 0F        0679
001537  0F 0F        0679
001539  03 0F        0679
00153B  0B 0F        0679
00153D  04 8F        0680        .DBY $048F,$24FF,$9EFF,$1C9F,$0CFF,$DADF ;VERBO
TENE BIT
00153F  24 FF        0680
001541  9E FF        0680
001543  1C 9F        0680
001545  0C FF        0680
001547  DA DF        0680
```

```
001549  1A 9F          0681          .DBY $1A9F,$80FF,$020F,$A2FF,$0303 ;VERBOTENE L
DX
00154B  80 FF          0681
00154D  02 0F          0681
00154F  A2 FF          0681
001551  03 03          0681
001553                 0682 ;-----------------------------------------------
001553                 0683 JMPTBL                    ;SPRUNGVERTEILER
001553  F6 12          0684          .WOR GR1,GR2,GRJSL,GRPER,GRBRL,GRWDM,GR3,GR4,GR
5,GR6,GR7
001555  06 13          0684
001557  3C 14          0684
001559  41 14          0684
00155B  48 14          0684
00155D  5D 14          0684
00155F  12 13          0684
001561  22 13          0684
001563  2E 13          0684
001565  3A 13          0684
001567  46 13          0684
001569  4E 14          0685          .WOR GRMVX,GRPEI,GRPEA
00156B  52 14          0685
00156D  55 14          0685
00156F  56 13          0686          .WOR GR8,GR9,GRA,GRB,GRC,GRD,GRE,GRF
001571  5B 13          0686
001573  7E 13          0686
001575  93 13          0686
001577  98 13          0686
001579  B1 13          0686
00157B  BE 13          0686
00157D  D2 13          0686
00157F  DB 13          0687          .WOR GROIMP,GRCPX,GRCPY,GRLDX,GRLDY
001581  EB 13          0687
001583  E8 13          0687
001585  DE 13          0687
001587  EB 13          0687
001589  60 14          0688          .WOR GRLDXZ,GRLDXA ;LDX/STX,Y
00158B  65 14          0688
00158D  37 14          0689          .WOR GRSEP,GRTRBA,GRTRB
00158F  59 14          0689
001591  5A 14          0689
001593  F3 13          0690          .WOR GRBRAN,GRBRAN,GRJSR,GRBRK6 ;EINSATZPUNKT/8
16
001595  F3 13          0690
001597  32 14          0690
001599  FC 13          0690
00159B  DA 10          0691          .WOR BYTE,BYTE,BYTE,BYTE,GRLDX
00159D  DA 10          0691
00159F  DA 10          0691
0015A1  DA 10          0691
0015A3  DE 13          0691
0015A5  F9 13          0692          .WOR GRBRK8,GRSMB,GRBBS,BYTE,BYTE ;EINSATZPUNKT
C02
0015A7  00 14          0692
0015A9  16 14          0692
0015AB  DA 10          0692
0015AD  DA 10          0692
0015AF  DA 10          0693          .WOR BYTE,GRBIT,BYTE,BYTE,BYTE,BYTE
0015B1  2F 14          0693
0015B3  DA 10          0693
0015B5  DA 10          0693
0015B7  DA 10          0693
0015B9  DA 10          0693
0015BB  DA 10          0694          .WOR BYTE,BYTE,BYTE,GRLDX,BYTE ;EINSATZPUNKT 65
02
0015BD  DA 10          0694
0015BF  DA 10          0694
0015C1  DE 13          0694
0015C3  DA 10          0694
0015C5                 0695 ;-----------------------------------------------
0015C5                 0696 MNEMOS                    ;BEFEHLSTEXTE, JEWEILS 3 BYTE
```

```
0015C5  42 52        0697 T0     .BYT 'BRKBPLJSRBMIRTIBVCRTSBVSBRABCCLDYBCSCPYBN
ECPXBEQ'
0015F5  4F 52        0698 T1     .BYT 'ORAORAANDANDEORBORADCADCSTASTALDALDACMPCM
PSBCSBC'
001625  43 4F        0699 T2     .BYT 'COPORAJSLANDWDMEORPERADCBRLSTALDXLDAREPCM
PSEPSBC'
001655  54 53        0700 T4     .BYT 'TSBTRBBITBITMVPMVNSTZSTZSTYSTYLDYLDYCPYPE
ICPXPEA'
001685  41 53        0701 T6     .BYT 'ASLASLROLROLLSRLSRRORRORSTXSTXLDXLDXDECDE
CINCINC'
0016B5  52 4D        0702 T7     .BYT 'RMBSMB'
0016BB  50 48        0703 T8     .BYT 'PHPCLCPLPSECPHACLIPLASEIDEYTYATAYCLVINYCL
DINXSED'
0016EB  41 53        0704 TA     .BYT 'ASLINCROLDECLSRPHYRORPLYTXATXSTAXTSXDEXPH
XNOPPLX'
00171B  50 48        0705 TB     .BYT 'PHDTCSPLDTSCPHKTCDRTLTDCPHBTXYPLBTYXWAIST
PXBAXCE'
00174B  54 53        0706 TC     .BYT 'TSBTRBBITBITJMPJMPJMPJMPSTYSTZLDYLDYCPYJM
LCPXJSR'
00177B  42 42        0707 TF     .BYT 'BBSBBR'
0017B1               0708 ;-----------------------------------------------------
----
0017B1               0709 TXTADR                 ;TEXTBEGINNE MNEMON WORTGRUPPE
N TO-TF
0017B1  C5 15        0710        .WOR T0,T1,T2,T4,T6,T7,T8,TA,TB,TC,TF
0017B3  F5 15        0710
0017B5  25 16        0710
0017B7  55 16        0710
0017B9  85 16        0710
0017BB  B5 16        0710
0017BD  BB 16        0710
0017BF  EB 16        0710
0017C1  1B 17        0710
0017C3  4B 17        0710
0017C5  7B 17        0710
0017C7               0711 ;-----------------------------------------------------
----
0017C7  2E 42        0712 BYTTXT .BYT '.BYT $'     ;AUSGABE BEI UNG=LTIGEN OPCOD
ES
0017CD  2E 57 4F 52 20 0713 WORTXT .BYT '.WOR '      ;AUSGABE IM WOR-FORMAT

0017A2               0714 ;ADRESSIERUNGSARTEN DER GRUPPE C, IM AKKU UEBERGEBEN
0017A2  00           0715 TABC   .BYT 0,0,0,4,0,$10,$20,$24,0,0,0,4,0,$20,0,$24
0017A3  00           0715
0017A4  00           0715
0017A5  04           0715
0017A6  00           0715
0017A7  10           0715
0017A8  20           0715
0017A9  24           0715
0017AA  00           0715
0017AB  00           0715
0017AC  00           0715
0017AD  04           0715
0017AE  00           0715
0017AF  20           0715
0017B0  00           0715
0017B1  24           0715
0017B2               0716
0017B2               0717 TEXTE                  ;TEXTBASIS
0017B2  0D           0718 TEXT2  .BYT $0D,'WELCHE CPU? '
0017B3  57 45        0718
0017BF  43 52        0719        .BYT 'CR=6502 R=R65C02 G=65816'$
0017D7  57 45        0720 TEXT3  .BYT 'WELCHE REGISTERBREITE?',$0D
0017ED  0D           0720
0017EE  41 3D        0721 TEXT4  .BYT 'A=AKKU 16 BIT,     INDICES 8 BIT',$0D
00180D  0D           0721
00180E  58 3D        0722        .BYT 'X=INDICES 16 BIT, AKKU 8 BIT',$0D
00182A  0D           0722
00182B  43 52        0723 TEXT5  .BYT 'CR=ALLE REGISTER=16, ELSE ALLES 8 BIT'$
001850  43 52        0724 TEXT9  .BYT 'CR=NORMALE AUSGABE. ELSE MIT LABELN'$
```

```
001873  43 52          0725 TEXT10 .BYT 'CR=NORMAL- B=.BYT-  W=.WORD-AUSGABE'$
001896                 0726        .EFI
001896                 0727        .END
 ERRORS=0000
```

Die nachfolgenden Anwendungsbeispiele beziehen sich auf Stellen in diesem Programm

```
<1>OFF
<ON Ä>FROM=1553TO=1560
WELCHE CPU? CR=6502 R=R65C02 G=65816
CR=NORMALE AUSGABE, ELSE MIT LABELN
CR=NORMAL- B=.BYT-  W=.WORD-AUSGABE

H1553 .WOR H12F6
H1555 .WOR H1306
H1557 .WOR H143C
H1559 .WOR H1441
H155B .WOR H1448
H155D .WOR H145D
H155F .WOR H1312
```

Ausgabe mit Kunstlabeln

```
<Ä>FROM=1000TO=1016
WELCHE CPU? CR=6502 R=R65C02 G=65816
CR=NORMALE AUSGABE, ELSE MIT LABELN
CR=NORMAL- B=.BYT-  W=.WORD-AUSGABE

1000 20 A3 E7    JSR $E7A3
1003 AD 1C A4    LDA $A41C
1006 85 10       STA $10
1008 AD 1D A4    LDA $A41D
100B 85 11       STA $11
100D 20 A7 E7    JSR $E7A7
1010 A0 00       LDY #$00
1012 84 14       STY $14
1014 84 18       STY $18
```

Normale Ausgabe

```
WELCHE CPU? CR=6502 R=R65C02 G=65816
CR=NORMALE AUSGABE, ELSE MIT LABELN
CR=NORMAL- B=.BYT-  W=.WORD-AUSGABE

15C5 .BYT $42 'B'
15C6 .BYT $52 'R'
15C7 .BYT $4B 'K'
15C8 .BYT $42 'B'
15C9 .BYT $50 'P'
15CA .BYT $4C 'L'
15CB .BYT $4A 'J'
15CC .BYT $53 'S'
15CD .BYT $52 'R'
15CE .BYT $42 'B'
15CF .BYT $4D 'M'
```

Ausgabe im Format .BYT

```
<Ä>FROM=1000TO=1016
WELCHE CPU? CR=6502 R=R65C02 G=65816
WELCHE REGISTERBREITE?
A=AKKU 16 BIT,    INDICES 8 BIT
X=INDICES 16 BIT, AKKU 8 BIT
CR=ALLE REGISTER=16, ELSE ALLES 8 BIT
CR=NORMALE AUSGABE, ELSE MIT LABELN
CR=NORMAL- B=.BYT-  W=.WORD-AUSGABE

001000 20 A3 E7    JSR $E7A3
001003 AD 1C A4    LDA $A41C
001006 85 10       STA $10
001008 AD 1D A4    LDA $A41D
00100B 85 11       STA $11
00100D 20 A7 E7    JSR $E7A7
001010 A0 00 84    LDY #$8400
001013 14 84       TRB $84
001015 18          CLC
```

Ausgabe im Format 'G' für 65816

Roland Löhr

Konzepte der CPU's 320xx

Übersicht

Nachdem in den früheren Heften immer wieder auf interessante Weiterentwicklungen in den CPU-Familien 6xxxx hingewiesen wurde, soll hier auf die Familie 32032 von National Semiconductor eingegangen werden, die drei moderne CPU-Typen mit ebenfalls moderner Architektur umfaßt. Die Unterschiede liegen vor allem in der Breite des Datenbusses: 32008 (8 Bit), 32016 (16) und 32032 (32). Intern haben alle die gleiche Breite der Hauptregister, nämlich 32 Bit. Die Befehlssätze sind vollkommen kompatibel. Und bei den mit einem Displacement arbeitenden Adressierungsformen sind bei ihnen allen Offsets in bis zu 32 Bit möglich. Alle haben einen durchgehenden Adressenraum von 16 MB mit teilweise gemultiplextem Adressen-/Datenbus und können mit einer optionalen Fließkommaeinheit (32081) betrieben werden, ebenso mit anderen angepaßten Beschaltungen als Slave Processor. Die Typen 32016 und 3203 lassen daneben als Slave Processor eine Memory Management Unit vom Typ 32082 zu, die Speicherzugriffe nach dem Prinzip 'Demand Paged' erlaubt, auf das wir noch kommen werden. Alle erwähnten Bausteine sind bereits lieferbar, auch die benötigte Timing Control Unit (32201) und die Interrupt Control Unit 32202.

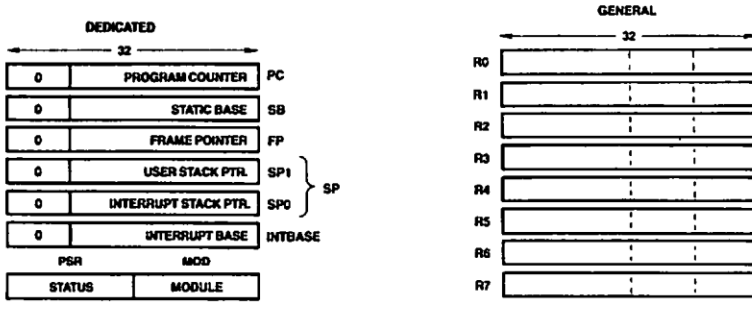
Wenn man davon absieht, daß der 32008 keine MMU bedient, so darf man sagen, daß diese CPU-Familie ein durchgehendes und kompatibles Konzept verwirklicht: Einheitlicher Befehlssatz, gleiche Register, Anschluß einer Fließkommaeinheit und virtuelle Speicherverwaltung. Demgegenüber sind bei 68xxx die höheren Attribute erst ab 68010 (virtuelle Maschine) und 68020 verwirklicht. Der Befehlssatz ist hier aufwärtskompatibel.

Das Registermodell

Dem Anwender stehen 8 allgemeine Register zur Verfügung (R0-R7), die als Adreß- und Datenregister benutzt werden können. Systemregister sind der Programmzähler PC, ein Anwender- (SP1) und ein Interrupt-Stackpointer (SP0), ein Basisregister für Interrupte (INTBAS) und ein Frame Pointer auf Variable auf dem Stack. Das Static Base Register wird benutzt, um auf globale Variable eines Moduls zu zeigen und das Module Register zeigt auf den Deskriptor des zur Zeit in Arbeit befindlichen Moduls, der nach bestimmten Regeln aufgebaut sein muß. Im Status des Prozessors sind zehn Bits benutzt, die u.a. den privilegierten Status beinflussen (gewisse Zugriffe durch Anwenderprogramme sind ausgeschlossen) und die Ergebnisse von vorzeichenfreien oder vorzeichenbehafteten Vergleichen widerspiegeln, wenn man von den üblichen Bedingungsbits absieht. Interessant ist auch das Configuration Register, in dem der CPU per Befehl mitgeteilt wird, ob extern eine Interrupt Control Unit, eine MMU, eine Fließkommaeinheit oder ein Custom Slave Processor angeschlossen ist. Wenn die Bits hier nicht gesetzt sind, so werden entsprechende Befehle auf ein Trap geführt, so daß die Leistungen solcher Bausteine in Software emuliert werden können. So ist es bei der Entwicklung von Systemen mit einem einzigen Befehl auf den jeweiligen Ausrüstungsstand Rücksicht zu nehmen.

Adressierungsarten und Datentypen

Dem Leser werden die nachfolgenden Prinzipien bei der Bildung der effektiven Adressen von Operanden im wesentlichen bekannt sein: Der Operand ist in einem der allgemeinen Register R0-R7 (bei gewissen Befehlen für Slave-Prozessoren können stattdessen 8 andere Register angesprochen werden). Das Register kann eine Basisadresse enthalten, zu dem ein Displacement (8, 16, 32 Bit) hinzugerechnet wird. Bei dieser Adressierungsart können auch die Zeigerregister PC, SP, SB oder FP die Basis bilden. Es kann zu einem



Registermodell NS 32032

indirekten Pointer im Speicher ein Offset addiert werden. Immmediate und absolut sprechen für sich. Daneben gibt es Top of Stack, External (über eine Link-Tafel) und den Scaled Index, den wir auch beim 68020 erwähnten. Hier wird eine nach den vorerwähnten Methoden gebildete effektive Adresse (außer immdediate) mit dem Inhalt eines der allgemeinen Register nach-indiziert. Scaled heißt, das der Inhalt des Indexregisters zunächst mit 1, 2, 4 oder 8 multipliziert wird, ehe die Addition erfolgt. Auf diese Weise kann man Arrays bequem abrastern.

Als Datentypen werden Byte, Wort, Doppelwort, Fließkomma (32 Bit mit Mantisse in 23 Bit) und Fließkomma lang (64 Bit mit 52 Bit für Mantisse) unterstützt. Der Befehlssatz läßt aber auch die Bearbeitung von Bitfeldern vor, die nicht an Byte-Grenzen ausgerichtet sind, ähnlich wie wir es für den 68020 beschrieben.

Grundausrüstung abweichen. Zunächst: Die CPU's sind Zwei-Adreßmaschinen. Bei den Moves finden wir auch Move Effective Adress, und Block Move. Im Bereich Arithmetik sind Multiplikation und Division enthalten, auch in Spielarten mit Runden oder für den Rest. Additionen und Subtraktionen können auch mit BCD-Zahlen durchgeführt werden. Vergleiche sind über Bytefelder (multiple) möglich. Es gibt Befehle für das Testen und Setzen/Invertieren von Bits. Bei Arrays kann gegen Bereichsgrenzen geprüft werden. Leistungsfähige Befehle wurden extra für die Stringbearbeitung geschaffen.

Verzweigungen können als Mehrwegeverzweigungen angelegt werden. Für Aufrufprozeduren hat es eigene Befehle, auch für die Benutzung von Deskriptoren und die Bereitstellung von Stackframes, für das Retten der Register. Drei Gruppen von Befehlen decken daneben die Behandlung einer Fließkommaeinheit, einer MMU und eins Custom Slave ab.- In der Summe kann man den Befehlssatz als sehr vielseitig und leistungsfähig bezeichnen.

Virtuelle Maschine mit Demand Paging

Das Prinzip der virtuellen Maschine besteht darin, mit einem kleineren wirklichen Speicher zu arbeiten, als es dem logisch angesprochenen Adreßraum entspricht. Dazu müssen logische Adressen auf solche umgesetzt werden, die es auf der Maschine tatsächlich gibt. Dem dient eine Memory Management Unit, die aus der ihr mitgeteilten Verwaltungsinformation der Zuordnung entsprechende Speicher anspricht und die Datenblöcke von draußen nachzieht, wenn diese noch nicht im Speicher sind. Andererseits muß sie Datenblöcke auslagern, wenn es für das Nachziehen zu eng würde.

Man kann zwei Verwaltungsprinzipien unterscheiden, die Segmentierung und das Demand Paging. Bei der Segmentierung werden ganze Modulpakete in zusammenhängende Bereiche ein- und ausgelagert, was ggfs. erhebliche Transporte nach sich zieht. Dabei kann es vorkommen, daß einmal nicht genug zusammenhängender Speicherraum für ein benötigtes großes Modul zur Verfügung steht, obwohl die Summe der freien Lücken theoretisch ausreichen würde. In diesem Falle hat man weitere Umladevorgänge.

Demand Paging geht bei NS32032 von grundsätzlich kleinen Segmenten von 512 Bytes aus, die den Benutzern oder den Tasks nach Bedarf dort zugewiesen werden, wo sie frei sind, auch in zersplitterten Adreßbereichen, die entsprechend logisch auf physisch umgesetzt werden. Es kann damit nicht zu Konflikten wegen der Modulgrößen kommen und (je nach Aufgabe) dürfte auch die Menge der notwendigen Ein- und Austransporte geringer sein.

Zusammenfassung
Diese Beschreibung sollte einen ersten Überblick über eine Prozessorfamilie geben, die auf anderen Wegen als bei Motorola und Intel moderne Konzepte in durchgehender interner 32 Bit Struktur verwirklicht. Nach außen hin hat sie einheitlich einen Adressenraum in 24 Bit und je nach Wahl Datenbusse von 8, 16 und 32 Bit. Peripheriebausteine werden wie Speicherzellen angesprochen, und heute schon sind Fließkommaarithmetik und die Verwaltung virtueller Speicherräume geregelt. Es lassen sich, wie es scheint, auch preiswerte Aufbauten erzielen. Dem aktuellen Angebot eines Distributors sind folgende Preise entnommen: CPU's 6 MHz je nach Typ 160/150/485 DM, FPU DM 375, MMU DM 365, TCU DM 42 und ICU DM 215.

Neben den Bauteilen sind natürlich auch Entwicklungssysteme und Software von großer Bedeutung. In diesem Heft wird das Development Boad DB 16000 mit dem TDS besprochen. Über die weiteren Hilfen unterrichte man sich in den Firmenschriften: Multi-User Development System, PASCAL Compiler, C-Compiler, Real Time Multitasking Executive, GENIX Betriebssystem als Abkömmling von UNIX sowie über die Emulatoren.

Roland Löhr

Development Board

DB 16000 von National Semiconductor

Beim Herausgeber konnte vorübergehend dieser etwa 17x30 cm messende Einplatinenconputer erprobt werden. Seine wesentliche Ausrüstung ist: CPU 16032-6, Betriebssystem TDS (Tiny Development System) auf den Steckplätzen für 16k Festwertspeicher, 128K DRAM und ein mit normalerweise 9600 Baud betriebener serieller Port, der durch Schalter allerdings auch in anderen Geschwindigkeiten betrieben werden kann. Daneben finden sich Steckplätze für eine Memory Management Unit, für einen Fließkomma-Prozessor und für eine Interrupt Control Unit (siehe in diesem Heft). Natürlich ist für den Systemtakt auch eine Timing Control Unit 16201 vorhanden, ferner ein herausgeführter Interfacebbaustein 8255 mit 24 E/A-Portleitungen (3 Ports). Auf die Kontaktleisten P1 (86 Zungen) und P2 (60 Zungen) ist ein Multibus-Interface gelegt, das es dem Board ermöglicht, auf ein für diesen Bus kompatibles System als Master zuzugreifen. Zwei Buchsenleisten mit 44 Kontakten können Expansions-Module aufnehmen. Das begleitende User's Manual (ca. 80 S.) beschreibt die einzelnen Einheiten, die Kontaktbelegungen und die Konfigurationsmöglichkeiten durch Kontaktbrücken.

Das Board kann an einem Terminal betrieben werden oder an einem Host-Computer, auch im sog. Transparent Mode. Das Handbuch beschreibt die Inbetriebnahme, die dann auch problemlos erfolgt, wenn man auch die Spannungsversorgung mit 5V, +12/-12V vorhält. Nach Betätigen der Reset-Taste meldet sich dann das Board auf dem Terminal. - Mehr eigentlich als die Hardware, die man mit MMU, FPU und ICU, mit Multibus- und Expansions-Modulen zu einem leistungsfähigen größeren Entwicklungssystem ausbauen kann, interessierte den Autor die Handhabung unter dem Betriebssystem, wovon jetzt berichtet wird.

TDS - Tiny Development System
Dieses Betriebsprogramm in 16 KB, das sämtliche vorhandenen vier Steckplätze für Festwertspeicher belegt, hat drei Dienstleistungsbereiche, nämlich Editor, Assembler und symbolischen Debugger. Die Dokumentation (User's Manual, ca. 50 S.) beschreibt die Inbetriebnahme und mit Beispielen auch die zur Verfügung stehenden Befehle. Ein Reset bewirkt offensichtlich keine vollständige Initialisierung, so daß empfohlen wird, zunächst den Befehl IT zu geben. Danach kann man mit 'OM' auch das Verhalten nach 'CR' beeinflussen (ob mit oder ohne 'LF' geechot wird).

Man hat für die einzelnen Teilprogramme merkwürdigerweise keine eigenen Befehlsebenen geschaffen. So mag ein Befehl den Editor betreffen, der nächste den Debugger.

Der Editor ist von der einfachsten Sorte. Es gibt 5 Kommandos, nämlich IN (für Insert Line), TP (Type, zeige Zeile oder eine Reihe von Zeilen), RP (Replace, ersetze eine Zeile), KI (Kill, lösche eine Zeile) und RS (Reset, lösche alle Zeilen). Bei der Eingabe eine Zeile muß man nach dem Prompt des Systems (*) zunächst den Befehl 'IN' geben, damit das Folgende als Eingabezeile erkannt wird. Das ist eigentlich lästig und hängt damit zusammen, daß man eben keine getrennten Befehlsebenen gibt. - Wenn man auch Textbereiche mit TP und Parametern auslisten kann, so werden doch schmerzlich Rangierbefehle vermißt und auch Such- und Ersetzungsfunktionen, wie wir sie seit langen Jahren z.B. im Editor des AIM 65 benutzen können. Der Editor des TDS ist zeilenmummern-orientiert.

Der Assembler: Es ist eine abgemagerte Version voller Assembler. Einige Befehle sind nicht implementiert und müssen per .Byte in hex codiert werden. Das betrifft auch die mnemonische Ansprache von Registern, die von R0-R7 verschieden sind. Die einzige arithmetische Operation zwischen Symbolen ist die Subtraktion. Symbolnamen dürfen nur 2 Byte umfassen, und die Prüfung auf Eingabefehler ist nicht vollständig. Für Pass 1 gibt es sechs mögliche Fehlermeldungen und für Pass 2 vier. Es wird mit Pass 1 abgebrochen, wenn noch Fehler vorhanden sind. Der Assembler wird mit 'AS' hereingerufen und einer Startadresse. Nach erfolgreichem Lauf muß das. Anwenderprogramm noch manuell oder im beschriebenen Verfahren automatisch in Kraft gesetzt werden.

Der Debugger stellt innerhalb des TDS die meisten Befehle zur Verfügung, wenn auch nicht alle erwünschten. Hier wird vor allem das Fehlen eines Disassemblers beklagt. Die Befehle kann man grob in drei Gruppen einteilen: Anzeige/Ausgabe der CPU-Register, der Register von SlaveProzessoren und von Speicherbereichen, dito das Ändern derselben. Das kann auch in symbolischer Form geschehen, wenn ein erfolgreicher Assemblerlauf stattfand. Die dritte Gruppe betrifft Hilfen für die Ausführungszeit und verschiedenes. Man kann Breakpoints setzen und tracen, n Befehle oder solange/bis Bedingungen erfüllt sind. Anwenderprogramme können als Unterprogramme aufgerufen werden, ebenso externe Unterprogramme. Die verschiedenen Kommandos betreffen Block Move, Fill memory, Search Memory for Data und den hexadezimalen Dump von Speicherinhalten. Bei Fehlansprächen wird eine von 16 Fehlermeldungen ausgegeben.

Eine gewünschte Massenspeicherung ist mit einem im Handbuch beschriebenen Interface und zwei Befehlen für die Textspeicherung und -eingabe auf Tonband möglich.

Zusammenfassend meint der Autor, daß das Development Board mit Steckplätzen für nur 16 KB Festwertprogramm in diesem Punkt zu klein bemessen worden ist. Immerhin stehen 128 KB RAM daneben zur Verfügung. Diese Beschränkung führt dann zu einer Beschränkung der Dienstleistungen, die im Tiny Development System zur Verfügung gestellt werden können. Als schmerzlich werden hier, wie erwähnt, ein leistungsfähigerer Texteditor, ein Assembler mit mehr Komfort und ein Disassembler vermißt, die sich integrieren ließen, wenn man die Steckplätze für EPROMs vom Typ 2764 oder 27128 in der Kapazität verdoppeln oder vervierfachen würde. In dieser Umgebung kommen auf den Systemprogrammierer noch zuviele Aufgaben zu, die er selber lösen muß, wenn er sich nicht von Anfang an bei Mehrausgaben für eine großzügere Arbeitsbasis entscheidet.

Roland Löhr

Der LASER 3000

Personal Computer mit 6502A

Beim Autor konnte ein Apple-kompatibler Computer mit vorstehender Bezeichnung getestet werden. Es handelt sich um ein Erzeugnis der Video Technology in Hong Kong, das hierzulande von der Firma Sanyo Video in Hamburg vertrieben wird. Der Computer bildet eine Konsole, die die Tastatur und die Elektronik enthält. Zum Betrieb muß man entweder einen Video-Monitor in s/w oder farbig oder ein Fernsehgerät bereitstellen und ist dann sofort arbeitsbereit. Für den Anschluß eines Fernsehgerätes ist allerdings dafür noch ein TV-Adapter erforderlich. Die Bildschirmausgabe kann per Umschalter und auch per Software wahlweise auf 40 oder 80 Zeichen pro Zeile gesteuert werden. Im letzteren Falle wird offensichtlich von den 2 MHz Takt Gebrauch gemacht, für den die CPU ausgelegt ist. Programmabläufe sind im 80-Zeichen-Modus etwa doppelt so schnell wie im anderen Fall.

Der LASER ist damit sowohl für den Heim- wie auch für den Bürobetrieb ausgelegt. 40 Zeichen je Zeile wird man beim Anschluß eines Fernsehgerätes wählen. Beim Anschluß eines Monitors erhält man ein gestochen scharfes Bild, auch bei 80 Zeichen und bei den grafischen Betriebsarten, die noch besprochen werden. Die Konsole ist massiv gebaut und hat kräftige Tasten die in zwei Hauptfeldern angeordnet sind. Über dem größeren Buchstabenfeld sind acht Funktionstasten angeordnet, rechts davon ein numerisches Feld mit ENTER-Taste und darüber vier Tasten für die Cursorsteuerung. Die Funktionstasten sind beim Einschalten mit Standard-Befehlstexten belegt (mit Shift und CTRL können insgesamt 24 auch lange Befehlstexte abgerufen werden). Der Benutzer kann diese Tasten jedoch auch mit eigenen Texten belegen. Im Buchstabenfeld hat es keine deutschen Umlaute und leider auch kein 'ß'. Diese Zeichen sind offensichtlich auch nicht im Zeichengenerator gespeichert. Mit der Taste Caps Loc erreicht man, daß die Tastatur wie diejenige einer Schreibmaschine reagiert. Ungeshiftet hat man Kleinbuchstaben und geschiftet Großbuchstaben. Dem Vernehmen nach bemüht sich Sanyo als Importeur, künftig Versionen zu erhalten, die den deutschen Zeichensatz sowie eine deutsche Tastaturbelegung haben. Auch die jetzt noch eingelötete CPU soll gesockelt werden.

Der Aufbau des Rechners unterscheidet sich vom Apple mit seinen Slots. Auf der Rückseite finden wir neben den Buchsen für Video (Composite Signal und RGB) drei Herausführungen der Platine auf Kontaktzungen. Eine dient dem optionalen RS232-Interface, eine dem Drucker und die dritte schließlich normalerweise dem Floppy Disk Controller, der zwei Laufwerke bedienen kann. Für mögliche Erweiterungen ist der Systembus außerdem seitlich auf eine Buchsenleiste herausgeführt. Zur Ausstattung mit Speichern ist zu bemerken: Ein Festwertspeicher mit dem Betriebssystem (Kernel) und Microsoft-BASIC liegt im Bereich von $C000-FFFF. Ab $C000 sind ihm E/A-Adressen unterlegt. Durchgängiges RAM (einschließlich Video-RAM) liegt unterhalb dieser Adressen, also 48 KB. Der Rechner hat jedoch 64 KB RAM und kann im Ausbau 256 KB adressieren. Hierzu ist eine Segmentumschaltung vorgesehen, die jeweils Blöcke von 16 KB umfaßt und die durch Beschreiben von Adressen im C-Bereich bewirkt wird. Mehr noch: Die Segmente können für Adressenbereiche 'gemapt' werden, sie können also verschiedenen Bereichen im logischen Adreßraum von 64 KB zugeordnet werden, womit ein hohes Maß an Beweglichkeit erreicht wird. Die Dekodierung aller Adressen erfolgt im System durch zwei PALs.

Die interne Zeichendarstellung entspricht offenbar der des Apple, womit Abweichungen gegenüber der ASCII-Darstellung gegeben sind. Der Autor bemerkte das sehr deutlich bei der Inbetriebnahme der RS232-Schnittstelle. Im empfangenden Computer war eine Codewandlung mit nicht ganz durchgängiger Logik notwendig. Die Codes des Rechners sind allerdings in den Handbüchern beschrieben.

Der Computer kann ohne Floppy Disk betrieben werden. In diesem Falle speichert und lädt man Programme und Daten über den Tonbandkanal. Files erhalten dabei jedoch keinen Vorspann mit einem Namen und einer Startadresse, was die Haushaltung für Dateien erschwert. Andererseits können Aufzeichnungen an jeden Speicherplatz geladen werden. An anderer Stelle im Heft ist beschrieben, wie man mit einem anderen Computer Tonbandfiles erzeugen kann, die der LASER dann lesen kann (Informationsaustausch und Transport). - Bei aufgestecktem Floppy Controller (DM 218, ein Laufwerk kostet DM 598) wird beim Einschalten zunächst das DOS 3.3 gebootet (geladen). Dem Benutzer stehen dann die üblichen Befehle für das Disketten-Handling zur Verfügung, die im 96-seiten DOS-Handbuch dokumentiert sind.

Sofern man eine Floppy hat (ein Laufwerk speichert 160 KB) und noch einmal etwa DM 1000 für das aufsteckbare Z80-Modul und für die CP/M-Diskette mit Handbuch ausgibt, dann hat man auch auf dem LASER die für dieses Betriebssystem vorhandene Software zur Verfügung, so wie es auch bei Apple möglich ist. Das konnte nicht getestet werden. - Die Erprobung des Rechners erfolgte hier mit monochromen Monitor, von der Farbtüchtigkeit (8 Farben) konnte kein Gebrauch gemacht werden. (Man muß sich auch vor Augen halten, daß 'Farbe' auf dem Bildschirm zusätzlich kostet, der LASER hat immerhin einen Grundpreis vom knapp DM 1700.)

Was bietet nun der LASER 3000, und welche Wünsche bleiben vielleicht offen? Zunächst zum BASIC. Es hat einen ähnlichen Kern, wie andere Microsoft BASICs z.B. von Commodore. Es kann auch auf logische Kanäle schreiben und von dort lesen, wobei der Benutzer die Möglichkeit hat, eigene Treiberprogramme zum Einsatz zu bringen. Der Hauptunterschied liegt in der Unterstützung der sehr guten grafischen Fähigkeiten des Computers, auch im SOUND für den eingebauten Klangerzeuger. Die wesentlichen Befehle sind hier DRAW und XDRAW (löschen) verbunden mit AT (Koordinaten), SCALE (Größe), ROT (Drehung), CIRCLE, SQUARE (beide als gefüllte Fläche oder als Umriß). Man kann jedoch im Speicher auch eigene Umrißtafeln (bis 256) ablegen, auf die mit DRAW n AT zurückgegriffen wird, n ist dabei eine laufende Nummer. Den Beginn einer solchen Tafel legt

man zuvor an einer bestimmten Adresse ab. Das BASIC-Handbuch beschreibt, wie eine Tafel aufgebaut werden soll. Der Befehl SHLOAD dient dem Laden einer 'Shape-Table'. - Dem Setzen von Punkten oder dem Ziehen von Linien dient der Befehl HPLOT. Farben werden mit HCOLOR gesteuert. PAINT dient dem farbigen Füllen einer Fläche auf dem Bildschirm

Grafiken können in verschiedenen Auflösungsgraden erzeugt werden, bis hin zu 560x192 Pixels (Befehle der HGR-Gruppe), im Low Resolution Mode mit Text und Grafik hat man 280x160 Pixels sowie vier Zeilen Text unten im Bildschirm. Der Befehl TEXT mit drei Farbparametern steuert die Farben der Buchstaben, des Hintergundes und des Bildrandes. Daneben gibt es INVERSE und FLASHING (blinkende) Darstellung, Rückschaltung mit NORMAL. Ein Demoprogramm für den LASER führt die grafischen Fähigkeiten beeindruckend vor, zusammen auch mit SOUND (drei Kanäle, daneben Höhe, Dauer, Lautstärke) und NOISE. Wenn man sich hierzu die Programmlisten anschaut, so bemerkt man, daß mit wenigen Befehlen viel Effekt erzielt wird. Das Programmieren von Grafik und Sound, worin deutliche Stärken des LASER liegen, wird also vom BASIC gut unterstützt.

Es gibt daneben andere immer wieder nützliche BASIC-Befehle, die durchaus keine Selbstverständlichkeit sind. Wir nennen die wichtigsten: TRON, TROFF (für den Trace), HIMEM, LOWMEM (Speicherbegrenzung des BASIC), ON ERROR GOTO nebst RESUME, PRINT USING, HTAB und VTAB (Ausgabeformatierung und Tabulierung), RESTORE (DATA) mit Zeilennummer, STORE und RECALL für Abspeicherung von numerischen Arrays - Zahlen werden mit 9 signifikanten Ziffern errechnet, allerdings auch hier wieder nicht im dezimalen Modus der CPU, so daß wir hier die üblichen Rundungsfehler haben. Die Berechnung von 7 hoch 7 bringt einmal wieder keine Ganzzahl. Fehler dieser Art sind auch anderswo bekannt. Das BASIC des LASER darf gleichwohl als großzügig und als auf die Fähigkeiten dieses Gerätes angepaßt bezeichnet werden.

Mit CALL 151 gelangt man in ein schmales Monitorprogramm, mit dem man Hexbytes von Speicherbereichen anzeigen und solche Bytes ablegen kann. Auch ein SAVE und LOAD ist mit vergleichbaren Befehlen für Speicherabzüge möglich. Auch aus dieser Ebene heraus kann man die logischen Kanäle ansprechen. Ferner ist ein Block-Move und ein Vergleich von Speicherbereichen möglich. Auf der maschinensprachlichen Ebene bleiben sicher noch viele Wünsche offen. So sähe man gerne mindestens einen zeilenweise arbeitenden Assembler, wenn nicht lieber ein System von Editor und Assembler. Vom Generalimporteur ist zu vernehmen, daß er um solche Software bemüht ist.

Zur Dokumentation: Sie ist besser, als wir es vom größten Anbieter von Personal Computern in unserem Lande gewohnt sind. Noch handelt es sich um Handbücher in englischer Sprache, an deutschen Ausgaben wird offensichtlich gearbeitet. Das kleine Anwenderhandbuch (User's Manual) kann man vergessen, wenn man schon auf einem Computer gearbeitet hat. Dem Anfänger erklärt es die Inbetriebnahme und die wichtigsten Hantierungen. Gut ist das 208-seitige BASIC Reference Manual, das jeden Befehl erklärt, auch mit Beispielen. Als sehr nützlich ist das Technical Reference Manual zu bezeichnen, wenn man sich eingehender mit der Maschine befassen will. Es hat 288 Seiten, enthält Schaltpläne der Baugruppen, Anschlußbelegungen, Memory Maps, ferner Systemadressen (Pointer, Schalter usw.) und Einsprungspunkte für wichtige Systemroutinen. Diese werden mit ihren Voraussetzungen und Wirkungen dann im einzelnen beschrieben. Dahinter schließt sich ein ROM-Listing für den Kernel (Betriebsprogramm) an. Zusammen mit dem Mini-Monitor sind damit Voraussetzungen gegeben, 'in die Maschine einzudringen', wenn auch viele Bausteine nicht bezeichnet sind. Die Wirkung der PALs, deren Ausgangssignale verzeichnet wurden, ist natürlich nicht auf Anhieb zu verstehen. - Das DOS Manual wurde bereits erwähnt, es enthält auf 96 Seiten alle Diskettenbefehle mit Beispielen. Zusatzbausteine werden ebenfalls dokumentiert, so liegt hier

für die Schnittstelle RS232 eine Beschreibung auf 9 Seiten vor, die auch einen Schaltplan enthält.

Zusammenfassend: Der LASER 3000 wird als 'Apple-kompatibel' bezeichnet. Nach Auskunft von Sanyo heißt das: Er ist in der Grundausstattung reichlicher als der Apple II und seine vielen Nachbauten, daher auch der mittlere Preis. Er kann Apple-Software abfahren, wenn sie nicht zu kunstreich in die Besonderheiten des Rechners eingreift. Dieser Punkt und auch die CP/M-Fähigkeit konnten hier nicht erprobt werden. Groß- und Kleinschreibung ohne erforderliche Zusätze sowie 80 Zeichen in einer Zeile, verbunden mit Betrieb unter 2 MHz, dürfen wohl als relative Vorteile gegen das Vorbild angesehen werden. Im Vergleich fehlen die vielen freien 'Slots' für die Systemerweiterung mit den vielen zum Apple erhältlichen Modulen. In den Handbüchern wird zwar ein Super System Expander erwähnt, in den Preislisten habe ich ihn nicht gesehen. Damit ist auch unbekannt, ob er Karten aufnehmen kann, wie die Slots im Apple. - Der Autor hat den Rechner nicht geöffnet, obwohl es ihm erlaubt wurde. Es wird berichtet, daß auch der Festwertspeicher eingelötet ist, was man als ein Hindernis für 'systemverändernde Maßnahmen' ansehen muß, aber nicht als Unmöglichkeit. Immerhin muß man auch bei anderen Herstellern ZeichenGeneratoren auslöten oder die Bohrlöcher für RAM-Erweiterungen entzinnen.

Peter Porbadnigk, 2070 Ahrensburg

C-64 am IEEE 488 Bus

— Erweiterungen —

Für das in Heft 39 veröffentlichte Betriebssystem des C-64 mit der parallelen IEEE-Schnittstelle haben sich einige sinnvolle Ergänzungen ergeben. Das Einlesen des Directory ist z.B. eine lästige Angelegenheit. Wenn man dabei nicht ein im Speicher befindliches Programm zerstören will, benötigt man Hilfsprogramme. Außerdem stört das verschiebliche Laden bzw. die Unkenntnis, wo ein Programm überhaupt hingeladen wurde - Hierzu wurde jetzt unter Verzicht auf die Cassettenroutinen (für Datasette) das Betriebssystem nochmals geändert. Es sind im wesentlichen die Routinen von hex F72C bis FCB7, die entfallen können.

Hier kann jetzt eine Menge nützlicher Utilities untergebracht werden, so z.B. eine Directory-Routine, die durch die Tastenkombination SHIFT+RUN/STOP ausgelöst wird. Diese Routine liest die Directories von Drive 0 und Drive 1. Mittels der F7/F8-Taste kann die Ausgabe angehalten oder mit der STOP-Taste abgebrochen werden. Weiter ist es möglich, aus dem auf dem Bildschirm stehenden Directory zu laden, indem man die Blockanzahl einfach mit LOAD überschreibt. Das rechts im Directory stehende PRG braucht nicht gelöscht bzw. mit einem Doppelpunkt abgetrennt zu werden. Auch ist die Eingabe von ',8' nicht mehr nötig, um die Einheit anzusprechen. Alle Lade- und Speicherbefehle beziehen sich ohne Angabe einer Gerätenummer auf die Floppy. Versuchte Cassetten-Operationen führen zu einer Fehlermeldung. Außerdem wird beim Laden aus dem Directory der untere Bildschirmteil gelöscht. Weiterhin wird die Lade-Anfangs und -Endadresse angezeigt.

Wegen der Anzeige der Ladeadressen wurde auf das verschiebliche Laden verzichtet. Als Standard sind jetzt ,8,1 als Ladeparameter vorgegeben, d.h. daß ein einfacher LOAD-Befehl absolut von der Floppy lädt. Um verschieblich zu laden, muß LOAD "NAME",8 eingegeben werden..

Sollten die Routinen für Datasette doch noch einmal benötigt werden, dann ist es ratsam,, die Version 2.4 auf Disk zu speichern, um sie ggfs. im RAM laufen zu lassen. Da der frei gewordene Adreßbereich durch die hier beschriebenen Routinen lange nicht aufgebraucht wurde, ist hier also noch eine Menge Platz für andere sinnvolle Ergänzungen.

Zusätzlich möchte ich noch eine weitere Änderung im C-64 beschreiben, die hier aus Gründen der Kompatibilität zu einem 8032-SK entstand. Im 8032 sind Umlaute und 'ß' auf den Tasten eckige Klammer auf (äÄ), eckige Klammer zu (üÜ), Backslash (öÖ) und das 'ß' auf dem sog. Klammeraffen. Da Texte auf dem C-64 ein unschönes Bild ergaben, sind im Betriebssystem folgende Änderungen gemacht worden, die jetzt die Umlaute im Kleinschriftmodus auf folgende Tasten legen: ä = SHIFT+eckige Klammer auf, Ä =CBM+eckige Klammer auf, ö = SHIFT+PFUND, Ö = CBM+PFUND, ü = SHIFT+eckige Klammer zu, Ü = CBM+eckige Klammer zu, ß = Klammeraffe.

Änderungen sind erforderlich in der Tastaturtabelle, damit die bisher nur über SHIFT erreichbaren Klammern auch mit CBM= möglich sind. Und der Charactergenerator (Video-Zeichengenarator) muß natürlich geändert werden. Hierzu wird dieser ins RAM kopiert und durch die im Textfile stehenden Werte geändert. Hier wurde der ursprüngliche Character-ROM mit einem 4K EPROM umschaltbar in den C-64 eingbaut.

```
1:IEEE/2.6.SC.....PAGE 0001

LINE# LOC   CODE        LINE


0001  0000            ;IEEE-BASIC VERSION 2.6
0002  0000            ;=====================
0003  0000            ;10.01.1985
0004  0000
0006  0000
0007  0000            ;DIRECTORY UND NEUER LOAD/SAVE/VERIFY
0008  0000
0009  0000            STATUS = $90          ;PERIPHERIE STATUS
0010  0000            DIREKT = $9D          ;DIREKT=$80 PROG=0
0011  0000            FNLEN  = $B7          ;FILENAME LANGE
0012  0000            SA     = $B9          ;SEC. ADRESSE
0013  0000            DN     = $BA          ;DEVICE NUMMER
0014  0000            FNAM   = $BB          ;FILENAME
0015  0000            ;
0016  0000            BASIC  = $A474        ;BASIC WARMSTART
0017  0000            LNPRT  = $BDCD        ;GIBT 16BIT ZAHL AUS
0018  0000            ;
0019  0000            BSOUT  = $FFD2        ;PRINT
0020  0000            IECIN  = $FFA5        ;IEC INPUT
0021  0000            CLSFIL = $F642        ;CLOSE FILE
0022  0000            TALK   = $FFB4        ;SEND TALK
0023  0000
0024  0000                   *= $E1D9
0025  E1D9  A2 08            LDX #8         ;DEFAULT FUR GERHTENR.
0026  E1DB  A0 01            LDY #1         ;SEKUNDHR ADRESSE 1
0027  E1DD
0028  E1DD                   *= $E20E
0029  E20E  20 A6 F7         JSR DIRLOD     ;LADEN AUS DIRECTORY
0030  E211
0031  E211                   *= $E494
0032  E494  36               .BYT '6'       ;VERSION 2.6
0033  E495
0034  E495                   *= $E5EE       ;TEST AUF RUN
0035  E5EE  4C 2C F7         JMP DIRECT
0036  E5F1
0037  E5F1                   *= $F384       ;OPEN
0038  F384  C9 02            CMP #2         ;RS 232
0039  F386  D0 08            BNE $F390      ;ILLEGAL DEV ERROR
0040  F388
0041  F388                   *= $F4F0       ;ANZEIGE DER LADEADRESSE
0042  F4F0  20 DA F7         JSR LADR01     ;STARTADRESSE ANZEIGEN
0043  F4F3
0044  F4F3                   *= $F528       ;ANZEIGE DER LADEADRESSE
0045  F528  20 EF F7         JSR LADR02     ;ENDADRESSE ANZEIGEN
0046  F52B
```

```
0047  F52B                        *= $F533          ;LOAD
0048  F533  EA                    NOP               ;RS 232 UND
0049  F534  EA                    NOP               ;TAPE
0050  F535  EA                    NOP               ;ILLEGAL DEV ERROR
0051  F536
0052  F536                        *= $F659          ;SAVE
0053  F659  EA                    NOP               ;RS 232 UND
0054  F65A  EA                    NOP               ;TAPE
0055  F65B  EA                    NOP               ;ILLEGAL DEV ERROR
0056  F65C
0057  F65C                        * = $F72C
0058  F72C
0059  F72C  A9 0D       DIRECT   LDA #$D           ;RETURN AUSGEBEN
0060  F72E  20 D2 FF             JSR BSOUT
0061  F731  A9 24                LDA #'$           ;FILENAME = $
0062  F733  85 FB                STA $FB
0063  F735  A9 FB                LDA #$FB          ;FILENAME IN $FB
0064  F737  85 BB                STA FNAM
0065  F739  A9 00                LDA #0
0066  F73B  85 BC                STA FNAM+1
0067  F73D  A9 01                LDA #1            ;1 ZEICHEN
0068  F73F  85 B7                STA FNLEN
0069  F741  A9 08                LDA #8
0070  F743  85 BA                STA DN
0071  F745  A9 60                LDA #$60          ;SEC ADR.
0072  F747  85 B9                STA SA
0073  F749  20 D5 F3             JSR $F3D5         ;SENDNAM
0074  F74C  A5 BA                LDA DN
0075  F74E  20 B4 FF             JSR TALK
0076  F751  A5 B9                LDA SA
0077  F753  20 96 FF             JSR $FF96         ;SA NACH TALK SENDEN
0078  F756  A9 00                LDA #0            ;RESET STATUS
0079  F758  85 90                STA STATUS
0080  F75A  A0 03                LDY #3
0081  F75C  84 FB       LOOP1    STY $FB
0082  F75E  20 A5 FF             JSR IECIN
0083  F761  85 FC                STA $FC
0084  F763  A4 90                LDY STATUS
0085  F765  D0 39                BNE ENDE
0086  F767  20 A5 FF             JSR IECIN
0087  F76A  A4 90                LDY STATUS
0088  F76C  D0 32                BNE ENDE
0089  F76E  A4 FB                LDY $FB
0090  F770  88                   DEY
0091  F771  D0 E9                BNE LOOP1
0092  F773  A6 FC                LDX $FC
0093  F775  20 CD BD             JSR LNPRT         ;BLÖCKE AUSGEBEN
0094  F778  A9 20                LDA #' '
0095  F77A  20 D2 FF             JSR BSOUT
0096  F77D  20 A5 FF       LOOP3  JSR IECIN
0097  F780  A6 90                LDX STATUS
0098  F782  D0 1C                BNE ENDE
0099  F784  AA                   TAX
0100  F785  F0 06                BEQ LOOP2
0101  F787  20 D2 FF             JSR BSOUT
0102  F78A  4C 7D F7             JMP LOOP3
0103  F78D  A9 0D       LOOP2    LDA #$D           ;RETURN
0104  F78F  20 D2 FF             JSR BSOUT
0105  F792  A5 C5       WAIT     LDA $C5           ;TASTE
0106  F794  C9 3F                CMP #$3F          ;IST STOP ?
0107  F796  F0 08                BEQ ENDE
0108  F798  C9 03                CMP #3            ;F7/F8
0109  F79A  F0 F6                BEQ WAIT
0110  F79C  A0 02                LDY #2
0111  F79E  D0 BC                BNE LOOP1
0112  F7A0  20 42 F6       ENDE   JSR CLSFIL
0113  F7A3  4C 74 A4             JMP BASIC
0114  F7A6
```

```
0115  F7A6  A9 50     DIRLOD LDA #'P          ;PROGRAMM FILE
0116  F7A8  A0 00            LDY #0
0117  F7AA  D1 7A            CMP ($7A),Y      ;LAUFENDES ZEICHEN =P ?
0118  F7AC  F0 03            BEQ IGNPRG       ;IGNORIERE PRG
0119  F7AE  4C FD AE         JMP $AEFD        ;NORMAL WEITERMACHEN
0120  F7B1  98       IGNPRG TYA              ;PRG LOSCHEN
0121  F7B2  91 7A            STA ($7A),Y
0122  F7B4  C8               INY
0123  F7B5  C0 03            CPY #3
0124  F7B7  D0 F9            BNE IGNPRG+1
0125  F7B9  A5 D1            LDA $D1          ;AKTUELLE ZEILENADRESSE
0126  F7BB  85 FB            STA $FB          ;NACH $FB/$FC
0127  F7BD  A5 D2            LDA $D2
0128  F7BF  85 FC            STA $FC
0129  F7C1  A0 00            LDY #0
0130  F7C3  A9 20            LDA #' '         ;SPACE
0131  F7C5  91 FB     CLR0   STA ($FB),Y      ;BILDSCHIRM AB
0132  F7C7  E6 FB            INC $FB          ;DER BEFEHLSZEILE
0133  F7C9  D0 FA            BNE CLR0         ;LOSCHEN BIS $7FF
0134  F7CB  E6 FC            INC $FC
0135  F7CD  A6 FC            LDX $FC
0136  F7CF  E0 08            CPX #8           ;NORMAL WEITERMACHEN
0137  F7D1  D0 F2            BNE CLR0
0138  F7D3  68               PLA              ;3 RUCKSPRUNGE ENTFERNEN
0139  F7D4  68               PLA
0140  F7D5  68               PLA
0141  F7D6  68               PLA
0142  F7D7  68               PLA
0143  F7D8  68               PLA
0144  F7D9  60               RTS              ;ZURUCK ZUM AUFRUF
0145  F7DA
0146  F7DA  20 D2 F5  LADR01 JSR $F5D2        ;LOADING,VERIFYING
0147  F7DD  A5 9D            LDA DIREKT
0148  F7DF  10 0D            BPL NOADR
0149  F7E1  A9 20            LDA #' '
0150  F7E3  20 D2 FF         JSR BSOUT
0151  F7E6  20 F9 F7         JSR ADROUT
0152  F7E9  A9 2D            LDA #'-
0153  F7EB  20 D2 FF         JSR BSOUT
0154  F7EE  60       NOADR  RTS
0155  F7EF
0156  F7EF  20 FB ED  LADR02 JSR $EDFB        ;UNTALK SENDEN ($EDEF)
0157  F7F2  A5 9D            LDA DIREKT
0158  F7F4  10 F8            BPL NOADR
0159  F7F6  4C F9 F7         JMP ADROUT
0160  F7F9
0161  F7F9  A5 AE     ADROUT LDA $AE          ;LADEADRESSE LOW
0162  F7FB  48               PHA              ;ZWISCHENSPEICHERN
0163  F7FC  A5 AF            LDA $AF          ;LADEADRESSE HIGH
0164  F7FE  20 02 F8         JSR NUMA
0165  F801  68               PLA
0166  F802                  ;1BYTE IN ASCII WANDELN UND AUSGEBEN
0167  F802  48       NUMA   PHA              ;BYT IN ASCII AUSGEBEN
0168  F803  4A               LSR A
0169  F804  4A               LSR A
0170  F805  4A               LSR A
0171  F806  4A               LSR A
0172  F807  20 1A F8         JSR BYTASC       ;HALBBYTE IN ASCII
0173  F80A  AA               TAX
0174  F80B  68               PLA
0175  F80C  29 0F            AND #15
0176  F80E  20 1A F8         JSR BYTASC       ;HALBBYTE IN ASCII
0177  F811  48               PHA
0178  F812  8A               TXA
0179  F813  20 D2 FF         JSR BSOUT
0180  F816  68               PLA
0181  F817  4C D2 FF         JMP BSOUT
0182  F81A
```

```
0183  F81A                ;1 HALBBYTE IN ASCII UMWANDELN
0184  F81A  18           BYTASC CLC
0185  F81B  69 F6               ADC #246
0186  F81D  90 02               BCC L001
0187  F81F  69 06               ADC #6
0188  F821  69 3A         L001  ADC #58
0189  F823  60                 RTS
0190  F824
0191  F824
0192  F824                ;ANDERUNGEN FÜR UMLAUTE UND ß
0193  F824                ;----------------------------
0194  F824
0195  F824                      *= $EBF2
0196  EBF2  5C                  .BYT $5C          ;KL Ö
0197  EBF3                      *= $EC30
0198  EC30  DB                  .BYT $DB          ;Ä
0199  EC31                      *= $EC33
0200  EC33  DC                  .BYT $DC          ;Ö
0201  EC34                      *= $EC35
0202  EC35  DD                  .BYT $DD          ;Ü
0203  EC36
0204  EC36                      .END

ERRORS = 0000

END OF ASSEMBLY
```

SPEICHERAUSZUG FÜR UMLAUTE UND ß WENN CHR-ROM AUF ADRESSE $2000

```
.M 2800 2800
.:2800 3C 66 66 6C 66 66 6C 00    ;ß
.M 28D8 28E8
.:28D8 66 00 3C 06 3E 66 3E 00    ;Ä
.:28E0 66 00 3C 66 66 66 3C 00    ;Ö
.:28E8 66 00 00 66 66 66 3F 00    ;Ü
.M 2AD8 2AE8
.:2AD8 66 18 3C 66 7E 66 66 00    ;Ä
.:2AE0 66 3C 66 66 66 66 3C 00    ;Ö
.:2AE8 66 00 66 66 66 66 3C 00    ;Ü
.M 2C00 2C00
.:2C00 C3 99 99 93 99 99 93 FF    ;REVERSE ß
.M 2CD8 2CE8
.:2CD8 99 FF C3 F9 C1 99 C1 FF    ; -''-   Ä
.:2CE0 99 FF C3 99 99 99 C3 FF    ; -''-   Ö
.:2CE8 99 FF FF 99 99 99 C0 FF    ; -''-   Ü
.M 2ED8 2EE8
.:2ED8 99 C3 99 81 99 99 99 FF    ; -''-   Ä
.:2EE0 99 C3 99 99 99 99 C3 FF    ; -''-   Ö
.:2EE8 99 FF 99 99 99 99 C3 FF    ; -''-   Ü
```

RLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRLRL

Bücher

Walwyn, H.: Strategiespiele auf dem Commodore 64. McGraw-Hill Verlag, Hamburg 1985, 225 S., ISBN 3-89028-021-8. Die im Buch enthaltenen vollständig gelisteten Spiele (BASIC) sind sämtlich Kriegsspiele, nämlich Waterloo, Caractacus, Torpedo-Bomber (Solospiel), Richthofen, Seeschlacht und Strategix. Mit einer Ausnahme sind sie für zwei Teilnehmer gedacht und durch Bildschirmauszüge und Erklärungen zum Ablauf/Umfeld unterlegt. Das hat an manchen Stellen fast die Form eines Kriegsberichtes. - Das Buch enthält auch Vorschläge zur Anpassung an andere CBM-Computer. - Über den Spielwert kann man allein nach dem Buch nichts sagen, denn vor der Erprobung steht die Mühe des Eintippens.

Horst Brettin, 1000 Berlin 44

COPY 4

Das nachfolgende Programm kopiert Disketten-Files von einem Laufwerk zum anderen bei Commodore-Floppies der verschiedensten Typen. Es kopiert nicht nur Programme und sequentielle Files, sondern auch relative Dateien. Beim Start wird der Betreiber nach dem Ziel-Laufwerk gefragt. Auf diesem kann er gegebenenfalls eine Diskette noch NEWen. Danach wird das Inhaltsverzeichnis des Quell Laufwerkes auf den Schirm geschrieben. Nun kann zeilenweise ausgewählt werden, welche Files zu kopieren sind. Das Programm wird mit dem Hochpfeil verlassen.

```
100 rem --- c o p y  4 0 4 0   v4 ---
105 rem laeuft auch auf anderen floppies, kopiert auch rel-files
110 goto 610:(c) by h.brettin 05.05.84
200 rem read directory
210 print "Sq    rdirectory from drive "sd$"q"
220 cc$="$"+sd$+":"+i$:open 2,dn,0,cc$ :rem befehlsstring zusammensetzen
230 cc$="open 2,"+right$(str$(dn),1)+",0,"+qt$+cc$+qt$
240 gosub 910:print: rem fehlerabfrage
250 get§2,a$,a$ :rem 2 zeichen ueberspringen
260 for k=0 to 232: rem fuer directory bis 8250
270 get§2,a$,a$,a$,i$:if st goto 320: rem hole lfd. eintrag
280 i$=str$(asc(a$+nil$)+256*asc(i$+nil$))+" ": rem blockzahl dekodieren und formatieren
290 get§2,a$:i$=i$+a$:if a$?"" goto 290: rem zeile zusammensetzen
300 print i$: rem zeile ausgeben
310 d$(k)=left$(i$,30):next: rem string abspeichern
320 k=k-2:print:cc$="read directory":gosub 910
330 cc$="close 2":close 2:goto 910: rem weil schon im unterprogramm goto
340 :
400 i=1:print"Sqqqqqqqqqqqqqqqqqqqqqqqqqqreditor: '+'=copy  '-'=no copy  'ß'=exit";: rem
405 j=i-15:j=-j*(j>0)                                                    kommandozeile am schluß
410 en=j+23:if en>k then en=k+1:j=en-23:if j<0 then j=0
415 print"s";:for j=j to en:print left$("r",-(i=j))d$(j):next
420 rem poke 151,0    tastenrepeat nur fuer cbm 3001
425 get a$:if a$="" goto 425
430 if a$="q" goto 470: rem cursorkommandos
435 if a$="Q" then i=i+(i1):goto 405
440 if a$="s" goto 400
445 if a$="S" then i=k:goto 405
450 if a$="ß" then return: rem steuerkommandos
455 if a$="-" then d$(i)=left$(d$(i),29)+" ":goto 470
460 if a$ <>"+" goto 425
465 d$(i)=left$(d$(i),29)+"rc": rem kennzeichen fuer das kopieren
470 i=i-(i<k):goto 405
499 :
500 print"Sr copy:
510 for i=1 to k:if mid$(d$(i),31)<>"c" then next:return
520 i$=d$(i):for j=8 to 25:if mid$(i$,j,1)<>qt$ then next: rem suche anfuehrungszeichen
530 print left$(i$,29):i$=mid$(i$,8,j-8): rem nur namen
540 cc$="c"+dd$+":*="+sd$+":"+i$: rem befehlsstring aufbauen
550 print§15,cc$:gosub 910
560 if en<>63 then next i:return: rem file exists?
570 input"file scratchen    nPPP";a$:if a$<>"y" then next i:return
580 cc$="s"+dd$+":"+i$:print§15,cc$:gosub 910:goto 540: rem dann scratch
590 :
600 rem hauptprogramm
610 print"Sqqr c o p y   d i s k   f i l e s q"
615 print"    (c) by horst brettin 1984qq
620 print"insert diskettes in drivesq": rem dialog
630 dim a$,i$,d$(232):nil$=chr$(0):qt$=chr$(34): rem a$ vorn anlegen, 232 directory-eintraege moeglich,
640 dn=8:open15,dn,15                                                    konstanten definieren
650 input"destination drive number   PPP";i
660 dd$=right$(str$(i),1):sd$=right$(str$(i-1),1): rem kommandostring vorbereiten
670 cc$="i"+sd$:print§15,cc$: rem floppy initialisieren
680 print"qdo you wish to header drive "dd$;:input"  nPPP";a$:if a$="y"then 700
690 gosub 910:cc$="i"+dd$:print§15,cc$:goto 740
700 input"disk name   PPP";a$:if a$=" "then print"Q";:goto 700
710 input"id (for formatting)   PPP";i$:if i$=" " then i$="":goto 730
```

```
720 i$=","+left$(i$,2)
730 gosub 910:cc$="n "+dd$+":"+a$+i$:print§15,cc$
740 input"qrpatternR (use 4040 rules)  *PPP";i$
750 gosub 910:gosub 200
760 if k>0 then gosub 400:gosub 500
770 print"qcopy finished!!q":for i=0 to 1000:next
780 i$="*":sd$=dd$:gosub 200:goto 999
790 :
900 rem input from error channel
910 input§15,en,em$,et$,es$
920 if en=0 then return
930 print"r"cc$
940 if en=1 then print et$" "em$:return
950 print"R C>error §"en"r"em$;
960 if en<30 then print" on "et$"."es$;
970 print"q
980 input"continue  nPPP";a$:if a$="y"then return
999 close 15
```

Legende:
Die unterstrichenen Zeichen
in den Strings sind Steuerzeichen
q = Cursor down
Q = Cursor up
p = Cursor rechts
P = Cursor links
r = revers
S = Clear home

Peter Engels, 5308 Rheinbach

C-64 Disk-Monitor (2)

Heft 40 enthielt auf den Seiten ab 57 den Beginn des Disk-Monitors. Hier folgt nun der Abschluß der Programm-Liste.

```
08E7- 20 0C 08  1510 DSP1       JSR ASK
08EA- 20 BB 08  1520            JSR CRLF
08ED- AC 04 02  1530            LDY VON
08F0- 20 F2 0A  1540 DSP2       JSR TESTEND
08F3- 20 02 0B  1550            JSR ALTRIT
08F6- 98        1560            TYA
08F7- 20 26 09  1570            JSR WROB            ; ADRESSE
08FA- 20 BB 08  1580            JSR SPACE
08FD- A9 08     1590            LDA #COUNT          ; 8 BYTES
08FF- 20 93 08  1600            JSR DM              ;  ANZEIGEN
0902- 20 C3 0A  1610            JSR ASCDMP          ; ASCII-DUMP
0905- 4C F0 08  1620            JMP DSP2            ; IMMER
                1630 ;
0908- 4C 5E 08  1640 BEQS1      JMP START
                1650 ;
                1660 ; SPEICHER AENDERN
                1670 ;
0908- 20 48 09  1680 ALTM       JSR RDOB            ; ADRESSE LESEN
090E- 90 F8     1690            BCC BEQS1
0910- AB        1700            TAY
0911- A9 08     1710            LDA #COUNT          ; 8 BYTES
0913- 85 97     1720            STA *TMPC
0915- 20 7D 09  1730            JSR RDOC
0918- 20 7D 09  1740 A5         JSR RDOC
091B- 20 A9 08  1750            JSR BYT
091E- D0 F8     1760            BNE A5
0920- 20 C3 0A  1770            JSR ASCDMP
0923- 4C 5E 08  1780            JMP START
                1790 ;
                1800 ; BYTE ALS HEX-ZAHL AUSGEBEN
                1810 ;
0926- 48        1820 WROB       PHA
0927- 4A        1830            LSR A
0928- 4A        1840            LSR A
0929- 4A        1850            LSR A
092A- 4A        1860            LSR A
092B- 20 3E 09  1870            JSR ASCII           ; NACH ASCII INVERTIEREN
092E- AA        1880            TAX
092F- 68        1890            PLA
0930- 29 0F     1900            AND #%00001111
0932- 20 3E 09  1910            JSR ASCII
                1920 ;
```

```
                    1930 ; ZEICHEN IN X & A SCHREIBEN
                    1940 ;
0935- 48            1950 WRTWD      PHA
0936- 8A            1960            TXA
0937- 20 D2 FF      1970            JSR BASOUT
093A- 68            1980            PLA
093B- 4C D2 FF      1990            JMP BASOUT
                    2000 ;
093E- 18            2010 ASCII      CLC
093F- 69 F6         2020            ADC #$F6
0941- 90 02         2030            BCC ASC1
0943- 69 06         2040            ADC #6
0945- 69 3A         2050 ASC1       ADC #$3A
0947- 60            2060            RTS
                    2070 ;
                    2080 ; HEXBYTE LESEN UND IN ACCU BRINGEN
                    2090 ;
0948- A9 00         2100 RDOB       LDA #0
094A- 8D 03 02      2110            STA BAD
094D- 20 7D 09      2120            JSR RDOC
0950- C9 20         2130 RDOB1      CMP #' '    ; BLANK ?
0952- D0 09         2140            BNE RDOB2
0954- 20 7D 09      2150            JSR RDOC            ; NAECHSTES ZEICHEN
0957- C9 20         2160            CMP #' '    ; BLANK ?
0959- D0 0F         2170            BNE RDOB3
095B- 18            2180            CLC
095C- 60            2190            RTS
                    2200 ;
095D- 20 72 09      2210 RDOB2      JSR HEXIT           ; NACH HEX
0960- 0A            2220            ASL A
0961- 0A            2230            ASL A
0962- 0A            2240            ASL A
0963- 0A            2250            ASL A
0964- 8D 03 02      2260            STA BAD
0967- 20 7D 09      2270            JSR RDOC
096A- 20 72 09      2280 RDOB3      JSR HEXIT
096D- 0D 03 02      2290            ORA BAD
0970- 38            2300            SEC
0971- 60            2310            RTS
                    2320 ;
0972- C9 3A         2330 HEXIT      CMP #$3A
0974- 08            2340            PHP
0975- 29 0F         2350            AND #%00001111
0977- 28            2360            PLP
0978- 90 02         2370            BCC HEX09           ;0-9
097A- 69 08         2380            ADC #8              ; 9 ADDIEREN
097C- 60            2390 HEX09      RTS
                    2400 ;
097D- 20 CF FF      2410 RDOC       JSR INPUT           ; ZEICHEN LESEN
0980- C9 0D         2420 TSTCR      CMP #CR             ; CRLF ?
0982- D0 F8         2430            BNE HEX09           ; NEIN -> RETURN
0984- 68            2440            PLA
0985- 68            2450            PLA
0986- 4C 5E 08      2460            JMP START
                    2470 ;
                    2480 ; DOS - SUPPORT
                    2490 ;
0989- 20 CF FF      2500 DISK       JSR INPUT
098C- C9 0D         2510            CMP #CR
098E- D0 27         2520            BNE DSKCMD          ; DISK COMMANDO
0990- A9 00         2530            LDA #0
0992- 85 90         2540            STA *STATUS
0994- 20 BB 08      2550            JSR CRLF
0997- A9 08         2560            LDA #8
0999- 85 BA         2570            STA *FA             ; PRIM-ADR
099B- 20 B4 FF      2580            JSR TALK
099E- A9 6F         2590            LDA #15+$60         ; SEC-ADR 15
09A0- 85 B9         2600            STA *SA
09A2- 20 96 FF      2610            JSR SECTALK         ; SENDEN
09A5- 20 A5 FF      2620 ERRIN      JSR IECIN
09A8- 24 90         2630            BIT *STATUS
```

```
09AA- 70 05     2640            BVS ENDDSK
09AC- 20 D2 FF  2650            JSR BASOUT
09AF- D0 F4     2660            BNE ERRIN
09B1- 20 AB FF  2670 ENDDSK     JSR UNTALK
09B4- 4C 5E 08  2680            JMP START
                2690 ;
09B7- C9 24     2700 DSKCMD     CMP #'$'
09B9- F0 1D     2710            BEQ ERRL            ; CATALOG
09BB- 48        2720            PHA
09BC- A9 08     2730            LDA #8
09BE- 85 BA     2740            STA #FA             ; PRIM-ADR
09C0- 20 B1 FF  2750            JSR LISTEN
09C3- A9 6F     2760            LDA #15+$60
09C5- 85 B9     2770            STA *SA             ; SEC-ADR
09C7- 20 93 FF  2780            JSR SECLIST
09CA- 68        2790            PLA
09CB- 20 A8 FF  2800 CMDOUT     JSR IECOUT
09CE- 20 CF FF  2810            JSR INPUT
09D1- C9 0D     2820            CMP #CR
09D3- D0 F6     2830            BNE CMDOUT
09D5- 20 AE FF  2840            JSR UNLIST
09D8- 4C 5E 08  2850 ERRL       JMP START
                2860 ;
09DB- 20 7D 09  2870 DIRENT     JSR RDOC
09DE- 20 48 09  2880 HEXIN      JSR RDDB            ; TRACK LESEN
09E1- 90 F5     2890            BCC ERRL
09E3- 8D B1 0C  2900            STA TRACK
09E6- 20 7D 09  2910            JSR RDOC
09E9- 20 48 09  2920            JSR RDDB            ; SECTOR LESEN
09EC- 90 EA     2930            BCC ERRL
09EE- 8D B4 0C  2940            STA SECTOR
09F1- 20 BB 0A  2950            JSR OPNDIR
09F4- AD 0C 02  2960            LDA SAVX
09F7- C9 01     2970            CMP #1
09F9- F0 2B     2980            BEQ DIRWRITE
09FB- A9 31     2990 DIRIN1     LDA #'1'            ; "1"
09FD- 20 41 0A  3000            JSR SENDCMD         ; BLOCK READ BEFEHL AUSG.
0A00- A2 05     3010            LDX #15
0A02- 20 C6 FF  3020            JSR CHKIN
0A05- A2 00     3030            LDX #0
0A07- 20 CF FF  3040 DIRIN      JSR INPUT
0A0A- 9D BF 0C  3050            STA BUFFER,X
0A0D- E8        3060            INX
0A0E- D0 F7     3070            BNE DIRIN
0A10- 20 CC FF  3080            JSR CLRCH
0A13- 20 AC 0A  3090 ENDDIR     JSR CLSDIR
0A16- A0 00     3100            LDY #0
0A18- 8C 04 02  3110            STY VON
0A1B- A0 80     3120            LDY #$80
0A1D- 8C 05 02  3130            STY BIS
0A20- 4C E7 08  3140            JMP DSP1
                3150 ;
0A23- 20 73 0A  3160 DIRWRITE   JSR BUFPNT          ; BUFFERPOINTER SETZEN
0A26- A2 0D     3170            LDX #13
0A28- 20 C9 FF  3180            JSR CHOUT
0A2B- A2 00     3190            LDX #0
0A2D- BD BF 0C  3200 DIROUT     LDA BUFFER,X
0A30- 20 D2 FF  3210            JSR BASOUT
0A33- E8        3220            INX
0A34- D0 F7     3230            BNE DIROUT
0A36- 20 CC FF  3240            JSR CLRCH
0A39- A9 32     3250            LDA #'2'            ; "2"
0A3B- 20 41 0A  3260            JSR SENDCMD         ; BLOCK-WRITE BEFEHL SEND
0A3E- 4C 13 0A  3270            JMP ENDDIR
                3280 ;
0A41- 8D 7A 0C  3290 SENDCMD    STA CMDSTR+1
0A44- A2 0F     3300            LDX #15
0A46- AD B1 0C  3310            LDA TRACK
0A49- 20 B6 0A  3320            JSR NUMBASC
0A4C- 8E B1 0C  3330            STX TRACK
0A4F- 8D B2 0C  3340            STA TRACK+1
0A52- AD B4 0C  3350            LDA SECTOR
```

```
0A55- 20 B6 0A  3360            JSR NUMBASC
0A58- 8E B4 0C  3370            STX SECTOR
0A5B- 8D B5 0C  3380            STA SECTOR+1
0A5E- A2 0F     3390            LDX #15
0A60- 20 C9 FF  3400            JSR CKOUT
0A63- A2 00     3410            LDX #0
0A65- BD 79 0C  3420 COMDOUT    LDA CMDSTR,X
0A68- 20 D2 FF  3430            JSR BASOUT
0A6B- E8        3440            INX
0A6C- E0 0D     3450            CPX #$0D        ; ENDE ERREICHT
0A6E- D0 F5     3460            BNE COMDOUT
0A70- 4C CC FF  3470            JMP CLRCH
                3480 ;
0A73- A2 0F     3490 BUFPNT     LDX #15
0A75- 20 C9 FF  3500            JSR CKOUT
0A78- A2 00     3510            LDX #0
0A7A- BD B6 0C  3520 PNTOUT     LDA BUFTXT,X
0A7D- 20 D2 FF  3530            JSR BASOUT
0A80- E8        3540            INX
0A81- E0 08     3550            CPX #08
0A83- D0 F5     3560            BNE PNTOUT
0A85- 4C CC FF  3570            JMP CLRCH
                3580 ;
0A88- A9 0F     3590 OPNDIR     LDA #15
0A8A- A8        3600            TAY
0A8B- A2 08     3610            LDX #8
0A8D- 20 BA FF  3620            JSR SETPAR
0A90- A9 00     3630            LDA #0
0A92- 20 BD FF  3640            JSR SETNAM
0A95- 20 C0 FF  3650            JSR OPEN
0A98- A9 0D     3660            LDA #13
0A9A- A8        3670            TAY
0A9B- A2 08     3680            LDX #8
0A9D- 20 BA FF  3690            JSR SETPAR
0AA0- A9 01     3700            LDA #1
0AA2- A2 3D     3710            LDX #L,DADR
0AA4- A0 0C     3720            LDY #H,DADR
0AA6- 20 BD FF  3730            JSR SETNAM
0AA9- 4C C0 FF  3740            JMP OPEN
                3750 ;
0AAC- A9 0D     3760 CLSDIR     LDA #13
0AAE- 20 C3 FF  3770            JSR CLOSE
0AB1- A9 0F     3780            LDA #15
0AB3- 4C C3 FF  3790            JMP CLOSE
                3800 ;
0AB6- A2 30     3810 NUMBASC    LDX #$30        ; "0"
0AB8- 38        3820            SEC
0AB9- E9 0A     3830 NUMB1      SBC #10
0ABB- 90 03     3840            BCC NUMB2
0ABD- E8        3850            INX
0ABE- B0 F9     3860            BCS NUMB1
0AC0- 69 3A     3870 NUMB2      ADC #$3A
0AC2- 60        3880            RTS
                3890 ;
0AC3- 98        3900 ASCDMP     TYA
0AC4- 38        3910            SEC
0AC5- E9 08     3920            SBC #COUNT
0AC7- A8        3930            TAY
0AC8- 20 B8 08  3940            JSR SPACE
0ACB- A9 12     3950            LDA #18         ; RVS ON
0ACD- 20 D2 FF  3960            JSR BASOUT
0AD0- A2 08     3970            LDX #COUNT
0AD2- B9 BF 0C  3980 AC2        LDA BUFFER,Y
0AD5- 29 7F     3990            AND #$7F
0AD7- C9 20     4000            CMP #' '
0AD9- B0 04     4010            BCS AC3
0ADB- A9 2E     4020            LDA #'.'
0ADD- D0 03     4030            BNE AC4
0ADF- B9 BF 0C  4040 AC3        LDA BUFFER,Y
0AE2- 20 D2 FF  4050 AC4        JSR BASOUT
0AE5- A9 00     4060            LDA #0
0AE7- 85 D4     4070            STA *QUOTFLG
```

```
0AE9- C8          4080              INY
0AEA- CA          4090              DEX
0AEB- D0 E5       4100              BNE AC2
0AED- A9 92       4110              LDA #146          ; RVS OFF
0AEF- 4C D2 FF    4120              JMP BASOUT
                  4130 ;
0AF2- AD 02 02    4140 TESTEND      LDA WRAP
0AF5- D0 06       4150              BNE ENDEND
0AF7- CC 05 02    4160              CPY BIS
0AFA- B0 01       4170              BCS ENDEND
0AFC- 60          4180              RTS
                  4190 ;
0AFD- 68          4200 ENDEND       PLA
0AFE- 68          4210              PLA
0AFF- 4C 5E 08    4220              JMP START
                  4230 ;
0B02- 20 BB 08    4240 ALTRIT       JSR CRLF
0B05- A9 3A       4250              LDA #':'
0B07- A2 3E       4260              LDX #PROMPT
0B09- 4C 35 09    4270              JMP WRTWO
                  4280 ;
0B0C- A2 00       4290 ASK          LDX #0
0B0E- BD 53 0C    4300 ASK1         LDA TRK,X
0B11- F0 06       4310              BEQ SHOW
0B13- 20 D2 FF    4320              JSR BASOUT
0B16- E8          4330              INX
0B17- D0 F5       4340              BNE ASK1
0B19- AE 81 0C    4350 SHOW         LDX TRACK
0B1C- AD 82 0C    4360              LDA TRACK+1
0B1F- 20 35 09    4370              JSR WRTWO
0B22- 20 B8 08    4380              JSR SPACE
0B25- AE 84 0C    4390              LDX SECTOR
0B28- AD 85 0C    4400              LDA SECTOR+1
0B2B- 20 35 09    4410              JSR WRTWO
0B2E- 60          4420              RTS
                  4430 ;
0B2F- A2 00       4440 ENDFILE      LDX #0
0B31- BD 6C 0C    4450 LOOP         LDA ENDFI,X
0B34- D0 03       4460              BNE LOOP1
0B36- 4C 5E 08    4470              JMP START
0B39- 20 D2 FF    4480 LOOP1        JSR BASOUT
0B3C- E8          4490              INX
0B3D- D0 F2       4500              BNE LOOP
                  4510 ;
0B3F- AD 8F 0C    4520 NEXT         LDA BUFFER
0B42- F0 EB       4530              BEQ ENDFILE
0B44- 8D 81 0C    4540              STA TRACK
0B47- AD 90 0C    4550              LDA BUFFER+1
0B4A- 8D 84 0C    4560              STA SECTOR
0B4D- 20 88 0A    4570              JSR OPNDIR        ; OPEN 15,"#":OPEN 13
0B50- 4C FB 09    4580              JMP DIRIN1
                  4590 ;
0B53- A2 02       4600 INFO         LDX #2
0B55- 20 0E 0B    4610              JSR ASK1          ; TRACK & SECTOR AUSG
0B58- 4C 5E 08    4620              JMP START
                  4630 ;
                  4640 ;
0B5B- A0 00       4650 LISTLO       LDY #0
0B5D- 8C 04 02    4660              STY VON
0B60- A0 7F       4670              LDY #$7F
0B62- 8C 05 02    4680              STY BIS
0B65- 4C E7 08    4690              JMP DSP1
                  4700 ;
0B68- A0 80       4710 LISTHI       LDY #$80
0B6A- 8C 04 02    4720              STY VON
0B6D- A0 FF       4730              LDY #$FF
0B6F- 8C 05 02    4740              STY BIS
0B72- 4C E7 08    4750              JMP DSP1
                  4760 ;
0B75- A9 93       4770 CATALOG      LDA #$93          ; CLEAR SCREEN
0B77- 20 D2 FF    4780              JSR BASOUT
0B7A- A9 3E       4790              LDA #L,CADR
```

```
0B7C- 85 BB     4800            STA *FNADR
0B7E- A9 0C     4810            LDA #H,CADR
0B80- 85 BC     4820            STA *FNADR+1
0B82- A9 01     4830            LDA #1
0B84- 85 B7     4840            STA *FNLEN
0B86- A9 08     4850            LDA #8
0B88- 85 BA     4860            STA *FA
0B8A- A9 60     4870            LDA #$60
0B8C- 85 B9     4880            STA *SA
0B8E- 20 D5 F3  4890            JSR SENDNAM
0B91- A5 BA     4900            LDA *FA
0B93- 20 B4 FF  4910            JSR TALK
0B96- A5 B9     4920            LDA *SA
0B98- 20 96 FF  4930            JSR SECTALK
                4940 ;
0B9B- A9 00     4950            LDA #0                  ; STATUS LOESCHEN
0B9D- 85 90     4960            STA *STATUS
0B9F- A0 03     4970            LDY #3
0BA1- 84 97     4980 L1         STY *TMPC
0BA3- 20 A5 FF  4990            JSR IECIN
0BA6- 85 98     5000            STA *TMPC+1
0BA8- A4 90     5010            LDY *STATUS
0BAA- D0 37     5020            BNE L4
0BAC- 20 A5 FF  5030            JSR IECIN
0BAF- A4 90     5040            LDY *STATUS
0BB1- D0 30     5050            BNE L4
0BB3- A4 97     5060            LDY *TMPC
0BB5- 88        5070            DEY
0BB6- D0 E9     5080            BNE L1
0BB8- A6 98     5090            LDX *TMPC+1
0BBA- 20 CD BD  5100            JSR LNPRT               ; 16-BIT AUSGEBEN
0BBD- 20 B8 08  5110            JSR SPACE
0BC0- 20 A5 FF  5120 L3         JSR IECIN
0BC3- A6 90     5130            LDX *STATUS
0BC5- D0 1C     5140            BNE L4
0BC7- AA        5150            TAX
0BC8- F0 12     5160            BEQ L2
0BCA- 20 D2 FF  5170            JSR BASOUT
0BCD- 20 E4 FF  5180            JSR GET                 ; TEST OB SPACE
0BD0- C9 20     5190            CMP #' '
0BD2- D0 05     5200            BNE NOSPAC
0BD4- 20 E4 FF  5210 GETSP      JSR GET                 ; WARTET AUF 2. TASTE
0BD7- F0 FB     5220            BEQ GETSP
0BD9- 4C C0 0B  5230 NOSPAC     JMP L3
0BDC- 20 B6 08  5240 L2         JSR CRLF
0BDF- A0 02     5250            LDY #2
0BE1- D0 BE     5260            BNE L1
0BE3- A9 08     5270 L4         LDA #8
0BE5- 20 42 F6  5280            JSR CLSFIL
0BE8- 4C 5E 08  5290            JMP START
                5300 ;
0BEB- 20 E7 FF  5310 DEVICE     JSR CLALL               ; AUSGABE AUF SCHIRM
0BEE- 20 CF FF  5320            JSR INPUT
0BF1- C9 0D     5330            CMP #CR
0BF3- F0 06     5340            BEQ DEFAULT
0BF5- 20 48 09  5350            JSR RDDB
0BF8- 90 01     5360            BCC DEFAULT
0BFA- 2C        5370            .BY $2C
0BFB- A9 03     5380 DEFAULT    LDA #$03
0BFD- 8D 06 02  5390            STA PRINTDEV
0C00- A9 0E     5400 SETOUT     LDA #14
0C02- A8        5410            TAY
0C03- AE 06 02  5420            LDX PRINTDEV
0C06- 20 BA FF  5430            JSR SETPAR
0C09- A9 00     5440            LDA #0
0C0B- 20 BD FF  5450            JSR SETNAM
0C0E- 20 C0 FF  5460            JSR OPEN
0C11- A2 0E     5470            LDX #14
0C13- 20 C9 FF  5480            JSR CKOUT
0C16- 4C 5E 08  5490            JMP START
                5500 ;
                5510 ; BEFEHLS & ADRESS-TABELLE
                5520 ;
```

```
0C19- 3A        5530 CMDS       .BY ':'                 ; ALTER MEMORY
0C1A- 57        5540            .BY 'W'                 ; WRITE BLOCK
0C1B- 52        5550            .BY 'R'                 ; READ BLOCK
0C1C- 4D        5560            .BY 'M'                 ; MEMORY LIST
0C1D- 40        5570            .BY '@'                 ; DISK-COMMAND
0C1E- 58        5580            .BY 'X'                 ; EXIT
0C1F- 49        5590            .BY 'I'                 ; INFO TRACK SECTOR
0C20- 4E        5600            .BY 'N'                 ; NEXT TRACK SECTOR
0C21- 44        5610            .BY 'D'                 ; SET DEVICE
0C22- 31        5620            .BY '1'                 ; LIST $00-$7F

0C23- 32        5630            .BY '2'                 ; LIST $80-$FF
0C24- 24        5640            .BY '$'                 ; CATALOG
                5650 ;
0C25- 0A 09     5660 ADRH       .SI ALTM-1
0C27- DA 09     5670            .SI DIREKT-1
0C29- DA 09     5680            .SI DIREKT-1
0C2B- BF 08     5690            .SI DSPLYM-1
0C2D- 88 09     5700            .SI DISK-1
0C2F- 7A E3     5710            .SI READY-1
0C31- 52 0B     5720            .SI INFO-1
0C33- 3E 0B     5730            .SI NEXT-1
0C35- EA 0B     5740            .SI DEVICE-1
0C37- 5A 0B     5750            .SI LISTLO-1
0C39- 67 0B     5760            .SI LISTHI-1
0C3B- 74 0B     5770            .SI CATALOG-1
                5780 ;
0C3D- 23        5790 DADR       .BY '#'
0C3E- 24        5800 CADR       .BY '$'
0C3F- 0D 44 49  5810 MESSAGE    .BY CR 'DISK-MONITOR V 2.4' 0
0C42- 53 4B 2D
0C45- 4D 4F 4E
0C48- 49 54 4F
0C4B- 52 20 56
0C4E- 20 32 2E
0C51- 34 00
0C53- 93 0D 0D  5820 TRK        .BY $93 $0D $0D 'TRACK/SECTOR (DEZ) : ' $00
0C56- 54 52 41
0C59- 43 4B 2F
0C5C- 53 45 43
0C5F- 54 4F 52
0C62- 20 28 44
0C65- 45 5A 29
0C68- 20 3A 20
0C6B- 00
0C6C- 20 46 49  5930 ENDFI      .BY ' FILE-ENDE !' $00
0C6F- 4C 45 2D
0C72- 45 4E 44
0C75- 45 20 21
0C78- 00
0C79- 55 31 3A  5840 CMDSTR     .BY 'U1:13 0' $20
0C7C- 31 33 20
0C7F- 30 20
0C81- 00 00 20  5850 TRACK      .BY 0 0 $20
0C84- 00 00     5860 SECTOR     .BY 0 0
0C86- 42 2D 50  5870 BUFTXT     .BY 'B-P 13 00
0C89- 20 31 33
0C8C- 20 30 30
                5880 ;
0C8F- 00        5890 BUFFER     BRK
                5900 ;
                5910            .EN
END OF ASSEMBLY!
```

Kleinanzeigen

16 KByte RAM Europakarte für AIM 65 (Neudecker), dyn. RAMs 4116, Refresh auf der Karte, nur +5V erforderl., 44-pol. Steckerleiste, DM 100,-. M. Albrecht, Am Bärenbach 18, 4350 Recklinghausen, Tel.: 02361-71 251.

32 KB dyn. RAM-Karte (Neudecker), +5V, 64-pol. VG Steckerleiste, f. AIM geeignet, ein defektes 4116, DM 130,-. R. Löhr, Tel.: 04102-55 816.

Roland Löhr

Dump to Apple

Das nachfolgende Programm dient dem Datentransfer auf einen Tonbandträger, den der APPLE lesen kann. Es dient also dem Rechnerverbund über ein Medium. Es wurde zwar für den AIM 65 geschrieben, damit dem in diesem Heft beprochenen LASER 3000 Daten übertragen werden konnten. Das Prinzip kann jedoch auch bei anderen Rechnern leicht nachgebildet werden. Beim AIM liegt das Interface für die Ausgabe auf Tonband hinter dem Portpin PB7, daher hier die Initialisierung und das Umschalten des Ausgangssignals. - Die Abfragen des FROM und TO dienen zu Bestimmung des Speicherbereiches, der übertragen werden soll.

Beim APPLE liegen die Dinge offensichtlich anders als bei den meisten 6502-Systemen. Bei Ausgabe auf Tonband wird offensichtlich keine Startadresse abgesetzt, auch kein Name des Files. Dafür kann man das File nachher an beliebige Stelle wieder einladen. Und dann gibt es da noch grausige Dinge, die Eva, als sie den Trick mit dem Apfel ersann und damit den Sündenfall der Menschen auslöste, noch heute beunruhigen würde: Zur Erzeugung verschieden langer Impulse für die Synchronisation des Tonbandes und der Daten werden keine Timer eingesetzt, sondern Verzögerungsschleifen in Software. Das ist recht undurchsichtig. Der Autor machte keinen Versuch einer Verbesserung, sondern schrieb die wesentlichen Teile des Programms so ab, wie er sie im Zielcomputer vorfand, wenn dieser auf das Tonband schreibt. Im Ergebnis haben wir ohne Kommentare ein Programm, das seinen Dienst für die Übertragung von Bereichen auf einen LASER 3000 und auf einen APPLE tut.

```
000000                ;WRITE LASER 3000 CASSETTE
000000                ;BY ROLAND LOEHR 3.1.85
000000                ;SYMBOLE UND VARIABLE
000000                FROM    =$E7A3
000000                TO      =$E7A7
000000                START   =$E182
000000                ADDR    =$A41C
000000                ERROR   =$E394
000000                VIA     =$A800
000000                PORTB   =VIA
000000                T1CL    =VIA+4
000000                T1CH    =VIA+5
000000                T1LL    =VIA+6
000000
000000                REG1L   =0
000000                REG1H   =1
000000                REG2L   =ADDR
000000                REG2H   =ADDR+1
000000                CHKSUM  =3
000000                        *=$8200
008200  20 A3 E7              JSR FROM
008203  AD 1C A4              LDA ADDR
008206  85 00                 STA REG1L
008208  AD 1D A4              LDA ADDR+1
00820B  85 01                 STA REG1H
00820D  20 A7 E7              JSR TO
*****[illegible]***************************************************
INIT [illegible]SSETTEN-PORTS
00821[illegible] [illegible] A8              LDA $A802        ;DDRB
008213  09 80                 ORA #$80
008215  8D 02 A8              STA $A802
008218  AD 00 A8              LDA PORTB
00821[illegible] [illegible]              ORA #$80
0082[illegible] [illegible]              STA PORTB        ;PORTB, PB7 HI
```

```
008220  A9 40        0033 MTSAVE LDA #64
008222  20 5B 82     0034        JSR LEADER
008225  2C C6 82     0035        BIT TWOMHZ
008228  10 03        0036        BPL *+5
00822A  A0 54        0037        LDY #84
00822C  2C           0038        .BYT $2C
00822D  A0 25        0039        LDY #37
00822F  A2 00        0040 TSAVE1 LDX #0
008231  41 00        0041        EOR (REG1L,X)
008233  85 03        0042        STA CHKSUM
008235  A1 00        0043        LDA (REG1L,X)
008237  20 A8 82     0044        JSR WRBYTE
00823A  20 B1 82     0045        JSR INCRE2
00823D  2C C6 82     0046        BIT TWOMHZ
008240  10 03        0047        BPL *+5
008242  A0 3E        0048        LDY #62
008244  2C           0049        .BYT $2C
008245  A0 1B        0050        LDY #27
008247  A5 03        0051        LDA CHKSUM
008249  90 E4        0052        BCC TSAVE1
00824B  2C C6 82     0053        BIT TWOMHZ
00824E  10 03        0054        BPL *+5
008250  A0 44        0055        LDY #68
008252  2C           0056        .BYT $2C
008253  A0 20        0057        LDY #32
008255  20 A8 82     0058        JSR WRBYTE
008258  4C 82 E1     0059        JMP START
00825B               0060
00825B  2C C6 82     0061 LEADER BIT TWOMHZ
00825E  10 03        0062        BPL *+5
008260  A0 9B        0063        LDY #155
008262  2C           0064        .BYT $2C
008263  A0 49        0065        LDY #73
008265  20 7D 82     0066        JSR ZDELAY
008268  D0 F1        0067        BNE LEADER
00826A  69 FE        0068        ADC #$FE
00826C  B0 ED        0069        BCS LEADER
00826E  2C C6 82     0070        BIT TWOMHZ
008271  10 03        0071        BPL *+5
008273  A0 44        0072        LDY #68
008275  2C           0073        .BYT $2C
008276  A0 1F        0074        LDY #31
008278               0075 WRTBIT
008278  20 7D 82     0076        JSR ZDELAY
00827B  C8           0077        INY
00827C  C8           0078        INY
00827D  88           0079 ZDELAY DEY
00827E  D0 FD        0080        BNE ZDELAY
008280  90 10        0081        BCC WTAPE
008282  2C C6 82     0082        BIT TWOMHZ
008285  10 03        0083        BPL *+5
008287  A0 65        0084        LDY #101
008289  2C           0085        .BYT $2C
00828A  A0 2F        0086        LDY #47
00828C  88           0087 ODELAY DEY
00828D  D0 FD        0088        BNE ODELAY
00828F  4C 92 82     0089        JMP WTAPE
```

```
*****************************************************************
TOGGLE OUTPUT AFTER SAVING ACC.
008292  48           0091 WTAPE  PHA
008293  AD 00 A8     0092        LDA PORTB          ;PORTB
008296  49 80        0093        EOR #$80
008298  8D 00 A8     0094        STA PORTB
00829B  68           0095        PLA
00829C  2C C6 82     0096        BIT TWOMHZ
00829F  10 03        0097        BPL *+5
0082A1  A0 5A        0098        LDY #90
0082A3  2C           0099        .BYT $2C
0082A4  A0 2A        0100        LDY #42
0082A6  CA           0101        DEX
0082A7  60           0102        RTS
0082A8  A2 10        0103 WRBYTE LDX #16
0082AA  0A           0104 WBYTE1 ASL A
0082AB  20 78 82     0105        JSR WRTBIT
0082AE  D0 FA        0106        BNE WBYTE1
0082B0  60           0107        RTS
0082B1  A5 01        0108 INCRE2 LDA REG1H
0082B3  CD 1D A4     0109        CMP REG2H
0082B6  90 07        0110        BCC INCRE3
0082B8  D0 05        0111        BNE INCRE3
0082BA  A5 00        0112        LDA REG1L
0082BC  CD 1C A4     0113        CMP REG2L
0082BF  E6 00        0114 INCRE3 INC REG1L
0082C1  D0 02        0115        BNE *+4
0082C3  E6 01        0116        INC REG1H
0082C5  60           0117        RTS
0082C6  20           0118 TWOMHZ .BYT $20
0082C7               0118        TWOMHZ .BYT $20
 ERRORS=0000
```

Peter Porbadnigk, 2070 Ahrensburg

Verbesserte Disk-Befehle

im BASIC 4 der CBM 8032/4032

Ähnliche Erweiterungen, wie sie in diesem Heft für den 64er beschrieben wurden, lassen sich auch beim CBM 8032 oder 4032 verwirklichen. Nach Änderungen gemäß nachstehendem Listing können alle Diskettenbefehle direkt in das angezeigte Directory eingegeben werden. Es braucht hier nur die Blockzahl mit dem Befehl in abgekürzter Form überschrieben zu werden. Somit ist vor dem PRG-Namen genug Platz für die evtl. Angabe des Laufwerkes, z.B. dL d1,"name" oder sC d1,"name". In einigen Sonderfällen wie RENAME muß mit der Insert-Taste 1 Platz zusätzlich geschaffen werden.

Beim Laden aus dem Directory wird zusätzlich der untere Schirmteil gelöscht, und es werden die Ladeadressen ausgegeben. Eine weitere Modifikation betrifft den COPY-Befehl. Soll ein File unter gleichem Namen von einem Laufwerk zum anderen kopiert werden, so ist es unsinnig, den Namen zweimal eintippen zu müssen. Jetzt heißt es hier COPY "name"d1tod0. Außerdem wurde für LOAD, SAVE und VERIFY die Gerätenummer auf 8 geändert. Damit lädt LOAD"name" von Diskette und nicht mehr von Cassette. Auch der Befehl CATALOG wurde erweitert, so daß jetzt Namen vorgegeben werden können, z.B. CATALOGd1,"p*. Es wird nach mit 'p' beginnenden Einträgen gesucht. Weiterhin: Die Meldung OUT OF MEMORY ERROR wird unterdrückt, wenn man in den RAM-Bereich $9000-AFFF geladen hat und dann ein BASIC-Programm mit RUN startet. - Für das Monitorprogramm ist eine neue Kopfzeile vorgesehen, in der die Statusflags in binärer Form angezeigt werden.

```
0003  0000             WRITE  = $FFD2
0004  0000
0005  0000             ;MONITORAUSGABE AUF 16 BYTES/LINE
0006  0000             ;================================
0007  0000
0008  0000                    *= $D5EE
0009  D5EE
0010  D5EE  A9 10              LDA #16
0011  D5F0
0012  D5F0                    *= $D625
0013  D625
0014  D625  A9 10              LDA #16
0015  D627
0016  D627             ;MONITOR-BREAK MIT AUSGABE DER
0017  D627             ;   STATUS-FLAGS NV-BDIZC
0018  D627             ;================================
0019  D627
0020  D627                    *= 54665
0021  D589
0022  D589  BD DB DE           LDA TEXT,X       ;NEUER TEXT
0023  D58C
0024  D58C                    *= 54672
0025  D590
0026  D590  E0 28              CPX #40          ;NEUE LANGE
0027  D592
0028  D592                    *= 54711
0029  D5B7
0030  D5B7  20 04 DF           JSR STATUS       ;STATUS AUSGABE
0031  D5BA
0032  D5BA                    *= $DEDB
0033  DEDB
0034  DEDB  0D          TEXT   .BYT $D,'     PC  IRQ
0034  DEDC  20 20
0035  DEEA  53 52             .BYT 'SR AC XR YR SP '
0036  DEF9  4E 56             .BYT 'NV-BDIZC    '
0037  DF04
0038  DF04  20 F7 D4    STATUS JSR 54519        ;REGISTER AUSGEBEN
0039  DF07  20 31 D5           JSR 54577        ;SPACE AUSGEBEN
0040  DF0A  AD 02 02           LDA $202         ;STATUS BYTE
0041  DF0D  A2 08              LDX #$08
0042  DF0F  85 FC              STA $FC
0043  DF11  26 FC       LOOP   ROL $FC          ;BITS TESTEN
0044  DF13  90 02              BCC NULL
0045  DF15  B0 03              BCS EINS
0046  DF17  A9 30       NULL   LDA #'0
0047  DF19  2C                 .BYT $2C         ;DUMMY
0048  DF1A  A9 31       EINS   LDA #'1
0049  DF1C  20 D2 FF           JSR WRITE
0050  DF1F  CA                 DEX
0051  DF20  D0 EF              BNE LOOP
0052  DF22  60                 RTS
0053  DF23
0054  DF23             ;COPY"NAME",D1TOD0
0055  DF23             ;=================
0056  DF23
0057  DF23                    *= 55994
0058  DABA
0059  DABA  20 D3 DE           JSR NERROR       ;EX 55325
0060  DABD
0061  DABD                    *= 56344
0063  DC18  20 C4 DE           JSR NCOPY        ;EX 56033
0064  DC1B
0065  DC1B                    *= $DEC4
0066  DEC4
0067  DEC4  AD 3E 03    NCOPY  LDA 830
0068  DEC7  29 03              AND #3
0069  DEC9  C9 03              CMP #3
0070  DECB  D0 03              BNE COPY1
```

```
0071  DECD  4C E1 DA          JMP 56033
0072  DED0  4C D4 DA    COPY1 JMP 56020
0073  DED3  29 C4       NERROR AND #196
0074  DED5  D0 01              BNE ERROR1
0075  DED7  60                 RTS
0076  DED8  4C FF B3    ERROR1 JMP 46079
0077  DEDB
0078  DEDB              ;BASIC4 IN DIRECTORY
0079  DEDB              ;====================
0080  DEDB
0081  DEDB              CURADR = $C4
0082  DEDB              FROM   = $5E
0083  DEDB              CHRGET = $76
0084  DEDB              DIRFLG = $37
0085  DEDB              BUFEND = $77
0086  DEDB
0087  DEDB                     *=$DDB3          ;(56755)
0088  DDB3
0089  DDB3  4C CF DF           JMP DIRLOD
0090  DDB6
0091  DDB6                     *=$DE20          ;(56864)
0092  DE20
0093  DE20  4C CF DF           JMP DIRLOD
0094  DE23
0095  DE23                     *=$DFCF          ;(57327)
0096  DFCF
0097  DFCF  A5 37       DIRLOD LDA DIRFLG
0098  DFD1  F0 2A              BEQ ENDE
0099  DFD3  A5 C4              LDA CURADR       ;CLEAR SCREEN AB
0100  DFD5  85 5E              STA FROM         ;CURSOR ZEILE NACH
0101  DFD7  A5 C5              LDA CURADR+1     ;DEM COMMAND
0102  DFD9  85 5F              STA FROM+1
0103  DFDB  A0 00              LDY #0
0104  DFDD  A9 20              LDA #' '
0105  DFDF  91 5E       CLR0   STA (FROM),Y
0106  DFE1  E6 5E              INC FROM
0107  DFE3  D0 02              BNE CLR1
0108  DFE5  E6 5F              INC FROM+1
0109  DFE7  A6 5F       CLR1   LDX FROM+1
0110  DFE9  E0 87              CPX #$87         ;4032 -> 83
0111  DFEB  D0 F2              BNE CLR0
0112  DFED  A6 5E              LDX FROM
0113  DFEF  E0 D0              CPX #$D0         ;4032 -> E8
0114  DFF1  D0 EC              BNE CLR0
0115  DFF3  A9 00              LDA #0
0116  DFF5  A8                 TAY
0117  DFF6  91 77       ENDLOP STA (BUFEND),Y   ;SET BUFFER ENDE =000
0118  DFF8  C8                 INY
0119  DFF9  C0 03              CPY #3
0120  DFFB  D0 F9              BNE ENDLOP
0121  DFFD  4C 76 00    ENDE   JMP CHRGET
0123  E000              ;CATALOG D0,"NAME*"
0124  E000              ;====================
0125  E000
0126  E000                     *= 55418
0127  D87A
0128  D87A              ;SYNTAXPRUFROUTINE UMGEHEN
0129  D87A
0130  D87A  4C 29 DF           JMP TEST
0131  D87D
0132  D87D                     *= $DF29
0133  DF29
0134  DF29  AD 3E 03    TEST   LDA 830          ;SYNTAX FLAG
0135  DF2C  29 01              AND #$01         ;DATEINAME VORHANDEN?
0136  DF2E  D0 03              BNE NAME         ;JA -> AUSWERTEN
0137  DF30  4C 7D D8           JMP 55421        ;WEITER WIE BISHER
0138  DF33  20 E1 DB    NAME   JSR 56289        ;DS$ LOSCHEN
0139  DF36  A9 24              LDA #'$'         ;$ AS ERSTES ZEICHEN
```

```
0140  DF38  8D 53 03          STA 851          ;IN DEN BEFEHLSTRING
0141  DF3B  AD 3B 03          LDA 827          ;DRIVE HOLEN
0142  DF3E  09 30             ORA #$30         ;ASCII ZEICHEN
0143  DF40  8D 54 03          STA 852          ;UND IN BEFEHLSTRING
0144  DF43  A9 3A             LDA #':          ;TRENNZEICHEN
0145  DF45  8D 55 03          STA 853          ;EINFUGEN
0146  DF48  EE 41 03          INC 833          ;STRINGLANGE FUR LOOP
0147  DF4B  A0 00             LDY #0           ;UM EINS ERHOHEN
0148  DF4D  B9 42 03    LOOPA LDA 834,Y        ;NAMEN UBERTRAGEN
0149  DF50  99 56 03          STA 854,Y        ;IN FILENAME BUFFER
0150  DF53  C8                INY
0151  DF54  CC 41 03          CPY 833          ;ALLES UMGESPEICHERT
0152  DF57  D0 F4             BNE LOOPA        ;WEITER UMLADEN
0153  DF59  AE 41 03          LDX 833          ;STRINGLANGE +1 HOLEN
0154  DF5C  E8                INX              ;FUR $
0155  DF5D  E8                INX              ;FUR DRIVENUMMER
0156  DF5E  20 4C DC          JSR 56396        ;PARAMETER IN ZERO PAGE
0157  DF61  4C 8D D8          JMP 55437        ;WEITER IN CATALOG ROUTINE
0158  DF64
0159  DF64
0160  DF64                    ;AUSGABE VON START UND ENDADRESSE BEIM DIREKTEN
0161  DF64                    ;LOAD VON DISKETTE UND SETZEN VON DEVICE 8  BEI
0162  DF64                    ;LOAD , SAVE UND VERIFY.   ABHANGEN VON 'OUT OF
0163  DF64                    ;MEMORY ERROR'  NACH  LADEADRESSE >       $8000
0164  DF64                    ;=============================================
0165  DF64
0166  DF64                    BSOUT  = $E202
0167  DF64                    DIREKT = 62289
0168  DF64
0169  DF64                    ;AUSGABE DER STARTADRESSE
0170  DF64
0171  DF64                          *= 62348
0172  F38C
0173  F38C  20 60 FD                JSR STAADR
0174  F38F
0175  F38F                    ;AUSGABE DER ENDADRESSE
0176  F38F
0177  F38F                          *= 62406
0178  F3C6
0179  F3C6  A5 FC                   LDA 252          ;LADEZEIGER SAVEN
0180  F3C8  A6 FB                   LDX 251
0181  F3CA  20 76 FD                JSR ENDADR
0182  F3CD  EA                      NOP
0183  F3CE  20 AE F1                JSR 61870        ;UNTALK
0185  F3D1                          *= 62597
0186  F485
0187  F485  20 87 FD                JSR DEVICE
0188  F488
0189  F488                          *= $FD60
0190  FD60
0191  FD60  20 6D F4    STAADR JSR 62573         ;LOADING BZW. VERIFYING
0192  FD63  20 51 F3           JSR DIREKT
0193  FD66  D0 0D              BNE ENDSTR
0194  FD68  A9 20              LDA #' '         ;SPACE AUSGEBEN
0195  FD6A  20 02 E2           JSR BSOUT
0196  FD6D  20 17 D7           JSR 55063
0197  FD70  A9 2D              LDA #'-
0198  FD72  20 02 E2           JSR BSOUT
0199  FD75  60          ENDSTR RTS
0200  FD76
0201  FD76  C9 80       ENDADR CMP #$80         ;LADEADRESSE > $8000
0202  FD78  B0 04              BCS NOCHGE       ;NICHT SPEICHERN
0203  FD7A  85 CA              STA $CA
0204  FD7C  86 C9              STX $C9
0205  FD7E  20 51 F3    NOCHGE JSR DIREKT       ;DIREKT MODE?
0206  FD81  D0 03              BNE ENDEND
0207  FD83  20 17 D7           JSR 55063
0208  FD86  60          ENDEND RTS
```

```
0209  FD87
0210  FD87                DEVICE
0211  FD87  A2 08               LDX #8          ;LOAD-SAVE ON DEV. 8
0212  FD89  86 D4               STX 212
0213  FD8B  A2 01               LDX #1
0214  FD8D  60                 RTS
0215  FD8E
0216  FD8E`                    .END
```

Bücher

Schneider, Wolfgang: Strukturiertes Programmieren in BASIC. Braunschweig, Vieweg-Verlag 1985, 371 S., br. DM 48,--, ISBN 3-528 04320-2. Der als Professor an der FHS Wolfenbüttel tätige Autor legt wiederum ein gründlich erarbeitetes Buch vor, das durch seine klaren Darstellungen besticht. Oft genug ist erwähnt worden, daß BASIC einen Wildwuchs im Programmierstil nicht verhindert, und es sind Vorschläge gemacht worden, wie mit mehr Planung strukturierte Problemlösungen geschrieben werden können, die übersichtlicher und besser zu warten sind. Zum vorliegenden Buch wird bemerkt, daß es keine Vorkenntnisse erfordert. Vier einleitende Kapitel geben dem Anfänger eine Übersicht. Es folgen im handwerklichen Teil Abschnitte zur Entwicklung von strukturierten Anwenderprogrammen und Darstellungstechniken zur Beschreibung der Syntax von Programmiersprachen. Hier sind die beschriebenen 'Fahrnetze' interessant, die zusammen mit Struktogrammen nun im Hauptteil des Buches auf BASIC und seine Sprachelemente angewendet werden. Nach Kenntnis des Rezensenten gibt es wenigstens in deutscher Sprache kein BASIC-Buch, das diese strukturierenden und durchfahrenden Darstellungstechniken in ähnlicher Weise konsequent benutzt und auch auf vollkommen ausprogrammierte Lösungen anwendet. Der Fluß der Darstellungen ist dabei durch grafische Techniken gegliedert: Hervorhebungen, Schemazeichnungen und Umrandung wichtiger Beschreibungen und Merksätze, womit sich eine gute Lesbarkeit ergibt. - Das Buch darf nicht nur dem ernsthaft sich einarbeitenden Anfänger empfohlen werden, sondern auch dem erfahrenen Programmierer, der hier einiges zur Verbesserung seiner Techniken lernen kann.

Watt, Daniel: LOGO, Computersprache für Eltern und Kinder. te-wi-Verlag München 1984, 366 S., DM 59,-, ISBN 3-921803-20-9. Das amerikanische Original wurde in den USA zum Buch des Jahres gekürt. Es ist für Kinder und ihre Eltern oder Erzieher (Schulen) gedacht, die 'in das Reich der Schildkröte' eindringen wollen, auch in gemeinsamen Bemühen. Für junge Menschen dürften die vielen grafischen Anwendungen der Sprache Anreize zur eigenen Betätigung am Computer geben. Alle auszuführenden Schritte, auch die der Hantierung, werden ausführlich erklärt und mit vielen lustigen Zeichnungen für das kindliche Gemüt unterlegt, und es gibt durchgehende grafische Symbole für 'Reinfall', 'Erkundung', 'tolle Idee' und 'Patnertip' (für den Erzieher), mit denen das Lernen animiert wird. Da es bei LOGO inzwischen verschiedene Versionen gibt, ist das Buch vorwiegend für Commodore-LOGO geschrieben, versehen mit Hinweisen für die Versionen auf Apple, Atari und IBM. Der Anhang enthält entsprechende Hinweise für TI-LOGO, so daß alle wichtigen Computer abgedeckt sind, die für diese Sprache in Betracht kommen. Für jugendliche Leser dürfte dieses Buch zu den ausführlichsten gehören, die eine sehr eigenständige Einarbeitung in die Computerei ermöglichen und dabei Spaß und Spiel vermitteln.

Editorial

Bei der Softwarearbeit beschäftigte den Herausgeber in den vergangenen Monaten häufig die Frage nach dem zweckmäßigen Programmierstil. Es geht um die Frage, in welchem Umfang man auf Programmteile aufbauen soll, die im Computer bereits im Betriebssystem vorhanden sind und die man als Dienstleistung für eine Problemlösung benutzen kann.

Tatsache ist, daß es in den Zeitschriften für alle geläufigen Rechner Vorschläge gegeben hat, wie das eine und das andere zu verbessern sei. Das spricht dafür, daß kein Rechner auf Anhieb alle vorkommenden Wünsche abdeckt. Oft auch schieben Hersteller verbesserte Versionen des Betriebssystems nach, die dann neue Einsprungspunkte haben, so daß frühere eigene Adressenbezüge nicht mehr verwertbar sind.

Mögliche Änderungen kann man als geringfügige und als umfangreiche klassifizieren. Erstere haben den Vorteil, daß sie ein Problem auf die schnelle Art erledigen. Wo aber viele solche gering-

fügigen Änderungen zusammenkommen, wie z.B. für den AIM oder für Commodore-Rechner, da führt das zu einer 'Landschaftszersiedelung': Die Änderungen werden da untergebracht, wo noch etwas freier Platz ist, auch versprengt im Adressenraum. Es führt dazu, daß jeder Betreiber möglicherweise schließlich sein individuelles System hat und daß zusammenhängende Teile der Dienstleistungen schlecht dokumentiert und zerrissen sind. Der Autor hält das für gefährlich. Sicher, wo man mit wenigen Handgriffen etwas anpassen kann, da sollte man es tun.

Es spricht aber auch manches dafür, verbesserungswürdige Teile als ein zusammenhängendes Programm zu schreiben, das nur zu einem Minimum auf Systemroutinen zurückgreift. Solche wären die des auf jedem Computer vorhandenen BIOS (Basic Input Output System). Das mag zwar dazu führen, daß einige Dienstleistungen zweimal auf der Maschine sind, einmal in der Firmware und einmal im problemlösenden Programm des Anwenders. Was man hier als Verschwendung von Mühe und von Platz ansehen könnte, hat gleichwohl auf längere Sicht seine Berechtigung:

Man sollte kein Programm als endgültig und fertig ansehen. Man findet selber Verbesserungen zur eigenen Arbeit oder man erhält Anregungen von außen. Und plötzlich sind früher ausreichende Lücken zu klein geworden. Änderungen ziehen zwangsläufig weitere 'Verschlimmbesserungen' nach sich.

In Hinsicht darauf meine ich empfehlen zu können, lieber zusammenhängende Erledigungen zu schreiben, die leichter gewartet und auch übertragen werden können. Man hat dann auch mehr gedankliche Freiheit, gründlich zu arbeiten, und starrt nicht wie gebannt auf Systemroutinen, die im Einzelfall vielleicht auch noch verbesserungswürdig sind. - Wenn man diesen Weg verfolgt, dann kommt es wohl wesentlich darauf an, daß man sich bei Änderungen an passender Stelle in das System 'einhängen' kann., z.B. in Form eines Aussprunges. - Das vorliegende Heft enthält zwei Disassembler. Der eine bemüht sich, eine eigenständige Lösung zu finden, deren Prinzip man auf andere Rechner übertragen kann und deren Leistungen man für eine andere CPU analog verwerten könnte. Die andere ist im Bereich der größeren Änderungen zu sehen, die sehr auf das System bezogen sind. Für welche Lösung man sich entscheidet, hängt von den Erwartungen ab. Z.B. wird man bei der R65C02 bleiben oder wird man sich mit weitere Typen befassen?

Es soll nicht verkannt werden, daß die vielen kleinen Änderungen schließlich einmal zu einer großen führen. Wenn sich jemand hinsetzt, um die Fäden zusammenzuknüpfen. In diesem Sinne sind die Leser weiterhin gebeten, in der einen oder anderen Form zum Inhalt der Hefte beizutragen.

Bezüglich der neuen CPU-Typen 65802/816 ergeben sich viele neue Programmiermöglichkeiten, für die dieses Heft bereits eine Reihe von Anregungen enthält, die einerseits die Leistungsfähigkeit belegen und für die es wohl in noch keiner anderen Zeitschrift irgendwelche Gegenstücke gibt. Richtig wird man die Neuerungen erst ausschöpfen und bewerten können, wenn es Systeme gibt, die mit mehreren Speicherbänken arbeiten. Hier ist zu vermelden, daß an drei dem Autor bekannten Stellen an entsprechender Hardware gearbeitet wird, die in einem Falle schon zur Hannover-Messe präsentiert werden soll. Man wird weiter davon hören.

Rekursionen in FIG FORTH. Zu diesem in Heft 40 veröffentlichten Artikel und Programm folgende Korrekturen: S. 35, 3. Absatz. Dort muß es heißen 'Ab dann wäre das Wort unbekannt'. Einige Zeilen weiter: 'RECUR (s.o.) dürfte ...'. Im Programm auf Seite 37 muß es heißen '2 CONSTANT D.DP'.

Aus der Branche

VALVO: 68000-Workshop 5.-7.3.85 in Hamburg. – Das bekannte statische CMOS-RAM PCD 5101P mit 256x4-Organisation ist jetzt auch im Miniaturgehäuse SO-24 für die SMD-Technik lieferbar und hat dann die Bezeichnung PCD5101T.

Neues 256k CMOS EPROM von Toshiba. Das TC57256D ist zu 32kx8 organisiert und hat Zugriffszeiten von 250 und 200 ns, Standby-Stromaufnahme 100 uA. Datenblätter vin bitronic GmbH.

Computer Aided Geometric Design ist eine neue Fachzeitschrift in englischer Sprache speziell für die Gebiete CAD/CAM in einer mehr mathematischen Betrachtungsweise. Probeexemplar bei

North Holland, A Division of Elsevier Science Publishers BV, attn. Karin van Schouten, PO Box 1991, NL-1000 BZ Amsterdam.

Commodore PC 1o und PC 20. Der seit der Hannover-Messe 1984 angekündigte Personal Computer mit Kompatibilität zum IBM-PC wird seit Anfang Februar an die Commodore-Händler ausgeliefert. Das Gehäuse ähnelt nun doch mehr dem des IBM und ist ansprechend. Deutsche Tastatur, CPU 8088 mit 8087 optinalem Coprozessor, 256 K RAM Standard, RAM-Erweiterung intern 512/640 KB. Schnnittstellen Centronics, RS232, Tastatur. Video-Karte monochrom mit 80 Zeichen in 25 Zeilen. Sehr gut lesbare Zeichen in 9x14 Punktmatrix. Angenehmer gelb-grüner Phosphor des BS-Monitors. Farbkarte kann nachgerüstet werden. 2 Diskettenlaufwerke 2x360 KB, DOS 2.11. Es heißt, daß praktische alle IBM-Programme abgefahren werden können. Schwierigkeiten soll es noch mit der Freigabe des Microsoft BASIC geben. Preis DM 4950,- + MWSt. – Das Modell PC 20 mit 1 Floppylaufwerk und einer Winchesterplatte mit 10 MB soll ab Hannover-Messe zur Auslieferung gelangen. **Neu kommt auch der PC 128.** Es handelt sich um einen erweiterten C-64 mit 128 KB RAM und BASIC 7.0. Daneben kann er auch im C-64-Modus gefahren werden und dessen Software weiterbenutzen. Mit Anschluß eines Z80 läuft er im CP/M 3.0 Modus. Im Modell -D ist ein Diskettenlaufwerk in der Console mit untergebracht. Es handelt sich um das auch frei aufstellbare Modell 1741 mit 470 KB pro Platte (neues Format!), das zehnmal schneller als die 1541 arbeitet. Im CP/M-Mode hat man 80 Zeichen je Zeile auf dem Schirm. Die Konsole des Computers ist formschön gelungen. Man rechnet mit einem Preis, der bei DM 1300 liegen könnte + Floppy. Der Rechner wird auf der Hannover-Messe vorgestellt und soll dann sofort lieferbar sein.

SOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOSOS

Dipl.-Ing. H. Grunewald, 2150 Buxtehude

DISMOS V1.0

Zum sinnvollen Arbeiten mit einer CMOS-Version der 6502 gehört neben einem entsprechenden Assembler /1/, /2/ auch ein Disassembler. Dieser sollte den vollständigen Befehlssatz der Rockwell CPU R65C02 beherrschen. Eine Einbindung in das Betriebsprogramm ist dabei unbedingt erforderlich, um die Funktionen anderer Programme, z.B. symbolische Disassembler, nicht zu stören.

Das Programm DISMOS V1.0 erfüllt diese Kriterien. Hierbei wurde eine Idee aus /3/ aufgegriffen und für das Betriebssystem des AIM 65 aufgearbeitet.

Literatur:
/1/ R65C02-Assembler: Roland Löhr, Herausgeber des MICRO MAG
/2/ M. Albrecht: ASSMOS 1.0, 65xx MICRO MAG Nr. 36
/3/ H. Siggemann: Erweiterung für den AIM 65-Disassembler, MC 1/85

```
DEFINITIONEN

0000 LENGTH          =$EA
0000 REG             =$116
0000 SAVPC           =$A425
0000
0000 GM              =$E7D4
0000 BLANK           =$E83E
0000 OUTPUT          =$E97A
0000 NUMA            =$EA46
0000 NOUT            =$EA51
0000 PCLLD           =$EB56          ;LDA ($A425),Y
0000
-------------------------------------------------------------

EINBINDUNG

0000                 *=$F495
F495        6C4E02 JMP START
F498        EA     NOP
```

```
-------------------------------------------------------------
HAUPTPROGRAMM

F499                *=$7000
-------------------------------------------------------------
neue Befehle

7000 START  A000    LDY #0
7002        2056EB  JSR PCLLD       ;LDA ($A425),Y
7005        2046EA  JSR NUMA
7008        203EEB  JSR BLANK
700B        2056EB  JSR PCLLD       ;LDA ($A425),Y
700E HEX1   D98671  CMP CODE,Y      ; Code-Tabelle
7011        F01D    BEQ OPCODE      ; Code bekannt?
7013        C8      INY
7014        C01B    CPY #$1B
7016        D0F6    BNE HEX1
7018        298F    AND #$8F        ; Filter für RMB/SMB, BBR/BBS
701A HEX2   D98671  CMP CODE,Y      ; Code-Tabelle
701D        F011    BEQ OPCODE      ; Code bekannt?
701F        C8      INY
7020        C01F    CPY #$1F
7022        D0F6    BNE HEX2
7024        20D4E7  JSR QM          ; nein, Code unbekannt '???'
7027        20D4E7  JSR QM
702A        20D4E7  JSR QM
702D        4C0071  JMP ZER
7030
-------------------------------------------------------------
ja, Code gefunden

7030 OPCODE 8C1601  STY REG         ; Position in Code-Tabelle
7033        B96771  LDA ASCPOS,Y    ; X-Index für ASCII-Tabelle holen
7036        AA      TAX
7037        A003    LDY #3
7039 LOOP1  BDA571  LDA ASCTAB,X    ; 3x ASCII ausgeben
703C        207AE9  JSR OUTPUT
703F        E8      INX
7040        88      DEY
7041        D0F6    BNE LOOP1
7043        AE1601  LDX REG         ; Code-Position in X
7046        BDE271  LDA SBRTAB,X    ;Index für Sprundweite holen
7049        8D1901  STA REG+3
704C        204F70  JSR OPERAN
704F OPERAN 18      CLC
7050        68      PLA
7051        6D1901  ADC REG+3
7054        8D1A01  STA REG+4
7057        68      PLA
7058        6900    ADC #0
705A        8D1B01  STA REG+5
705D        6C1A01  JMP (REG+4)     ; Disassembliere d. Operand
7060
-------------------------------------------------------------
DISASSEM. NACH SPALTEN (COL)

7060 COLA2  4C0071  JMP ZER         ; PHY, PLY, PHX, PLX
7063
7063 COL41  203EEB  JSR BLANK       ; TSB, TRB, STZ ZP
```

```
7066        200D71 JSR NEXT       ; Zero-Page ausgeben
7069        4CFA70 JMP ONE
706C
706C COLC1  203EE8 JSR BLANK      ; TSB, TRB, STZ ABS
706F        200571 JSR NEXT2      ; Absolut-Adresse ausgeben
7072        4CFD70 JMP TWO
7075
7075 COL42  203EE8 JSR BLANK      ; BIT, STZ ZP,X
7078        200D71 JSR NEXT       ; Zero-Page ausgeben
707B        201671 JSR KOMMAX     ; ',X' ausgeben
707E        4CFA70 JMP ONE
7081
7081 COLCE  203EE8 JSR BLANK      ; BIT, STZ ABS,X
7084        200571 JSR NEXT2      ; Absolut-Adresse ausgeben
7087        201671 JSR KOMMAX     ; ',X' ausgeben
708A        4CFD70 JMP TWO
708D
708D COL2   203EE8 JSR BLANK      ;ORA, AND, EOR, ADC, STA, LDA, CMP, SBC (IND)
7090        A928   LDA #'('
7092        207AE9 JSR OUTPUT
7095        200D71 JSR NEXT       ; Zero-Page ausgeben
7098        A929   LDA #')'
709A        207AE9 JSR OUTPUT
709D        4CFA70 JMP ONE
70A0
70A0 COL9   203EE8 JSR BLANK      ; BIT IMM
70A3        A923   LDA #'#'
70A5        207AE9 JSR OUTPUT
70A8        200D71 JSR NEXT
70AB        4CFA70 JMP ONE
70AE
70AE COLA1  203EE8 JSR BLANK      ; INC, DEC .A
70B1        A92E   LDA #'.'
70B3        207AE9 JSR OUTPUT
70B6        A941   LDA #'A'
70B8        207AE9 JSR OUTPUT
70BB        4C0071 JMP ZER
70BE
70BE COLC2  203EE8 JSR BLANK      ; JMP (IND,X)
70C1        A928   LDA #'('
70C3        207AE9 JSR OUTPUT
70C6        200571 JSR NEXT2      ; Absolut-Adresse ausgeben
70C9        201671 JSR KOMMAX     ; ',X' ausgeben
70CC        A929   LDA #')'
70CE        207AE9 JSR OUTPUT
70D1        4CFD70 JMP TWO
70D4
70D4 COLF   202171 JSR BITNUM     ; BBR, BBS0...7
70D7        200D71 JSR NEXT       ; Zero-Page ausgeben
70DA        203EE8 JSR BLANK
70DD        EE25A4 INC SAVPC
70E0        D003   BNE AC0
70E2        EE26A4 INC SAVPC+1
70E5 AC0    203371 JSR REL        ; Sprungadresse ausgeben
70E8        4CFA70 JMP ONE
70EB
70EB COL1   203EE8 JSR BLANK      ; BRA RELATIVE
70EE        203371 JSR REL        ; Sprungadresse ausgeben
70F1        4CFA70 JMP ONE
```

Fortsetzung der Programm-Liste im nächsten Heft.

Assembler

R65C02-Assembler

für AIM 65 und kompatible Systeme. 2-Pass-Assembler für den kompletten (!) Befehlssatz mit zusätzlichem Komfort. Assemblerliste formatiert, Assemblierung mit Offset für virtuelle Speicherbereiche. Der Assembler läuft auf der herkömmlichen 6502. - 2 EPROMs für Speicherbereiche Ihrer Wahl DM 195,--

6805/68705 Cross-Assembler

für AIM 65 und kompatible Systeme. 2-Pass-Assembler mit allem gewohnten Komfort und 6502-Syntax. Erzeugt aus den Mnemonics von Motorola 6805-Code. 2 EPROMs wie vor DM 280,--

6805/68705-Assembler unter FORTH

wie in Heft 35 beschrieben: 1-Pass-Assembler mit Verarbeitung von Symbolen und Labeln. Codeablage für virtuelle Speicherräume. 2 EPROMs für AIM 65 DM 100,--

Mathe-ROM für 6502

Implementierung nach Peter Rix (s. Hefte 28/29). Fließkommaarithmetik und höhere mathematische Funktionen wie in Microsoft-BASIC für AIM-65-FORTH und für jedes 6502-Assemblerprogramm (20 S. Dokumentation mit Einsprungspunkten und Argumenten), für Sockel $D000 DM 124,30

Neu: Assembler für 65802/816, für R65C02 und für 6502

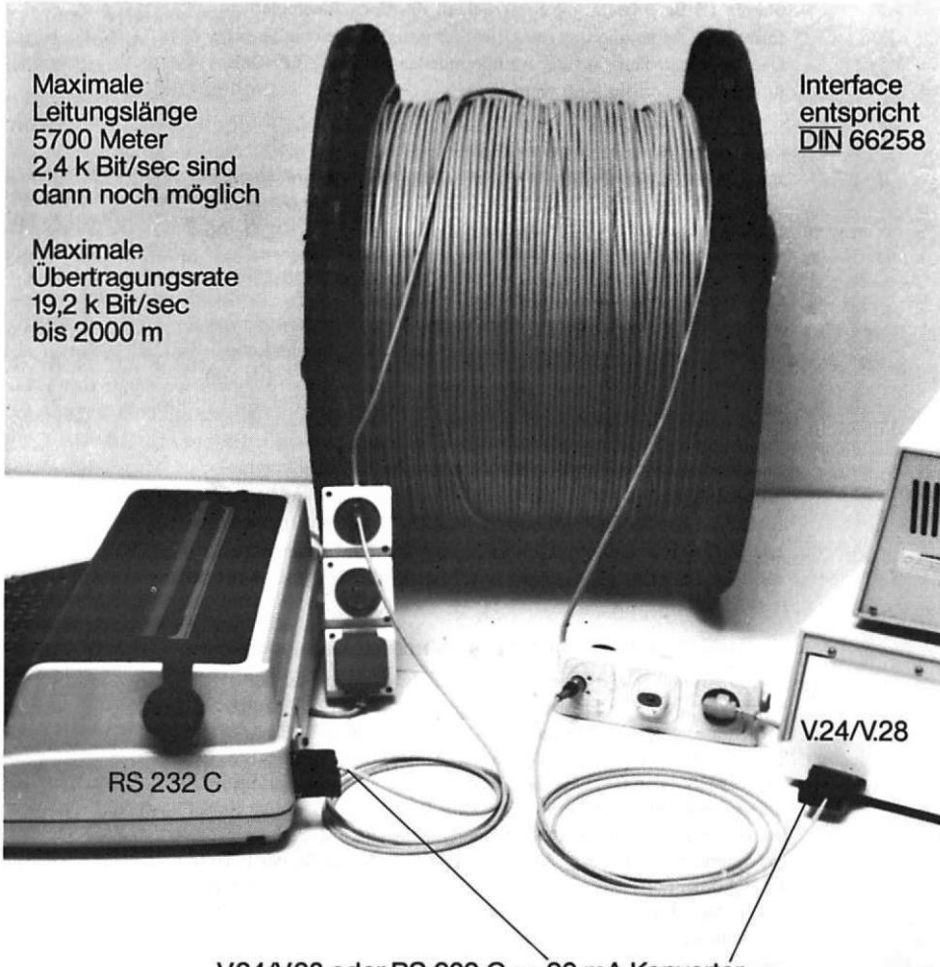
2 Pass Assembler mit einstellbaren Befehlssätzen für jede CPU zur Vermeidung von Irrtümern. Geschrieben in 6502-Code, damit auf vorhandenen Systemen einsetzbar. Generierbar für beliebige Adreßbereiche. Standardmäßig lieferbar für AIM 65 und kompatible (für SYSTEM 65 a. Anfr.). Für andere Fälle dokumentierte Sprungleiste auf externe Routinen des E/A-Systems. Die Syntax folgt dem Rockwell-Assembler. Symboltafel alphabet. geordnet mit Zeilenangabe, wo definiert. Formatierte Assembler-Liste mit 80 Zch/Z, wie in Heft 41 dokumentiert. Kommandofiles möglich, die Quelltexte von verschiedenen Eingabemedien verbinden. Damit zügige Durchläufe mit Library-Funktion. Assemblerung mit Offset in beliebige Bereiche als Vorlage für EPROMs. Für AIM 65, inkl. Dokumentation, Preis inkl. MWSt DM 300,--

Cross-Assembler für 6804

in Vorbereitung. Lieferbar etwa Ende März 1985. Bitte anfragen

Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg
Tel.: 04 102 - 55 816

Suchen Sie eine preiswerte Alternative zur Dezentralisierung Ihrer Datenverarbeitung? . . . dann ist 20 mA Current Loop die Lösung für Ihr lokales Punkt zu Punkt Netzwerk!



V.24/V.28 oder RS 232 C ↔ 20 mA Konverter
eingebaut in einem serienmäßigen Subminiatur „D" Steckergehäuse

Wir beraten Sie und planen Ihr lokales Netzwerk!
Fordern Sie bitte weiteres Informationsmaterial an.

DBGM: G 8331 081.9
G 8336 080.8

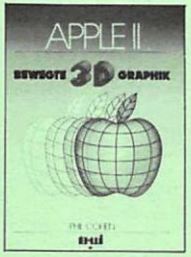
INGENIEURBÜRO
STECKER

5000 Köln 60 (Niehl)
Postfach 60 07 66
Delmenhorster Str. 20
Tel. (02 21) 712 4018

APPLE II –
Anwenderhandbuch
(L. Poole)
Erst mit Hilfe dieses Leitfadens werden Sie Ihren Apple II erfolgreich einsetzen, denn Text und Bildmaterial gehen weit über das hinaus, was herstellerseitig an Literatur angeboten wird.
416 Seiten, Softcover, DM 56,–

LOGO –
Jeder kann programmieren
(Daniel Watt)
Buch des Jahres in den USA. Für die Computer C64, ATARI, APPLE II, IBM-PC und TI-99.
Hochwertiges Textbuch für Logo-Kurse für zu Hause und im Lehrbereich.
A4, DM 59,–

APPLE II PASCAL –
Eine praktische Anleitung
(A. Luehrmann, H. Peckham)
Unentbehrlich für alle, die die Programmiersprache PASCAL lernen wollen und Zugang zu einem Apple-Computer haben.
544 Seiten, Softcover, DM 59,–



APPLE II – Bewegte 3D-Graphik
(Phil Cohen)
Selbstentworfene Graphiken und Diagramme – animiert oder als Standbilder – eben oder räumlich: alle erforderlichen BASIC-Programme mit Erklärung finden Sie in diesem Buch.
ca. 190 Seiten, Softcover, DM 49,–
(2. Quartal 85)

Computer für Kinder
(Sally Greenwood Larson)
Ein Buch für Kinder, ihre Lehrer und Eltern.
„Computer für Kinder" richtet sich an Kinder im Alter von 8 bis 13 Jahren, für deren Interesse an Computern dieses Buch bewußt geschrieben wurde.
Unterhaltsam und leicht verständlich.
A4 quer, Fadenheftung, DM 29,80

Apple Maschinensprache
Für BASIC-Programmierer der einfachste Zugang zur Muttersprache des Apple. Wesentlich schnellere Maschinenprogramme, direkte Manipulation des Mikroprozessors 6502 im Apple – als Brücke dorthin benötigt dieses Buch nur die drei BASIC-Befehle, POKE, CALL, PEEK. D. Inman/K. Inman. DM 49,–

te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

te-wi

tm 4153

Noch im Programm:
6502 – Programmieren in Assembler DM 59,–
VisiCalc, 50 Programme auf Diskette, DM 79,–

In Vorbereitung:
Macintosh Programmier-Handbuch DM 59,–

MICRO MAG

HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☎ (04102) 5 58 16

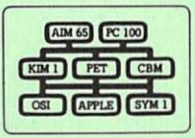
MICRO MAG (vormals 65xx MICRO MAG) erscheint zweimonatlich, jeweils Mitte Februar, April usw.. COPYRIGHT 1984 by Roland Löhr. Alle Rechte vorbehalten, auch die des auszugsweisen Nachdruckes, der Übersetzung, der fotomechnischen Wiedergabe und die der Verbreitung auf magnetischen und sontigen Trägern. Von den veröffentlichten Programmen, Schaltungen und Angaben wird ohne eine Gewährleistung von hier aus angenommen, daß sie fehlerfrei und frei von den Schutzrechten Dritter sind. Beiträge, die nicht besonders gekennzeichnet sind, stammen vom Herausgeber. Offsetdruck: L & L Druckservice, Hamburg 73

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- im Inland, bzw. DM 59,-- im Ausland (surface mail). Luftpostzustellung auf Anfrage. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu 4 Wochen vor deren Ablauf. - Nachliefermöglichkeiten siehe unten.

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

Leser-Service des Herausgebers

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Nachliefermöglichkeiten:

Vergriffen sind 'Das Buch 1-6 des 65xx MICRO MAG' sowie die Hefte 1-13 der Zeitschrift. Es erfolgt keine Neuauflage.

Lieferbar: 'Das Buch 7-13 des 65xx MICRO MAG' zu DM 42,-- sowie die Hefte 14-40 zu DM 7,80/St. (ab 10 St. in einer Sendung: DM 6,--/St.).

FORTH User's Manual (Lieferung, solange vorrätig)
Rockwell-Handbuch für das FIG-FORTH des AIM 65. Mit der Erläuterung des Befehlssatzes auch für andere FORTH-Betreiber geeignet. Ca. 300 S., engl. DM 30,--

Mathe-ROM nach P. Rix für FORTH des AIM + Assemblerprog. m. Doku DM 124,30

Vorstehende Preise inkl. MWSt, zuzüglich DM 2,50/Sendung + ggfs. NN + DM 2,--

Assembler für R65C02, MC6805 und 6809: Siehe im Heftinneren.