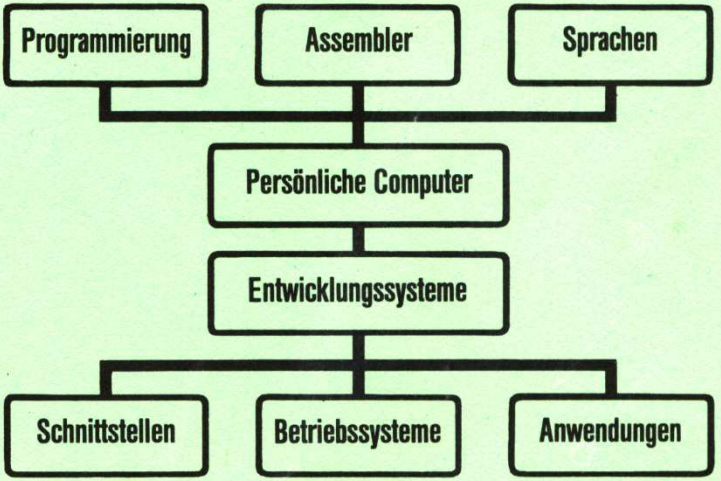
# MICRO MAG

DM 9,50 Nr. 46 Mai 1986




## Inhaltsverzeichnis



**dBASE III: Einführung und Referenz**
Die deutsche dBASE III-Version – an Hunderten von Beispielen illustriert, in Moduln lexikalisch dargestellt, mit schnell orientierenden Befehlserklärungen.
Ein Mehrzwecktext:
**Als Einführung** in kurserprobten Lernsequenzen lesbar.
**Als Referenz** in alphabetsicher Befehlsfolge lesbar.

Als dBASE III-Kurstext entwickelt, nutzt dieses Buch die beiden Anwendungsvarianten von dBASE: über **Direktbefehle** und über **Befehlsfolgen in Programmform** benutzbar zu sein.
Ideal als Kursunterlage durch 2. Inhaltsverzeichnis (guided tour) mit Führung durch Lernmoduln und Erfolgsabfrage. Geschrieben für Erfordernisse der Praxis.
**Russel Stultz, ca. 464 Seiten, Softcover, reiches Bild-/Beispielmaterial, DM 79,–**

**NEU**
**BASIC Programmierung: PC-10/PC-20**
Ein amerikanisch-lockerer BASIC-Kurs von dem kalifornischen Professor Lien. Durch seine Systematik ideal als Kursunterlage für PC-10/PC-20 und Kompatible. Mit Einführung in das PC-10-System und Tastendarstellung im Text; BASIC-Befehlsschreibung mit Aufgaben und Antworten; Sammlung von Beispielprogrammen; BASIC-Befehle für Ton/Graphik/Text; SAVE/MERGE/CHAIN-Anwendungen. Ein leicht lesbares Buch für Erstanwender, systematisch und vollständig für Erfahrene.
**David A. Lien, 488 Seiten, Softcover, DM 59,–**

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

Roland Löhr

# Die Atari-ST Serie

**Die Hardware**

Nach einigen Monaten der Erprobung darf man die neue Atari-Produktlinie lobend besprechen: Es gibt jetzt drei verschiedene Computer-Consolen. Der 260ST ist ein "Aufsteigermodell" mit 1/2 MB Speicher zum Preise von DM 998 (inkl. MWSt) und inklusive eines Fernseh-Adapterkabels. Die Maus und den Bildschirm muß man selber vorhalten oder dazukaufen. Bastler haben sich dem Vernehmen nach aus einem preiswerten Restposten auch selber ein Floppy-Lauferwerk dazu angeschlossen. - Das Modell 520ST+ hat 1 MB Speicher und kostet mit der "kleinen" Floppy (3,5 Zoll, 360 KB Speicher), Maus und Monochrom-Monitor DM 2698. Mit der "großen" Floppy (720 KB) kostet es DM 2998. Seit der CEBIT-Messe in Hannover wird nun auch das Spitzenmodell 1040ST ausgeliefert. Zu DM 3298 bietet es 1 MB Speicher und im Gehäuse eingebaut die "große" Floppy mit 720 KB. Dieses Gerät macht mit den vielen Kabeln und Gehäusen am Arbeitsplatz Schluß, denn auch das Netzgerät ist unter die Haube integriert. - In zahlreichen Demo-Programmen konnte man sich auch von der Farbtüchtigkeit der Geräte überzeugen. Man muß aber bemerken, daß die gestochen scharfe und ruhige monochrome Bildausgabe für die meisten Anwendungen die richtige Wahl sein dürfte.

Neu seit Hannover ist auch die Festplatte im eigenen Gehäuse mit Stromversorgung, Controller und Interface. Sie hat ein Speichervolumen von 20 MB formatiert. Der Anschluß erfolgt über DMA. Dieses Gerät war im April 1986 allerdings noch nicht im Handel zu haben. - Angekündigt wird der MS-DOS-Emulator für den Herbst 1986. Es ist eine eigene Karte, die ebenfalls über DMA angeschlossen werden soll. Sie hat die CPU 8088 mit 8 MHz, BIOS im ROM, 512 KB RAM (die man im Atari-Modus als RAM-Disk betreiben kann) und einen Sockel für den Mathematik-Coprozessor 8087. Sie soll DM 498 kosten. Man kann aber auch eine 5 1/4"-Floppy mitbeziehen und damit die reichlich für MS-DOS-Rechner auf Disketten erhältliche Software direkt ladbar machen. Dann gibt man DM 998 aus.

Das TOS-Betriebssystem des Atari ist bis jetzt noch beim Einschalten des Rechners zu laden. In Kürze sollen jedoch auch die nachsteckbaren ROMs mit dem TOS lieferbar sein. - Wer einen Atari heute bezieht, erhält das Betriebsprogramm CP/M 2.2 zur Emulation der Z80-ZPU und zum Betrieb ihrer Programme kostenlos dazu. - Nach den Ankündigungen auf der Pressekonferenz in Hannover arbeitet man an einem Grafik-Coprozessor für den Rechner, der auch höhere Auflösungen erlauben soll. Anfang 1987 will man dann auch einen UNIX-Rechner anbieten können, der offensichtlich die vorhandenen Computer-Consolen mitbenutzen soll. - Der Herausgeber benutzte in Hannover die Gelegenheit, die aus Kalifornien angereisten Designer noch einmal deutlich darauf hinzuweisen, daß die Tastatur der Rechner verbessert werden muß. Die Ausformung der Tasten und ihr Knackpunkt führen bislang zu einer hohen Fehlerquote, wenn man Text flink eingeben will. Es wurde mir versichert, daß man an einem Re-Design der Tastatur arbeitet. - Wenn man auch das noch geschafft hat, dann darf man mit den Ataris doch recht zufrieden sein und kann sie dann auch in den Betrieben am Arbeitsplatz aufstellen.

**Verkaufserfolge**

In Hannover wurde folgende Verkaufszahlen von der Markteinführung bis Ende Dezember 1985 genannt: 100.000 Stück weltweit, 40.000 in Deutschland. Der Autor glaubt, daß die Zahl für Deutschland deutlich zu hoch ist, denn die Verkäufe der neuen Geräte kamen erst im November/Dezember richtig in Gang. Heute im Frühjahr kann man jedenfalls aber feststellen, daß die genannte

Atari-Serie ganz klar ein großer Verkaufserfolg geworden ist. Es scheint fast so, als wenn die Betreiber, die sich nicht zu einem IBM-PC oder Nachbau entschließen konnten, auf solche Rechner mit CPU 68000 und mit einem hohen Preis-/Leistungsverhältnis geradezu gewartet hätten. - Bis zum Herbst gab es dabei eigentlich nur wenige Demo-Programme außerhalb des Entwicklungspaketes. Inzwischen gibt es etwa 250 Software-Pakete, die man bei Atari in den Katalogen "Softwareübersicht" und "Softwareübersicht, Update I" dokumentiert hat. Man räumt ein, daß auch viel "Stroh" dabei ist. Andererseits darf man als Betreiber vermelden, daß es ganz hervorragende Textsysteme, Assembler und Sprachkompiler gibt. Und man darf die Vermutung äußern, daß die vielen verkauften Systeme eigentlich zwangsläufig den Nachschub guter Software nach sich ziehen.

**Rechnerbetrieb**

Durch die vielen Veröffentlichungen in anderen Zeitschriften dürften dem Leser die wesentlichen Merkmale der Rechner bereits bekannt sein. Daher ganz kurz: Das Betriebssystem TOS muß beim Einschalten (noch) gebootet werden. Als Menue finden wir dann unter GEM die Schreibtischplatte, Desktop genannt, mit Ikonen der Floppy-Laufwerke und der Möglichkeit, mit der Rollkugel-Maus eine Auswahl aus Menues vorzunehmen, die wie eine heruntergelassene Jalousie aufgeschlagen und angetippt werden. Andererseits haben wir die Möglichkeit, das Inhaltsverzeichnis von Floppys mit Knopfdruck an der Maus aufzuschlagen und Programme von dort zu kopieren, zu laden/auszuführen und den Inhalt von Dokumenten zur Anzeige zu bringen. Das läuft wie auf vielen anderen aktuellen Rechnern mit der Benutzeroberfläche GEM. - Der Autor hatte anfangs etwas Bedenken, ob die Maus vielleicht so eine Art überflüssiges Spielzeug sei. Nach wenigen halben Stunden wurde ihm jedoch klar, daß die interaktive Benutzerführung mit GEM und Maus und deutschen Texten ein sicheres Arbeiten ermöglicht, das ein Nachschlagen in Computer-Handbüchern erspart. Man nehme z.B. nur einmal den DOPEN-Befehl bei Commodore zum Vergleich. Für ihn muß man den Dateinamen eintippen, ggfs. auch noch das Laufwerk. Oder bei COPY muß man zwei Namen und Laufwerksbezeichnungen tippen. Unter GEM brauchen wir keinen Text mehr einzugeben. Einzelne Dateien werden mit der Maustaste zur Auswahl "angeklickt" und mehrere Dateien können für das Kopieren oder für das Löschen mit einem Rahmen zusammengefaßt und der gewünschten Bearbeitung unterworfen werden. - Zusammenfassend läßt sich sagen, daß die geschilderte Benutzeroberfläche ein Einlernen in kürzester Zeit auch durch Laien und einen hohen Bedienungskomfort bietet. Und es gibt bereits viele Programme für den Anwender, die von den Möglichkeiten einer solchen Benutzerführung Gebrauch machen.

**Die Teilprogramme des Betriebssystems**

Spätestens mit den Atari-Rechnern kommt auf den Leser eine Flut von neuen Begriffen zu, die uns künftig immer wieder begegnen werden. Daher hier zunächst die wichtigen Erklärungen in kurzen Absätzen:

Das Betriebssystem der Rechner fußt auf dem CP/M-68K von Digital Research für die CPU 68000. Seine niedrigste Ebene ist das BIOS, das Basic Input Output System. Seine grundlegenden Dienstleistungen werden hereingerufen, indem man Parameter auf dem Stack der Maschine übergibt und zuoberst eine Funktionsnummer. Der eigentliche Funktionsaufruf erfolgt mit dem Assemblerbefehl TRAP #13. Dieser Befehl erzeugt einen Software-Interrupt (Exception), für dessen Bearbeitung ein Vektor als Langwort in der RAM-Adresse $B4 steht. Er weist auf Routinen des Betriebssystems, die die übergebene Funktionnummer entschlüsseln und die die gewünschte Tätigkeit per BIOS ausführen. In den meisten Fällen erhält man im Datenregister D0 entweder ein Eingabezeichen, einen Parameter oder einfach eine Antwort zurück, ob die gewünschte Funktion ausgeführt werden konnte.

Das Programm wird nach der Bearbeitung der Exception mit dem Befehl fortgesetzt, der auf TRAP #13 folgt. Es liegt jetzt in der Verantwortung des Programmierers, den Stapelzeiger auf den Wert zurückzuführen, den er vor der Beschickung des Stapels mit Parametern hatte. - Die Übergabeprozedur ist damit anders als beim CP/M-68K. Dort werden Parameter in Registern übergeben, hier beim Atari erfolgt die Übergabe auf dem Stapel. Wir dokumentieren hier kurz die **BIOS-Funktionen** mit ihrem Namen, der symbolischen Bezeichnung ggfs. zu übergebender Parameter, der Funktionsnummer und der Menge Bytes in Klammern, um die der Stapelzeiger nach der Ausführung zu berichtigen ist:

GETMBP Adresse.L, #0, (6). Get Memory Parameter Block. Die Funktion füllt ab Adresse (als Langwort übergeben) drei Langworte mit Zeigern. Sie wird bei der Systeminitialisierung benutzt und hat sonst wohl keine praktische Bedeutung.

BCONSTAT Gerätenummer.W, #1, (4). Eingabestatus eines Gerätes holen (0=Drucker, 1=RS232, 2=Konsole mit Tastatur und Bildschirm, 3=MIDI-Schnittstelle, 4=Tastaturport). Register D0=0, wenn kein Zeichen vorliegt und =$FFFF, also negativ, wenn mindestens 1 Zeichen anhängt.

BCONIN Gerätenummer.W, #2, (4). Ein Zeichen vom Eingabegerät holen (Nummern wie vor). Das Zeichen steht in den Bits 0...7 von D0. Bei Tastatureingabe findet man in den Bits 16...23 den sog. IBM-Scancode.

BCONOUT Gerätenummer.W, Zeichen.W, #3, (6). Ausgabe eines Zeichens auf das Gerät.

RWABS RW-Flag.W, Bufferadresse.L, Menge.W, Sektor-#.W, Gerät.W, #4, (14). Lesen oder Schreiben (0 oder 1) einer Menge Sektoren von/auf Laufwerk (0=A, 1=B) ab Bufferadresse und ab logischer Sektornummer.

SETEXC Vektornummer.W, Vektor.L, #5, (8). Umsetzen eines Ausnahme-Vektors.

TICKCAL, #6, (2). Liefert die Kalibrierung des System-Timers.

GETBPB, Gerätenummer.W, #7, (4). Liefert einen Zeiger auf einen BIOS-Parameterblock mit den Laufwerksparametern des Gerätes.

BCOSTAT, Gerätenummer.W, #8, (4). Liefert D0.W negativ ab, wenn ein Ausgabegerät bereit ist, sonst 0.

MEDIACH, Gerätenummer.W, #9, (4). Prüfung, ob ein Diskettenwechsel vorliegt (D0 = 0=nein, 1=vielleicht, 2=ja).

DRVMAP, #10, (2). Drivemap, binäre Strichliste als Langwort holen. Einsen dort zeigen, daß ein Drive (0...31) verfügbar ist. Die Strichliste wird in der Variablen mit Adresse $4C4 geführt.

KBSHIFT, Modus.W, #11, (4). Mit negativem Modus Holen der Tastendrücke Shift rechts (Bit 0), links (Bit 1), CTRL (2), ALT (3) und Caps Lock (4). Mit positivem Modus entsprechendes Setzen der Bits.

XBIOS-Funktionen. Diese etwa 40 Funktionen werden nach dem gleichen Schema wie beim BIOS aufgerufen, jedoch mit dem Befehl TRAP #14. Es handelt sich ebenfalls um grundlegende Dienstleistungen, die auf die spezielle Hardware der Atari-Rechner zugeschnitten sind. XBIOS steht dabei für Extended BIOS (erweitertes BIOS). Man kann die Routinen in solche der Video-Verwaltung, der Ein- und Ausgabe, der Floppy-Ansprache, der Interruptverwaltung und der

Klangerzeugung gruppieren. Es würde zu weit führen, sie hier alle mit den notwendigen Parametern zu beschreiben.

**GEMDOS-Funktionen.** Die hierarchisch nächst höhere Ebene des Betriebssystems, die die Hardware nicht mehr unmittelbar anspricht, sondern auf dem BIOS und XBIOS aufbaut, wird GEMDOS genannt. Begrifflich müssen wir hier von der graphischen Benutzeroberfläche des GEM unterscheiden. GEMDOS hat nichts mit GEM zu tun. Seine Funktionen werden wie üblich mit Parameterübergaben, einer Funktionsnummer und mit dem Befehl TRAP #1 aufgerufen. Seine etwa 60 Funktionen können wiederum in zeichen- und zeilenorientierte der Ein- und Ausgabe und der Fileverwaltung gruppiert werden. Dazu kommen Befehle zum Abbruch eines Programmes und zum Setzen wichtiger Parameter. - Die Benutzung und der Aufbau dieser Funktionen sind noch einfach, wie man auch am Programm ATARIRSF in diesem Heft sieht. Die genannten Teilprogramme BIOS, XBIOS und GEMDOS bilden zusammen das **TOS** genannte Betriebssystem.

**GEM.** Was sich nach dem Einschalten der Computer graphisch als eine Schreibtischplatte mit Ikonen und Mausbedienung präsentiert, ist eine Leistung des GEM. Die Routinen des GEM werden mit dem Befehl TRAP #2 aufgerufen. Die Übergabeprozedur ist jedoch komplizierter, als bei den bisher genannten Teilprogrammen. Zunächst einmal sind zwei Teile des GEM zu unterscheiden, die auch noch verschiedenen Aufrufbedingungen folgen, das **Virtual Device Interface VDI** und das **Application Environment System AES.** Das VDI stellt dabei im wesentlichen graphische Routinen zur Verfügung, Zeichnen von Linien, geometrischen Grundfiguren, Schraffur usw. Die Funktionen des AES dienen Anwenderprogrammen, um eine "Umgebung" zu schaffen: Beobachtung von Ereignissen z.B. bei der Mausbedienung, Arbeiten mit Auswahlmenues, Öffnen von Fenstern, Boxen usw.

In der Summe mögen VDI und AES etwa 200 Routinen umfassen, die als Library Bestandteil des Entwicklungspaketes sind, das zum Atari an Softwarehäuser geliefert wird. Sie sind aber als Bibliotheksroutinen ebenfalls z.B. auf der Diskette des C-Compilers enthalten, der von GST in England angeboten wird. Dem Programmierer bleibt damit das Nachtippen dieser Routinen erspart, er muß sich lediglich bezüglich der Parameterübergabe ausreichend informieren. - Angesichts der vielen weltweit verkauften Systeme mit GEM-Einbindungen darf man damit rechnen, daß Programme künftig vermehrten Gebrauch vom GEM machen werden, und nicht mehr nur Schrift auf den Bildschirm bringen.

**Software**

Auf die Atari-Kataloge, in denen etwa 200 Programme enthalten sind, wurde schon hingewiesen. Als Beobachter der Szene hat man den Eindruck, daß Software sehr freizügig kopiert und daß der Markterfolg der Rechner sicher dadurch gefördert wird. Was da alles schon im freien Umlauf oder regulär verkauft wird, läßt sich nach den wenigen Monaten seit Erscheinen der Rechner schon nicht mehr überblicken. Vor allem läßt sich auch wenig zur Qualität im einzelnen sagen. Daher ist zu empfehlen, daß man sich als Interessent bei einem leistungsfähigen Händler oder bei Bekannten informieren sollte, um die richtige Wahl treffen zu können. Der Autor hat sich bisher vorwiegend mit Textprogrammen und Assemblern befaßt, zu denen nachstehend Einzelheiten mitgeteilt werden.

Als so ziemlich erstes Textprogramm tauchte im Herbst der Metacomco-Editor auf. Er ist so einfach, daß man auf seinen Gebrauch verzichten sollte. - Freude macht dagegen das Arbeiten mit dem englischen **GST-Editor,** den man einzeln, zusammen mit dem Assembler oder mit dem C-Compiler erhält. Er ist in das GEM-System eingebunden, arbeitet mit Auswahlmenues und der Maus zusammen und benutzt die 10 Funktionstasten des Atari für das Editieren. Mit ihnen kann

man u.a. bildschirmweise im Text vor- und zurückblättern. Besonders angenehm ist es, daß man den Cursor mit der Maus an jede sichtbare Stelle umsetzen kann. Beim Berichtigen von Texten erspart das die Rangierarbeit mit den Cursortasten. Man kann Textstellen markieren, um später zu ihnen zurückzuspringen, und man kann Textblöcke bilden, um sie zu löschen, zu kopieren, umzusetzen oder um sie mit eigenem Namen abzuspeichern. Es können mehrere Textfiles in eigenen Fenstern zugleich dargestellt werden und man hat die Möglichkeit, Passagen aus dem einen File in das andere herüberzukopieren. Natürlich können an die aktuelle Cursorstelle auch ganze Files eingezogen werden.

Nicht so zufrieden ist man mit den Funktionen für das Suchen und für das Suchen und Ersetzen von Strings, die etwas umständlich zu hantieren sind. Sie müssen auch als relativ langsam bezeichnet werden. Diese Aussage gilt auch für die Erneuerung des Bildschirminhaltes z.B. an den Anfang des Textes oder eben auf die gesuchte Stelle. Störender ist eigentlich die schon beanstandete Tastatur des Atari. In der Summe: Mit dem GST-Editor läßt sich sicher und bequem arbeiten, man braucht wegen der Benutzerführung auch nur selten im Handbuch nachzusehen. Die Texte werden als sequentielles File in ASCII-Codierung ausgegeben, hinter den Carriage Returns steht jeweils ein Linefeed.

Ebenfalls von GST in England stammt **1st Word.** Dieses Textprogramm ist eine Erweiterung des vorgenannten Editors auf einen Wordprocessor. Die Auswahl- und Editierfunktionen sind vermehrt worden. Man kann verschiedene Schriftarten wählen, so normal, leicht und fett, normal und kursiv, mit Unterstreichung und mit tiefgestellten oder hochgestellten Subscripts. Sonderzeichen mathematischer Art und aus anderen Sprachen können aus einer Tabelle an die laufende Schreibposition eingezogen werden und das natürlich auch in den genannten verschiedenen Schriftdarstellungen. Man kann die Formularbreite und Blocksatz wählen. Die Seitenlänge und die Zahl der Kopf- und Fußzeilen sind einstellbar. Seiten können zwangsweise umgebrochen werden, wobei man auch noch auf einen Grenzbereich Einfluß nehmen kann. Die Editierarbeit ist wesentlich auf die Bedienung der Maus gelegt, und man hat hier verbesserte Möglichkeiten z.B. mit der Einfassung eines Textbereiches mit einem "Gummiband", das von der Pfeilspitze der Maus gezogen wird. Das Umsetzen, Kopieren und Löschen von Textblöcken ist damit wesentlich bequemer. - Die Abspeicherung von Text kann entweder als reines ASCII-File erfolgen oder mit den eingeschlossenen Kontrollzeichen als Wordprozessor-File. Dem Programm sind Druckertreiber mitgegeben, mit denen man ggfs. selbst eine Anpassung vornehmen kann. Beim Händler hat der Autor hervorragende Ausdrucke auf angepaßten STAR-Druckern gesehen, die von den vielen Schriftarten und Editiermöglichkeiten Gebrauch machen.

Zum Schreiben von Programm-Quelltexten benötigt man nicht nur einen guten Editor, sondern auch gute Assembler und Compiler. Der Autor hat sich bisher nicht mit BASIC, LOGO oder PASCAL für diesen Rechner befaßt, sondern vor allem mit Assembler und etwas mit C. Danach ist wie folgt zu berichten: Dem Benutzer ist die Implementierung einer RAM-Disk sehr zu empfehlen, weil sie die zeitaufwendigen Diskettenzugriffe vermindert. Es sind verschiedene Programme für eine oder mehrere RAM-Disks mit auch unterschiedlichen Speichergrößen im Umlauf. Man sollte eine RAM-Disk wählen, die auf verschiedene Größen eingestellt werden kann. Beim Assemblieren und Linken braucht man schnell einmal 520 KB und mehr Platz für die schnellen Zugriffe.

Die zum Entwicklungspaket des Atari gehörenden Assembler und **Linker AS68.PRG und LINK68.PRG von Digital Research** sind auch beim Betrieb einer RAM-Disk weniger zu empfehlen. Beide haben außerordentlich langsame Durchläufe. Die

Zusammenarbeit des Assemblers mit der RAM-Disk klappte nicht ordentlich, sobald im Quelltext ein Fehler enthalten war.

**Der GST-Assembler und Linker** sind wiederum lobend zu erwähnen. Einerseits sind die Teilprogramme in einem Loseblatt-Ordner hervorragend dokumentiert, andererseits sind sie einfach zu bedienen und arbeiten sich in Windeseile durch ihren Job. Der Assembler kann einerseits ohne das Desktop betrieben werden. In diesem Fall erfolgt seine Steuerung durch textliche Eingaben. Im einfachsten Fall erfolgt die Abarbeitung in Assembler und Linker einfach mit der Eingabe des Dateinamens. - Ruft man jedoch alternativ das Programm GSTASM.PRG auf, so hat man die Einbindung in die Drop-Down-Menues zur Steuerung mit der Maus. In dieser Betriebsweise kann man für Berichtigungen auf das Editor-Programm zurückschalten und auch sagen: Assembliere und linke in einem Zug. - Der Assembler ist makrofähig und er kann aus Bibliotheken Auszüge für benötigte Module machen. Erzeugte Programme sind sofort lauffähig, ohne daß eine komplizierte Einbindung stattfinden muß.

□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□○□

Roland Löhr

# Atari 520ST empfängt seriell

Die neuen Atari-Rechner haben eine serielle Schnittstelle RS232 mit der ACIA 6850, die in das Betriebssystem eingebunden ist. Sie kann mit Funktionen des BIOS und des GEMDOS in der Ein- und Ausgabe bedient und auch vom Menue des Dektop her als VT52-Emulator betätigt werden. Und schließlich gibt es noch die Möglichkeit, die Druckeranpassung als MODEM vorzunehmen, so daß ein Dokument nach seiner Öffnung (2x auf dem Desktop anklicken) über den seriellen Kanal ausgedruckt werden kann. Der umgekehrte Weg ist allerdings vom Betriebssystem nicht vorgesehen, daß der Atari nämlich Daten seriell empfängt, sie im RAM zwischenspeichert und sie schließlich dauerhaft auf Diskette ablegt. Das nachstehende Programm ATARIRSF tut genau das. Die Daten werden dabei in einer Form abgespeichert, in der sie von den gängigen Editorprogrammen weiterbearbeitet werden können. Damit ist es z.B. möglich, Quelltexte von anderen Rechnern auf den Atari zu übertragen, um sie dort zu assemblieren oder zu kompilieren. Natürlich können auch Textdateien beliebiger anderer Art übertragen werden.

An anderer Stelle dieses Heftes wird dargestellt, daß die zahlreichen Routinen des GEMDOS mit auf dem Stack übergebenen Parametern, einer Funktionsnummer sowie mit dem Befehl TRAP #1 aufgerufen werden. Nach ihrer Abarbeitung ist der Stapelzeiger auf den Stand zu berichtigen, den er vor dem Aufruf hatte. - In diesem Programm nun werden nur einige wenige dieser Funktionen gebraucht, die der seriellen Eingabe, der Tastaturbedienung und der Ausgabe von Zeichen oder Strings auf den Bildschirm. Schließlich wird gezeigt, wie man unter GEMDOS eine Datei kreiert, sie beschreibt und wieder schließt. Einschließlich der interaktiven Führung des Benutzers bedarf es dazu keines sehr großen Programmieraufwandes, und der Ablauf bleibt übersichtlich. - Wir geben hier eine ausführliche Beschreibung der Programmierung, um den vielen Lesern, die sich inzwischen schon mit einem 68xxx-System befassen, Anregungen für die eigene Assembler-Programmierung zu vermitteln. Dabei wird man feststellen, daß Sprache und Syntax der Assembler für den 68000 viele Parallelen mit denen für den 6502 haben.

Der hier benutzte Assembler ist von GST in England. Er zeichnet sich durch schnelle Durchläufe beim Assemblieren und Linken aus. In seiner Syntax gibt es nur wenige Unterschiede zu den Assemblern von Motorola und Digital Research.

Ein Unterschied ist, daß die sonst üblichen drei Bereiche für .TEXT, .DATA und .BSS nicht unterschieden werden müssen. Stattdessen kann man einem Programmabschnitt einfach einen Namen geben, hier mit SECTION EINS. Eine weitere Eigenheit ist die sofortige Einbindung des Programmes in das Betriebssystem, wie immer das Label beim Programmbeginn heißt. Beim DRI-Assembler für den Atari sind wir gezwungen, den Punkt des Programmeintritts symbolisch mit dem Label _MAIN zu versehen.

Der Benutzer wird bei MAIN und an anderen Stellen interaktiv geführt. Die dazu benutzte Routine ist PRINTLINE. Sie verlangt nach den Konventionen des GEMDOS, daß das Register D0 zu Beginn die Speicheradresse des auszugebenden Strings enthält und daß dieser String mit einer hexadezimalen Null abgeschlossen ist. Im Prinzip läßt sich das beim 6502 mit der Beschickung eines Pointers auf den String vergleichen. - Bei den Labels M1 und Terminate wird der Befehl MOVE Multiple (MOVEM) zur Sicherung und Wiederherstellung mehrerer Register lt. Liste benutzt. Der Befehl LEA beim Label MAIN0 lädt die Anfangsadresse TXTBUF des geplanten Abspeicherungsbereiches in das Register A3. Dieses Register dient hernach in der MLOOP0 als Zeiger auf den aktuellen Punkt der nächsten Abspeicherung, die mit dem Befehl MOVE.B D0,(A3)+ erfolgt. Das bedeutet:

Speichere ein Byte aus dem Register D0 an die Stelle, auf die A3 weist und erhöhe die Adresse in A3 im Anschluß daran (Postinkrement).

Hinsichtlich des Befehles LEA (Lade Effektive Adresse) wird auf folgende Feinheiten hingewiesen: LEA LABEL,An transportiert die Adresse eines Labels in ein Adreßregister An. Gleichwertig wäre der Befehl MOVE.L #LABEL,An. Im Gegensatz dazu ist MOVE.L LABEL,An zu sehen. Dieser Befehl transportiert den Inhalt des Vektors bei LABEL nach An.

Bei MLOOP wird die Funktion AUXINSTAT (Status des Auxiliary Input, RS232-Schnittstelle) benutzt, ehe wenige Zeilen später die Funktion 3 mit Namen AUXIN (Holen des Zeichens von dort) benutzt wird. Diese Programmierung hat sich bei leerem Empfangspuffer als nützlich erwiesen. Es wurde darauf verzichtet, empfangene Zeichen auf dem Bildschirm auszugeben, weil der Atari bei 9600 Baud mit dem Bildschirm in Zeitnöte kommt, so daß ein Pufferüberlauf mit entsprechender Verstümmelung des Empfangenen entsteht. Empfangene Zeichen werden mit $7F maskiert und zu ASCII gemacht. Bei Bedarf kann man noch eine Code-Wandlung der empfangenen (Sonder-) Zeichen vornehmen. Linefeeds in der Eingabe werden übergangen, dafür wird an jedes CR ein Linefeed automatisch angeflochten, damit der Text in Textprogrammen darstellbar bleibt.

Das Ende einer Übertragung wird an CTRL-Z erkannt. Natürlich kann man auch jeden anderen Begrenzer wählen oder interaktiv abfragen und für den Vergleich speichern. Am Ende der Übertragung kommt man schließlich in eine kleine Kommandoebene bei MAINCMD. Es wird mit CONIN die Tastatur abgefragt. Nun kann man die empfangenen Zeilen mit 't' und 'd' (Top und Down) zur Anzeige bringen oder man kann weiteren Text empfangen ('a' = Append) und an das Gespeicherte anhängen. Die Taste CTRL-C bricht mit der GEMDOS-Funktion 00 das Programm ab. - Bis zu diesem Punkt dürfte die Programmierung kaum Geheimnisse enthalten haben, die man als 6502-Programmierer nicht aufdecken könnte.

Die Taste 'f' führt zur Abspeicherung des Textes auf Floppy. Dazu muß die Datei einen Namen erhalten, sie muß angelegt, beschreiben und wieder geschlossen werden. Das geschieht ab TXTSAVE. Die Namensgebung erfolgt mit dem Unterprogramm GETLINE von der Tastatur her. Man könnte dazu auch die Routine READLINE des GEMDOS nehmen, bei ihr muß man aber aufpassen, daß man nicht Tasten betätigt, die Editierfunktionen ausführen. GETLINE filtert die Zeichen daher auf die druckbaren aus.

Nun zur Namensgebung für Dateien und den in diesem Zusammenhang benutzten Begrif "Pfadname": Dateinamen bestehen aus bis zu 8 Buchstaben/Ziffern. Darauf kann ein Punkt mit weiteren drei Zeichen folgen, was dann Extension oder Namenserweiterung genannt wird. Das ist wie im CP/M und anderen darauf fußenden Betriebssystemen. Und der Pfad? Wenn wir auf dem Atari Ordner z.B MICROMAG.TXT anlegen, dann führt der Weg zu den Dateien über die Laufwerksangabe und den Namen des Ordners. Dabei dienen die rückwärtigen Schrägstriche (Backslashes) als Trenner. Eine Datei mit Namen HEFT46.INH im genannten Ordner, auf dem Laufwerk B, wird mit folgendem Pfadnamen angelegt oder mit der Suchfunktion SFIRST gefunden: B:\MICROMAG.TXT\HEFT46.INH. Ein solcher String ist in diesem Fall mit GETLINE am Label TXTS1 einzugeben. Wenn jedoch das aktuelle Laufwerk und das aktuelle Inhaltsverzeichnis benutzt werden sollen, dann reicht der eigentliche Dateiname HEFT46.INH.

Beim Label FCREATE wird eine Schreib-/Lesedatei mit dem gewählten Pfadnamen angelegt. Eine so geschaffene Datei muß nicht noch zusätzlich geöffnet werden. Wir können sie nach dem Schließen sogar auch schon auf dem Desktop sehen, auch wenn wir nichts in sie hineingeschreiben haben. - Wichtig ist nun der Begriff "handle": Beim Anlegen oder Öffnen einer Datei erhalten wir vom Betriebssystem eine logische Datei- oder Kanalnummer zugeteilt, die im Register D0.W abgeliefert wird. Wenn wir diese Zuordnungsnummer künftig verwenden, dann wird immer die entsprechende Datei angesprochen. Vergleichbar ist das z.B. mit den BASIC-Befehlen auf Commodore OPEN 7,8,9 und PRINT#7. Die 7 ist dabei die logische Dateinummer, die dem "handle" entspricht.

Auf FCREATE erfolgt eine Parameterübergabe, wo der Text beginnt und wieviele Bytes (im Stück) zu schreiben sind. Die Schreibfunktion FWRITE braucht dann nur noch den "handle". Schließlich erfolgt ein FCLOSE, wiederum mit Kanalnummer. An den wenigen Zeilen ab TXTSAVE bis zum FCLOSE sieht man, daß die Floppy-Funktionen im GEMDOS auch für den Assembler-Programmierer gut vorbereitet sind, so daß man sich nicht vor ihnen fürchten muß.

Es ist noch ein Blick auf die Unterprogramme zu werfen: GETLINE holt einen String zeichenweise von der Tastatur in einen Buffer, dessen Adresse im Register A5 mit dem Befehl LEA.L übergeben sein muß. Die Eingabe wird mit einem CR beendet. Dieses CR wird jedoch als hexadezimale Null abgespeichert, um den "Wünschen des Betriebssystems" zu entsprechen. Damit ist dieses Unterprogramm generell für die zeilenweise Eingabe geeignet, zumal es Steuerzeichen ausfiltert. Man kann es natürlich auch mit anderen Filtern versehen. Zur Logik von GETLINE ist nur noch zu bemerken, daß ein Backspace (Auslassungszeichen, $08) folgende Aktionen nach sich ziehen muß: Ausgabe dieses Zeichens als Cursor Left und Schreiben eines Blanks in die jetzt erreichte Vorspalte zum Löschen des dort noch angezeigten Zeichens. Das Schreiben des Blanks bewegt den Cursor jedoch wieder um 1 Stelle zuweit nach rechts, was durch ein weiteres Backspace ausgeglichen werden muß, um auf die aktuelle Schreibstelle zu positionieren.

Das Unterprogramm DRUCKBAR gibt Anlaß, noch einmal auf die Unterschiede bei Motorola-CPUs und beim 6502 bei Subtraktionen und Vergleichen hinzuweisen: Das Carry-Flag wird genau umgekehrt gesetzt: Beim "Borgen" dort wird das Carry-Flag gesetzt.

```
* ATARIRSF
* COPYRIGHT 1986 BY ROLAND LOEHR
* Atari 520ST empfaengt eine Textdatei seriell
* und speichert sie unter NAME.xxx auf Diskette ab
* Das Programm wird mit CTRL-C am Atari abgebrochen
* Ende-Zeichen der seriellen Datenuebertragung ist CTRL-Z
```

```
 SECTION  EINS                    Section des GST-Assemblers

gemdos    equ      1             TRAP-Nr. des gemdos

main      move.l   #textr,d0     Zeiger auf interaktiven Text setzen
m1        jsr      printline     Ausgabe per gemdos
          jsr      crlf          Zeilenvorschub
m2        movem.l  a3-a5,regsave Registerliste A3...A5 sichern
main0     lea      txtbuf,a3     Zeige auf Beginn des Textspeichers
          move.l   a3,txtpntr    laufenden Einsatzpunkt merken
m3        clr.b    (a3)          Dort eine Null als Begrenzer ablegen
*                                = indirekte Adressierung
* pruefen, ob ein Zeichen am RS232-Eingang vorliegt:
mloop     move.w   #18,-(sp)     Funktionsnummer: RS232-Status holen
          trap     #gemdos       Funktionsaufruf
          addq.l   #2,sp         Stackberichtigung
          beq.s    mloop         Kein Zeichen anhaengig
* Zeichen holen, auswerten und ablegen
mloop0    move.w   #3,-(sp)      Hole 1 Zeichen vom Kanal RS232 nach D0.W
          trap     #gemdos       mit AUXIN
          addq.l   #2,sp         Stack berichtigen
          and.w    #$007f,d0     Zeichen in D0.W zu ASCII maskieren
          cmp.b    #$1a,d0       Ende-Zeichen CTRL-Z?
          beq.s    txtend        ja
          cmp.b    #$0a,d0       Linefeed in der Eingabe
          beq.s    mloop         uebergehen
          move.b   d0,(a3)+      empfangenes Byte ins RAM, Pointer +1
          cmp.b    #$0d,d0       CR als Zeilenbegrenzer?
          bne.s    mloop         nein
          move.b   #$0a,(a3)+    ja, dann Linefeed im Editor anhaengen
          bra.s    mloop         Verzweige immer: Naechstes Zeichen

* Ende der Uebertragung an CTRL-Z erkannt
txtend    clr.b    (a3)          00 als Begrenzer im Speicher anhaengen
          move.l   a3,txtpntr    Adresse des Textendes merken

* Befehlsebene nach dem Empfangen einer Datei
maincmd   jsr      crlf          Neue Zeile
          move.l   #text2,d0     Befehlsebene anzeigen
          jsr      printline
* 1 Zeichen von der Tastatur lesen mit Echo auf den Bildschirm
maincmd1  move.w   #1,-(sp)      Funktionsnummer CONIN
          trap     #gemdos       Aufruf
          addq.l   #2,sp         Stack berichtigen
          move.w   d0,-(sp)      Taste auf dem Stack merken
          jsr      crlf          Neue Zeile
          move.w   (sp)+,d0      Tastaturzeichen zurueck
          cmp.b    #'a',d0       Append Text ans Ende?
          beq.s    append        ja
          cmp.b    #$03,d0       CTRL-C als Abbruch?
          beq.s    terminat      ja
          cmp.b    #'t',d0       Auf Topzeile setzen?
          beq.s    topzeile      ja
          cmp.b    #'d',d0       Down, in Folgezeile setzen?
          beq.s    down          ja
          cmp.b    #'f',d0       Auf Floppy speichern?
          beq      txtsave       ja
          bra.s    maincmd       Kein gueltiger Befehl
```

```
terminat  movem.l   regsave,a3-a5  Register zurueckholen
          clr.w     -(sp)          Funktions-Nummer 00
          trap      #gemdos        Prozess beenden

* Weiteren Text empfangen und anhaengen
append    move.l    txtpntr,a3     Alter Stand
          bra       mloop          Weitere Zeichen holen
* Auf erste Zeile positionieren
topzeile  lea.l     txtbuf,a3      Zeiger auf Textbeginn nach A3

* Auf Folgezeile positionieren
down      tst.b     (a3)           Sind wir am Ende des Editors?
          beq.s     maincmd        ja, neuen Befehl holen
* Eine Zeile, auf die A3 zeigt, auf den Bildschirm ausgeben
showline  move.b    (a3)+,d0       Laufendes Byte einer Zeile holen
          beq.s     maincmd        Abbruch, wenn 00-Begrenzer
          cmp.b     #$0d,d0        War letztes Byte CR?
          beq.s     showlcr        ja, return
          jsr       conout         Ausgabe mit GEMDOS
          bra.s     showline       Folgezeichen
showlcr   jsr       crlf           CRLF ausgeben
          bra.s     maincmd        Weiteren Befehl mit Prompt

* Lege ein Schreib-/Lesefile auf Floppy mit Namen an
txtsave   jsr       crlf           Neue Zeile
          move.l    #text1,d0      Zeiger auf interaktiven Text setzen
          jsr       printline      Ausgabe per gemdos
          lea.l     filename,a5    Benutze Namens-Puffer fuer die Eingabe
txts1     jsr       getline        1 Zeile mit Pfadnamen holen
* Lege neues File mit Namen und Attribut an
* Loesche ggfs. vorhandenes File gleichen Namens
fcreate   move.w    #0,-(sp)       Setze Attribut Lesen/Schreiben
          move.l    #filename,-(sp) Zeiger auf zu verwendenden Dateinamen
          move.w    #$3c,-(sp)     Funktionsnummer: Datei anlegen
          trap      #gemdos        Aufruf
          addq.l    #8,sp          Stack berichtigen
          move.w    d0,handle      Kanal-# merken
* Parameteruebergabe fuer FRWITE: Pufferbeginn und Menge zu schreibender Bytes
          move.l    #txtbuf,-(sp)  Zeige auf Pufferbeginn fuer fwrite
          move.l    txtpntr,d0     Ende des benutzten Speichers
          addq.l    #1,d0          txtpntr zeigte auf den Begrenzer
          sub.l     #txtbuf,d0     minus Beginn-Adresse
          move.l    d0,-(sp)       = Menge Bytes
          move.w    handle,-(sp)   Kanal-Nr.
          move.w    #$40,-(sp)     Funktions-# FWRITE
          trap      #gemdos        Aufruf
          add.l     #12,sp         Stack berichtigen
          tst.w     d0             Fehlerstatus erfassen

* FCLOSE, schliesse Datei mit handle-#
          move.w    handle,-(sp)   Kanal nennen
          move.w    #$3e,-(sp)     Funktionsnummer
          trap      #gemdos        Aufruf
          addq.l    #4,sp          Stack berichtigen
          tst.w     d0             Fehlerstatus, wenn negativ
          bra       maincmd        fertig, zur Befehlsebene

* Eine Eingabezeile als 00-terminierten String holen
```

```
getline   suba.l    a4,a4          Setze A4 zu 00
getline2  jsr       dircon         Zeichen holen, GEMDOS
          cmp.b     #$0d,d0        Zeilenabschluss?
          beq.s     getlincr       ja
          cmp.b     #$08,d0        Backspace?
          bne.s     getline3       nein
          cmpa.l    #0,a4          Sind wir in 1. Spalte?
          beq.s     getline2       ja, dann Backspace ignorieren
          jsr       conout         Zeichen ausgeben mit gemdos
          move.b    #' ',d0        Altes Zeichen mit Space
          jsr       conout         ueberschreiben
          move.b    #$08,d0        Cursor wieder auf alte
          jsr       conout         Schreibstelle setzen
          suba.l    #1,a5          Pointer berichtigen
          suba.l    #1,a4          Ebenso Zaehler
          bra.s     getline2
getline3  jsr       druckbar       Zeichen Filtern
          bmi.s     getline2       Nicht druckbar
          move.b    d0,(a5)+       Abspeichern in den Buffer
          jsr       conout         Und Ausgabe
          adda.l    #1,a4          Mitlaufender Zaehler +1
          cmpa.l    #80,a4         Zeile voll?
          bcc.s     getlincr       ja, 80 Zeichen, Zwangsabbruch
          bra.s     getline2
getlincr  clr.b     (a5)           00 als Begrenzer in den Buffer schreiben
getlinez  rts

* DRUCKBAR ist ein Filter fuer die Zeichen Space...Tilde ($7e)
druckbar  cmp.b     #$7f,d0        Rubout oder groesser?
          bcc.s     notdrb         ja, nicht druckbar
          cmp.b     #' ',d0        Space?
          bcs.s     notdrb         kleiner, nicht druckbar
          clr.w     filtflag       =00
          rts
notdrb    move.w    #$ffff,filtflag Flag negativ machen
          rts

*    dircon liest ein Zeichen von der Tastatur
*    ohne Echo-Ausgabe auf den Bildschirm,
dircon    move.w    #7,-(sp)             Funktionsnummer
dirconl   trap      #gemdos
          addq.l    #2,sp
          rts                            Nun Zeichen in D0 auswerten

*    conaut gibt ein in D0.W, Low Byte uebergebenes Zeichen auf
*    die Standardausgabe (Bildschirm) aus, das High Byte sollte 00 sein.
conout    move.w    d0,-(sp)             Zeichen auf den Stack
          move.w    #2,-(sp)             Funktions-Nr.
conoutl   trap      #gemdos              Aufruf
          addq.l    #4,sp
          rts
*    printline gibt einen mit 00 abgeschlossenen String aus,
*    auf dessen Beginn D0.L weist
printline move.l    d0,-(sp)             Adresse uebergeben
          move.w    #9,-(sp)             Funktionsnummer
          trap      #gemdos
          addq.l    #6,sp                Stapelzeiger berichtigen
          rts
```

```
*    crlf gibt ein CR und LF auf die Standard-Ausgabe (Bildschirm)
crlf      move.w    #$0d,d0          Das CR
          bsr.s     conout
          move.w    #$0a,d0          Das LF
          bsr.s     conout
          rts

* Arbeitsspeicher
regsave   ds.l      3                Register-Ablage zur Sicherung
txtpntr   ds.l      1                Zeiger auf Einsatzpunkt Editor
handle    ds.w      1                # des Kanals
filtflag  ds.w      1                Flag-Variable
filename  ds.b      80               Filename-Arbeitsspeicher
textl     dc.b      'Eingabe Filename/Pathname:   ',0
textr     dc.b      'RS232-Programm',0
text2     dc.b      'Befehl eingeben: a t d f CTRL-C  ',0

txtbuf    ds.b      1                Beginn des Abspeicherungsbereiches
 END
```

Horst Brettin, 1000 Berlin 44

# Kommando-Files

## auf dem CBM 3032

Viele Rechner bieten die Möglichkeit, Kommandofiles auszuführen. Außerdem vermißt man beim CBM bzw. C-64 das Laden von ASCII-Files. Beides wird mit dem Programm EXECUTE zumindest angenähert errreicht.

Der Aufruf erfolgt mit SYS 634,Name. Danach wird das File wie eine Eingabe von der Tastatur behandelt. Das Programm arbeitet im Interrupt. Dabei wird statt der Tastatur das Kommandofile gelesen und seine Zeichen im Tastaturpuffer abgelegt. Das Betriebssystem verarbeitet dann alles, als ob es von Hand eingegeben wäre. - An Ende des Files, wenn ein EOF (CTRL-C) gefunden wird oder nach dem Betätigen der Stop-Taste, geht die Kontrolle wieder an die Tastatur.

Da der vorgefundene Interruptvektor zwischengespeichert und am Ende wieder hergestellt wird, verträgt sich EXECUTE recht gut mit anderen Tools, die in den Interrupt eingesschleift sind, z.B. mit dem SM-Kit. - Kommandofiles lassen sich mit mit jedem Editor erstellen, der ASCII-Files ausgibt. Dazu noch ein Tip: LISTen Sie ein BASIC-Programm auf Floppy, bearbeiten Sie es im Editor und 'laden' es wieder mit EXECUTE. So lassen sich auch fertige Module in ein Programm übernehmen! Auch das Einlesen von BASIC-Programmen anderer Rechner ist auf diese Weise möglich, wenn sie als ASCII-Files vorliegen.

Einschränkungen: EXECUTE verwendet die Sekundäradresse 14. Sie darf natürlich nicht anderweitig benutzt werden. Das Kommandofile darf auch nicht geschlossen oder durch sich selbst gelöscht werden. - Wird der IEC-Bus anderweitig verwendet, so wartet EXECUTE, bis er wieder frei ist (z.B. bei LOAD oder SAVE) und arbeitet dann erst weiter, um die Datenübertragung nicht zu stören. Daraus folgt aber auch, daß daß Kommandofile nach einem CMD-Befehl nicht weiter abgearbeitet wird.

Auf eine Falle im CBM 3001 sei besonders hingewiesen: Aus unerfindlichen Gründen prüft das Betriebssystem bei LOAD, ob der Interruptvektor auf die

Tastaturabfrage zeigt. Sonst wird gewartet, bis die Adresse stimmt (wer oder was sollte sie an dieser Stelle ändern?) oder bis die Stop-Taste betätigt wird. Dieses wird von meinem Programm abgefangen. Und im Falle daß wird durch Manipulieren des Stacks weitergearbeitet (Zeile 74...83). - EXECUTE läßt sich an alle CBM's anpassen.

```
0003 0000           ;         EXECUTE  CBM 3032 31.12.84
0004 0000           ;         EXECUTE+          09.01.86
0005 0000           ;         EXEC 2+           10.01.86
0006 0000
0007 0000           ;         (C) BY HORST BRETTIN
0008 0000
0009 0000
0010 0000           EOT       =$03
0011 0000           SECADR    =$6E SEC.ADR 14
0012 0000
0013 0000           IRQV      =$90 IRQ-VEKTOR
0014 0000           ST        =$96 I/O-STATUS
0015 0000           KYINX     =$9E KEYBUFFERINDEX
0016 0000           KEYCOD    =$A6
0017 0000           NAMLEN    =$D1
0018 0000           FILE      =$D2
0019 0000           FILSEC    =$D3
0020 0000           DEVICE    =$D4
0021 0000           FILNAM    =$DA
0022 0000
0023 0000           KYBUF     =$26F
0024 0000
0025 0000           CHKCOM    =$CDF8
0026 0000           IRQEND    =$E6E4
0027 0000           GETNAM    =$F4FD
0028 0000           OPENI     =$F466
0029 0000           CLOSEI    =$F6F0
0030 0000           TALK      =$F0B6
0031 0000           TLKSEC    =$F164
0032 0000           UNTLK     =$F17F
0033 0000           IECIN     =$F18C
0034 0000
0035 0000                     ;FALLE IM BETIEBSSYSTEM DES CBM
0036 0000           TRPBGN    =$F8E6   TRAP-BEGIN
0037 0000           TRPEND    =$F8EF   TRAP-ENDE
0038 0000
0039 0000           PIAB      =$E812
0040 0000           PIA       =$E840
0041 0000
0042 0000                     *=$27A
0043 027A
0044 027A 20 F8 CD  EXEC      JSR CHKCOM        ;TEST AUF
0045 027D 20 FD F4            JSR GETNAM
0046 0280 A9 08               LDA#8
0047 0282 85 D4               STA DEVICE
0048 0284 A9 6E               LDA#SECADR
0049 0286 85 D3               STA FILSEC
0050 0288 A2 00               LDX#0
0051 028A 86 96               STX ST
0052 028C CA                  DEX               ;TASTENCODE
0053 028D 86 A6               STX KEYCOD        ;LOESCHEN
0054 028F 20 66 F4            JSR OPENI
0055 0292 A5 90               LDA IRQV
```

```
0056 0294 A6 91          LDX IRQV+1
0057 0296 8D 25 03       STA IRQH
0058 0299 8E 26 03       STX IRQH+1
0059 029C 78             SEI
0060 029D A9 A7          LDA#<IRQP
0061 029F A2 02          LDX#>IRQP
0062 02A1 85 90          STA IRQV
0063 02A3 86 91          STX IRQV+1
0064 02A5 58             CLI
0065 02A6 60             RTS
0067 02A7 A5 D4    IRQP  LDA DEVICE       ;RETTEN
0068 02A9 48             PHA
0069 02AA A5 96          LDA ST            ;I/O-STATUS RETTEN
0070 02AC 48             PHA
0071 02AD AD 12 E8       LDA PIAB
0072 02B0 29 10          AND#$10           ;STOP-TASTE ?
0073 02B2 F0 47          BEQ STOP
0074 02B4 BD 06 01       LDA $106,X        ;IN DER FALLE?
0075 02B7 C9 F8          CMP#>TRPBGN
0076 02B9 D0 10          BNE NOTRAP
0077 02BB BD 05 01       LDA $105,X
0078 02BE C9 E6          CMP#<TRPBGN
0079 02C0 90 09          BCC NOTRAP
0080 02C2 C9 EF          CMP#<TRPEND
0081 02C4 B0 05          BCS NOTRAP
0082 02C6 A9 EF          LDA#<TRPEND       ;RETURN-ADDR
0083 02C8 9D 05 01       STA $105,X        ;AUF TRAP-END
0084 02CB AD 40 E8 NOTRAP LDA PIA
0085 02CE 29 C1          AND#$C1           ;MASKE F. DAV,NRFD U. NDAC
0086 02D0 49 C1          EOR#$C1           ;IEC-BUS FREI?
0087 02D2 D0 42          BNE END
0088 02D4 85 96          STA ST
0089 02D6 A9 08          LDA#8
0090 02D8 85 D4          STA DEVICE
0091 02DA 20 B6 F0       JSR TALK
0092 02DD A9 6E          LDA#SECADR
0093 02DF 20 64 F1       JSR TLKSEC
0094 02E2 A4 9E    NXTC  LDY KYINX
0095 02E4 C0 0A          CPY#10            ;KEYBUFFER VOLL?
0096 02E6 B0 37          BCS FULL
0097 02E8 20 8C F1       JSR IECIN
0098 02EB C9 03          CMP#EOT
0099 02ED F0 09          BEQ TEND
0100 02EF 99 6F 02       STA KYBUF,Y
0101 02F2 E6 9E          INC KYINX
0102 02F4 A5 96          LDA ST
0103 02F6 F0 EA          BEQ NXTC
0104 02F8
0105 02F8 20 7F F1 TEND  JSR UNTLK
0106 02FB A5 D3    STOP  LDA FILSEC
0107 02FD 48             PHA
0108 02FE A9 08          LDA#8
0109 0300 85 D4          STA DEVICE
0110 0302 A9 6E          LDA#SECADR
0111 0304 85 D3          STA FILSEC
0112 0306 20 F0 F6       JSR CLOSEI
0113 0309 68             PLA
0114 030A 85 D3          STA FILSEC
```

```
0115 030C AD 25 03          LDA IRQH
0116 030F AE 26 03          LDX IRQH+1
0117 0312 85 90             STA IRQV
0118 0314 86 91             STX IRQV+1
0119 0316 68          END   PLA
0120 0317 85 96             STA ST
0121 0319 68                PLA
0122 031A 85 D4             STA DEVICE
0123 031C 4C E4 E6          JMP IRQEND
0124 031F
0125 031F 20 7F F1 FULL     JSR UNTLK
0126 0322 4C 16 03          JMP END
0127 0325
0128 0325
0129 0325           IRQH    *=*+2               ;HILFSSPEICHER F. IRQ-VEKTOR
```

□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O□O

Roland Löhr

# Emulation

**einer anderen CPU**

Dieser Artikel enthält lediglich eine Studie und Überlegungen, kein zusammenhängendes Programm: Den Autor interessierte die Frage, wie man die Programme, die im Maschinencode einer 'fremden' CPU geschrieben sind, auf einer gegebenen CPU nachfahren kann, die einen anderen Befehlssatz hat. Diese Frage hat durchaus aktuelle Bezüge. Einerseits bemüht man sich, die Betriebsprogramme eines anderen Systems nachzuahmen, so daß sich eine gleiche Benutzeroberfläche ergibt. Man denke hier an die Ähnlichkeiten bei z.B. CP/M, CP/M-68K und MS-DOS. Andererseits ist man dabei, mit den leistungsfähigen 16-Bit-Rechnern den Maschinencode z.B. des IBM-PC so nachzufahren, daß vorhandene Software auch in einer anderen Umgebung zur Ausführung gebracht werden kann. Gelegentlich geht man auch einen dritten Weg, indem man dem Computer eine zweite oder dritte CPU unter die Haube steckt, die dann jeweils 'ihren' Maschinencode abarbeiten kann.

Die Emulation einer anderen CPU setzt voraus, daß man in Software ein Abbild dieser anderen CPU schafft. Einerseits braucht man einen 'Sequenzer', eine Einrichtung, die sich von Befehl zu Befehl durch den Code hindurcharbeitet und die die den Befehlen entsprechenden Aktionen mit gleicher Wirkung zur Ausführung bringt. Diese Einrichtung könnte man auch 'Interpreter' nennen. Dementsprechend haben wir in der nachfolgenden Liste auch Label, die INTPR und ähnlich heißen. In dieser Studie ist der Versuch gemacht worden, daß die CPU 6502 ihren eigenen Maschinencode interpretierend zur Ausführung bringt. Ihre Register sind als AKKU, IX, YPS usw. als Zellen in der Zero Page abgebildet. Die Abarbeitung von Befehlen wird von drei Tabellen gesteuert: CODETAB enthält eine Verschlüsselung der Befehlslänge und der mit einem Opcode verbundenen Adressierungsart. Ein Opcode indiziert über das Register Y direkt in CODETAB. Wird dort eine 00 gefunden, so handelt es sich um einen verbotenen Opcode, und der Interpreter wird unsanft per BRK-Befehl zum 'Absturz' gebracht.

Die Tabellen LWORDS und HWORDS enthalten für die Befehle mit Opcode hex 0-7F bzw. 80-FF Sprungziele auf Routinen, die die fremden Befehle in Software abbilden. Solche Routinen sind z.B. —ASLA oder —CLC und —LDA. Der fremde Opcode vektoriert also unmittelbar auf solche Routinen.

Es wurde nicht der Versuch gemacht, alle Befehle in Software abzubilden, denn warum sollte die 6502 sich selbst emulieren? Wenn schon, dann sollte sie z.B. eine Z80 oder 6805 'nachfahren'. Aber auch das ist wegen der geringen zu erwartenden Ausführungsgeschwindigkeit noch recht witzlos. Besser ist es, wenn man die Grundidee auf einen 68000 umsetzt. Hier kann man die Register einer 8 Bit CPU ohne Schwierigkeiten in den Daten- und Adreßregistern der 68000 abbilden und indizierte Befehle dann auch mit den entsprechenden Registern indizieren. Das

in der vorliegenden Liste mit ADRESS bezeichnete 'Register' in der Zero Page würde man in ein Adreßregister der 68000 legen und die Operandenansprache per (Adreßregister indirekt) über dieses Register steuern. Auch könnte man den Stackpointer der anderen CPU viel bequemer in ein Adreßregister verlegen. - In der vorliegenden Liste wurde noch kein Stackpointer eingerichtet. Wenn man den Weg hier konsequent weitergehen wollte, so sollte man z.B. das X-Register der CPU 6502 als künstlichen Stackpointer benutzen.

An dieser Studie mag interessant sein, daß einige Befehle mit geringem Aufwand nachgebildet werden können, so z.B. Inherent-Befehle. Recht einfach sieht es auch noch bei den Sprüngen aus, für die das Register PC umgeladen werden muß. Etwas komplizierter ist es bei den bedingten Verzweigungen. Hier muß mit Hilfe des nachgeführten STATUS zunächst geprüft werden, ob die Verzweigungsbedingung zutrifft. Wenn ja, dann ist zu prüfen, ob nach vorne oder nach rückwärts zu verzweigen ist. Der PC ist entsprechend umzuladen.

Viele Spielarten gibt es dann bei den arithmetischen und logischen Befehlen. Ausgeführt wurden nur Sequenzen für LDA. Es gibt hier die Adressierungen Direktoperand und solche mit und ohne Nach-Indizierung mit X oder Y, Zero Page und Absolut. Das läßt sich noch recht einfach mit dem Unterprogramm MIT—XY nachbilden. Größere Umladevorgänge sind bei den indirekten Adresssierungen nowendig: Bei (indirekt,X) muß man zunächst bestimmen, welche Zellen die indirekte Adresse enthalten. Deren Inhalt lädt man nach ADRESS um. Etwas einfacher ist es bei (Pointer) ,Y. Der Pointer steht bereits in ADRESS, er ist um den Inhalt des Registers YPS zu erhöhen.

Im Laufe der Zeit ist es sicher interessant, von den Lesern über derartige Nachbildungen einer CPU z.B. auf dem 68000 zu erfahren.

```
0003                                ;EINE STUDIE, DIE CPU 6502 DURCH SICH SELBST ZU E
MULIEREN
0004                                ;VERKUERZTE LISTE
0005                                ;IM RAM ABGEBILDETE REGISTER
0006 000000                         *=$10               ;ANSIEDLUNG IN DER ZERO PAGE
0007 000010          AKKU           *=*+1
0008 000011          IX             *=*+1
0009 000012          YPS            *=*+1
0010 000013          STATUS         *=*+1
0011 000014          SP             *=*+2               ;STACKPOINTER
0012 000016          PC             *=*+2               ;VIRTUELLER PROGRAMMZAEHLER
0013 000018          OPCODE         *=*+1               ;VORGEFUNDENER OPCODE
0014 000019          ADRESS         *=*+2               ;ADRESSREGISTER DES PROZESSOR
                                        S LOW/HIGH
0015 00001B          ADMODE         *=*+1               ;ADRESSIERUNGSART PER OPCODE
0016 00001C          LEN            *=*+1               ;LAENGE DES 6502-BEFEHLES
0017 00001D          OFFSET         *=*+1               ;OFFSET BEI BRANCHES
0018 00001E          EXECUTE        *=*+2               ;VERTEILER INDIREKTE SPRUENGE
0019
0020 000020                         *=$1000             ;INTERPRETER-BEGINN
0021 001000 08       INTPR          PHP                 ;BRINGE STATUS INS RAM-REGIST
                                        ER
0022 001001 68                      PLA
0023 001002 8513                    STA STATUS
0024 001004 A51C     INTPR_1        LDA LEN             ;BEFEHLSLAENGE, LETZTER BEFEH
                                        L
0025 001006 18                      CLC
0026 001007 6516                    ADC PC              ;ZUORDNUNGSZAEHLER WEITERSETZ
                                        EN, LOW
0027 001009 8516                    STA PC
0028 00100B 9002                    BCC INTPR_2
0029 00100D E617                    INC PC+1            ;HIGH
0030 00100F A000     INTPR_2        LDY #0
0031 001011 B116                    LDA (PC),Y          ;HOLE OPCODE
0032 001013 A8                      TAY                 ;ALS ZEIGER IN CODE-TABELLE
0033 001014 8518                    STA OPCODE
0034 001016 B97113                  LDA CODETAB,Y       ;SEHEN OB ERLAUBT
0035 001019 F048                    BEQ FEHLER          ;NEIN
0036 00101B 29F8                    AND #%11111000      ;LAENGE ABMASKIEREN
0037 00101D 851B                    STA ADMODE          ;FESTHALTEN
0038 00101F B97113                  LDA CODETAB,Y
0039 001022 2903                    AND #%00000011      ;LAENGE BESTIMMEN
```

```
0039 001022 2903            AND #%00000011      ;LAENGE BESTIMMEN
0040 001024 851C            STA LEN             ;BEFEHLSLAENGE
0041 001026 A001    INTPR_3 LDY #1
0042 001028 A900            LDA #0
0043 00102A 851A            STA ADRESS+1        ;AUF VORRAT/VERDACHT EINSETZE
                                                N
0044 00102C B116    INTPR_3A LDA (PC),Y         ;HOLE BYTE(S) HINTER OPCODE
0045 00102E 8519            STA ADRESS          ;LOW-TEIL
0046 001030 297F            AND #$7F
0047 001032 851D            STA OFFSET          ;AUF VORRAT ABLEGEN
0048 001034 C8              INY
0049 001035 B116            LDA (PC),Y          ;HIGH
0050 001037 851A            STA ADRESS+1
0051 001039 A518    INTPR_4 LDA OPCODE
0052 00103B 0A              ASL A               A ;X2
0053 00103C AB              TAY                 ;ADRESSER
0054 00103D B00C            BCS INTPR_5         ;OBERE HAELFTE
0055 00103F B9A111          LDA LWORDS,Y
0056 001042 851E            STA EXECUTE         ;SPRUNGVERTEILUNG
0057 001044 B9A211          LDA LWORDS+1,Y
0058 001047 851F            STA EXECUTE+1
0059 001049 900A            BCC INTPR_8         ;ALWAYS
0060 00104B B9A112  INTPR_5 LDA HWORDS,Y
0061 00104E 851E            STA EXECUTE         ;SPRUNGVERTEILUNG
0062 001050 B9A212          LDA HWORDS+1,Y
0063 001053 851F            STA EXECUTE+1
0064 001055 A51B    INTPR_8 LDA ADMODE          ;MERKER IN X SETZEN
0065 001057 2920            AND #%00100000      ;F. INDIREKTE ADRESSIERUNGEN
0066 001059 AA              TAX
0067 00105A A000            LDY #00
0068 00105C A51B            LDA ADMODE          ;WELCHE ADRESSIERUNG?
0069 00105E C910            CMP #$10            ;DIREKTOPERAND?
0070 001060 6C1E00          JMP (EXECUTE)
0071
0072 001063 00      FEHLER  BRK                 ;ABBRUCH DER INTERPRETATION
0073
0074 001064         _ADC
0075 001064         _AND
0076 001064         _ASL
0077 001064 0A      _ASLA   ASL A
0078 001065 4C0010          JMP INTPR
0079 001068 AD7114  _BCC    LDA BITO
0080 00106B D000            BNE BRAN_1          ;ALWAYS
0081 00106D 2413    BRAN_1  BIT STATUS          ;NACHGEFUEHRTER STATUS
0082 00106F D093            BNE INTPR_1         ;KEINE VERZWEIGUNG!
0083 001071 2419    BRAN_1A BIT ADRESS          ;VORWAERTS/RUECKWAERTS?
0084 001073 3012            BMI BRAN_BACK       ;RUECKWAERTS
0085 001075 18              CLC                 ;SPRUNGZIEL?
0086 001076 A51D            LDA OFFSET
0087 001078 6516            ADC PC
0088 00107A 6902            ADC #2              ;BASIS IST DER FOLGEBEFEHL
0089 00107C 8516            STA PC              ;NEUER PROGRAMMZAEHLER
0090 00107E A517            LDA PC+1
0091 001080 6900            ADC #0
0092 001082 8517    BRAN_2  STA PC+1
0093 001084 4C0F10          JMP INTPR_2         ;ZUM FOLGEBEFEHL
0094 001087 18      BRAN_BACK CLC               ;BASIS FOLGEBEFEHL
0095 001088 A516            LDA PC
0096 00108A 6902            ADC #2
0097 00108C 9000            BCC BRAN_B1
0098 00108E         IND                         PC+1
0099 00108E 38      BRAN_B1 SEC
0100 00108F E51D            SBC OFFSET
0101 001091 8516            STA PC
0102 001093 A517            LDA PC+1
0103 001095 E900            SBC #0
0104 001097 4CB210          JMP BRAN_2
0105
0106 00109A AD7114  _BCS    LDA BITO
0107 00109D D008            BNE BRAN_MI1
0108 00109F AD7214  _BEQ    LDA BIT1
0109 0010A2 D003            BNE BRAN_MI1        ;ALWAYS
0110
0111 0010A4         _BIT
```

```
0112 0010A4 AD7814    _BMI       LDA BIT7              ;BEFEHL BMI
0113 0010A7 2413      BRAN_MI1   BIT STATUS
0114 0010A9 D0C6                 BNE BRAN_1A           ;ZIEL BERECHNEN
0115 0010AB 4C0410               JMP INTPR_1           ;KEINE VERZWEIGUNG
0116
0117 0010AE AD7214    _BNE       LDA BIT1
0118 0010B1 D0BA                 BNE BRAN_1            ;ALWAYS
0119 0010B3 AD7814    _BPL       LDA BIT7
0120 0010B6 D0B5                 BNE BRAN_1
0121 0010B8 00        _BRK       BRK                   ;SOFTWARE-INTERRUPT
0122 0010B9 AD7714    _BVC       LDA BIT6              ;ENTSCHEIDEN F. BVC
0123 0010BC D0AF                 BNE BRAN_1            ;ALWAYS
0124 0010BE AD7714    _BVS       LDA BIT6              ;PRUEFUNG FUER BVS
0125 0010C1 D0E4                 BNE BRAN_MI1          ;ALWAYS
0126
0127 0010C3 A9FE      _CLC       LDA #%11111110
0128 0010C5 2513      _CLC1      AND STATUS
0129 0010C7 8513                 STA STATUS
0130 0010C9 4C0410               JMP INTPR_1
0131 0010CC A9F7      _CLD       LDA #%11110111
0132 0010CE D0F5                 BNE _CLC1             ;ALWAYS
0133 0010D0 A9FB      _CLI       LDA #%11111011
0134 0010D2 D0F1                 BNE _CLC1
0135 0010D4 A9BF      _CLV       LDA #%10111111
0136 0010D6 D0ED                 BNE _CLC1
0137 0010D8           _CMP
0138 0010D8           _CPX
0139 0010D8           _CPY
0140 0010D8 B119      _DEC       LDA (ADRESS),Y
0141 0010DA 38                   SEC
0142 0010DB E901                 SBC #1
0143 0010DD 9119                 STA (ADRESS),Y
0144 0010DF 4C0010               JMP INTPR
0145 0010E2 C611      _DEX       DEC IX
0146 0010E4 4C0010               JMP INTPR
0147 0010E7 C612      _DEY       DEC YPS
0148 0010E9 4C0010               JMP INTPR
0149 0010EC           _EOR
0150 0010EC B119      _INC       LDA (ADRESS),Y
0151 0010EE 18                   CLC
0152 0010EF 6901                 ADC #1
0153 0010F1 9119                 STA (ADRESS),Y
0154 0010F3 4C0010               JMP INTPR
0155 0010F6 E611      _INX       INC IX
0156 0010F8 4C0010               JMP INTPR
0157 0010FB E612      _INY       INC YPS
0158 0010FD 4C0010               JMP INTPR
0159 001100 A519      _JMP       LDA ADRESS            ;EMULIERE JMP ABS
0160 001102 8516                 STA PC                ;UMKOPIEREN
0161 001104 A51A                 LDA ADRESS+1
0162 001106 8517                 STA PC+1
0163 00110B 4C0F10               JMP INTPR_2           ;FOLGEBEFEHL INTERPRETIEREN
0164 00110B A000      _JMPIND    LDY #00               ;EMULIERE JMP (..)
0165 00110D B119                 LDA (ADRESS),Y        ;HOLE SPRUNGZIEL
0166 00110F 8516                 STA PC
0167 001111 C8                   INY
0168 001112 B119                 LDA (ADRESS),Y        ;HIGH BYTE
0169 001114 8517                 STA PC+1
0170 001116 4C0F10               JMP INTPR_2           ;FOLGEBEFEHL INTERPRETIEREN
0171 001119           _JSR
0172 001119 D005      _LDA       BNE _LDA2             ;KEIN DIREKTOPERAND

0173 00111B 8510      _LDA1      STA AKKU
0174 00111D 4C0010               JMP INTPR             ;FOLGEBEFEHL
0175 001120 8A        _LDA2      TXA                   ;INDIREKTE ADRESSIERUNG?
0176 001121 D008                 BNE _LDA3             ;JA
0177 001123 208911               JSR MIT_XY            ;GGFS. PLUS INDEX
0178 001126 B119                 LDA (ADRESS),Y
0179 001128 4C1B11               JMP _LDA1
0180 00112B 241B      _LDA3      BIT ADMODE
0181 00112D 1003                 BPL _LDA4             ;KEINE VOR-INDIZIERUNG
```

```
0182 00112F 208911            JSR MIT_XY          ;MIT IX VOR-INDIZIEREN
0183 001132 B119      _LDA4   LDA (ADRESS),Y
0184 001134 48                PHA
0185 001135 C8                INY
0186 001136 B119              LDA (ADRESS),Y      ;DIE INDIREKTE ADRESSE HOLEN
0187 001138 851A              STA ADRESS+1
0188 00113A 68                PLA
0189 00113B 8519              STA ADRESS
0190 00113D 88                DEY                 ;Y=0
0191 00113E 241B              BIT ADMODE
0192 001140 3003              BMI _LDA5
0193 001142 208911            JSR MIT_XY
0194 001145 B119      _LDA5   LDA (ADRESS),Y      ;NUN AUS EFFEKTIVER ADRESSE H
                                  OLEN
0195 001147 8510              STA AKKU            ;UND ABLEGEN
0196 001149 4C0010            JMP INTPR
0197
0198 00114C           _LDX
0199 00114C           _LDY
0200 00114C           _LSR
0201 00114C 4A        _LSRA   LSR A
0202 00114D 4C0010            JMP INTPR
0203 001150 4C0410    _NOP    JMP INTPR_1
0204 001153           _ORA
0205 001153           _PHA
0206 001153           _PHP
0207 001153           _PLA
0208 001153           _PLP
0209 001153           _ROL
0210 001153 2A        _ROLA   ROL A
0211 001154 4C0010            JMP INTPR
0212 001157           _ROR
0213 001157 6A        _RORA   ROR A
0214 001158 4C0010            JMP INTPR
0215 00115B           _RTI
0216 00115B           _RTS
0217 00115B           _SBC
0218 00115B A900      _SEC    LDA #%00000001
0219 00115D 0513      _SEC1   ORA STATUS
0220 00115F 8513              STA STATUS
0221 001161 4C0410            JMP INTPR_1
0222 001164 A900      _SED    LDA #%00001000
0223 001166 D0F5              BNE _SEC1           ;ALWAYS
0224 001168 A904      _SEI    LDA #%00000100
0225 00116A D0F1              BNE _SEC1           ;ALWAYS
0226 00116C           _STA
0227 00116C           _STX
0228 00116C           _STY
0229 00116C A510      _TAX    LDA AKKU
0230 00116E 8511              STA IX
0231 001170 4C0010            JMP INTPR
0232 001173 A510      _TAY    LDA AKKU
0233 001175 8512              STA YPS
0234 001177 4C0010            JMP INTPR
0235 00117A A512      _TYA    LDA YPS
0236 00117C 8510              STA AKKU
0237 00117E 4C0010            JMP INTPR
0238 001181           _TSX
0239 001181 A511      _TXA    LDA IX
0240 001183 8510              STA AKKU
0241 001185 4C0010            JMP INTPR
0242 001188           _TXS                        ;OHNE STATUSBEEINFLUSSUNG
0243 001188 00        _NIX    BRK                 ;UNZULAESSIGER OPCODE
0244
0245 001189 241B      MIT_XY  BIT ADMODE          ;IST IX ODER YPS ZU ADDIEREN?
0246 00118B 100C              BPL MIT_XY2
0247 00118D 18                CLC
0248 00118E A511              LDA IX              ;ADDIERE IX
0249 001190 6519      MIT_XY0 ADC ADRESS
0250 001192 8519              STA ADRESS
0251 001194 9002              BCC MIT_XY1
0252 001196 E61A              INC ADRESS+1
```

```
0253 001198 60          MIT_XY1   RTS
0254 001199 50FD        MIT_XY2   BVC MIT_XY1          ;NICHTS INDIZIERT
0255 00119B 18          MIT_XY3   CLC
0256 00119C A512                  LDA YPS              ;ADDIERE Y-REGISTER
0257 00119E 4C9011                JMP MIT_XYO
0258
0259 0011A1 88105311 LWORDS       .WOR _BRK,_ORA,_NIX,_NIX,_NIX,_ORA,_ASL,_NIX
0259 0011A5 88118811
0259 0011A9 88115311
0259 0011AD 64108811
0277 0012A1                       .OPT LIS
0278 0012A1             HWORDS                         _NIX,_STA,_NIX,_NIX,_STY,_STA
                                    ,_STX,_NIX
0293 001371                       .OPT LIS
0294                              ;CODETAB SETZT OPCODES IN ADRESSIERUNGSARTEN UM
0295                              ;WENN EIN EINTRAG 00 IST, DANN IST DER OPCODE VER
BOTEN
0296                              ;BIT 0 UND 1 BESTIMMEN BEFEHLSLAENGE
0297                              ;BIT 7 BESTIMMT, OB MIT REGISTER IX
0298                              ;BIT 6 BESTIMMT, OB MIT REGISTER YPS
0299                              ;BIT 5 BESTIMMT, OB INDIREKT (...)
0300                              ;BIT 4 BESTIMMT, OB DIREKTOPERAND #
0301                              ;BIT 3 BESTIMMT, OB INHERENT AKKU
0302                              ;BIT 2 BESTIMMT, OB RELATIVE VERZWEIGUNG
0303 001371             CODETAB                        ;ZEILEN IN DER ANORDNUNG NACH
                                    MSD, 1. HALBBYTE
0304 001371 01A20000 CT0_         .DBY $01A2,$0000,$0002,$0200,$0112,$0900,$0003,$
                                    0300
0304 001375 00020200
0304 001379 01120900
0304 00137D 00030300
0322 001471                       .OPT LIS
0323 001471 01          BIT0      .BYT 1               ;ENTSPRICHT CARRY FLAG
0324 001472 02          BIT1      .BYT 2               ;ENTSPR. Z-FLAG IM STATUS
0325 001473 04          BIT2      .BYT 4               ;DITO INTERRUPT-FLAG
0326 001474 08          BIT3      .BYT 8               ;DITO DECIMAL FLAG
0327 001475 10          BIT4      .BYT 16              ;DITO: BREAK-FLAG
0328 001476 20          BIT5      .BYT 32              ;DITO, ENTSPRICHT NO-FLAG
0329 001477 40          BIT6      .BYT 64              ;DITO, OVERFLOW-FLAG3
0330 001478 80          BIT7      .BYT 128             ;DITO, NEGATIVE FLAG
0331
0332 001479                       .END
FEHLERZAHL INSGESAMT: 0000
```

Roland Löhr

# Banking im PC-128

**Wichtige Merkmale des Computers**

Der neue Commodore-Rechner wurde bereits in Heft 45 kurz vorgestellt. Seine wichtigsten Merkmale sind: Die CPU 8502 hat den Befehlssatz des 6502. Sie ist in Adresse 0 mit einem Datenrichtungsregister versehen und in Adresse 1 mit einem bidirektionalen Datenport mit 7 Pins. Dieses ist eine Änderung gegenüber der im C-64 benutzten CPU 6510, deren Port nur 6 Pins hat. Pin P6 fühlt die Caps Lock-Taste ab. Die übrigen Pins steuern wie beim C-64 die Datasette und die ROM/RAM-Umschaltung. - Wir sehen einmal von der möglichen Betriebsweise des Rechners als C-64 und als CP/M-Rechner mit der CPU Z80 ab. Dann bleibt für den PC-128-Modus herauszustellen: Er hat drei belegte Speicherbänke. In Bank 15 liegen ab $40000 BASIC Low und ab $8000 BASIC High. Ab $C000 folgt ein erweitertes Monitor-Programm, an das sich bei $D000 der Ein- und Ausgabebereich anschließt. Ab $E000 sind dann der Editor und der Kernel codiert. Bank 0 und Bank 1 sind reine RAM-Bänke zu 64 KB.

**Die Rolle der Memory Management Unit**
Da auch die CPU 8502 nur 16 Adreßbuspins hat, mit denen der Speicherraum auf 64 KB beschränkt ist, mußte zur Systemsteuerung eine Memory Management Unit (MMU) 8722 benutzt werden. Hinzu kommt ein Baustein 8721 zur Adreßmodifikation. Die MMU hat folgende wesentliche Aufgaben:

Zuordnung der benutzten RAM-Bank,
der Festwertspeicher
und der Ein- und Ausgabe per Adressen $D500/$FF00
Steuerung der Modi 40/80 Zeichen je Zeile,
PC-128 oder C-64-Modus,
Steckmodul EXROMIN im C-64-Modus,
Betrieb mit schneller Floppy
und eingeschalteter Prozessor 8502/Z80 per Adresse $D505
Steuerung der Common-Area im RAM per Adresse $D506
Vorgabe des Offsets Low/High der Zero Page per Adressen $D507/$D508
Vorgabe des Offsets Low/High der Stackseite per Adressen $D509/$D50A

Es sind diese Steuerungen durch die MMU, mit denen wir uns nachfolgend in ihrer Bedeutung für den Assembler-Programmierer befassen werden.

Zunächst noch der Hinweis auf die zwei Bausteine zur Bildschirmausgabe. Im 40-Zeichen-Modus wird der Baustein VIC 8564 benutzt. Seine Register sind ab Adresse $D000 dekodiert. Zur Zeichenausgabe greift er auf das RAM des Systems zu, und zwar ab Adresse $400...$7FF. Bei der Ausgabe von 80 Zeichen/Zeile ist dagegen der Baustein VDC 8563 aktiv. Seine zwei Register sind an der Adresse $D600 das Adreßregister (Vorwahlregister) für das Schreiben, das auch als Statusregister für das Lesen dient und bei $D601 das Datenregister zum Lesen und Schreiben. Dieser Baustein verwaltet ein eigenes Video-RAM von 16 KB, das dem direkten Zugriff durch die CPU entzogen ist. Es ist also ein RAM zusätzlich zu den beiden RAM-Speicherbänken, die dem Prozessor zur Verfügung stehen. Das Video-RAM kann nur über den engen Trichter der beiden Register bei $D600/$D601 erreicht werden.

**Das Konfigurationsregister der MMU, $D500/$FF00**
Der vorstehende Abschnitt zur Rolle der MMU ließ bereits ahnen, daß (unabhängig von Betriebsweisen C-64 oder Z80) innerhalb des Rechners viele Zuordnungen vorgenommen werden können, die zunächst einmal verwirrend sein mögen. Die Frage ist, wie man damit umgehen soll, wenn man den Rechner in eine

Umgebung bringen möchte, in der er ein reines Assemblerprogramm ausführt oder in der aus dem BASIC heraus maschinensprachliche Routinen eingebunden werden sollen. - Wir gehen daher zunächst auf das Bitmuster im Konfigurationsregister ein, ehe weitere Begriffe zum PC-128 benutzt werden.- Zunächst ist deutlich darauf hinzuweisen, daß die Register der MMU wie die Register eines Interface-Bausteines unter Speicheradressen erreicht werden, und zwar ab $D500...$D50B. Nun gehören Adressen im Bereich ab $D000 zum Ein- und Ausgabebereich, die durch die MMU selbst in ihrer Speicherdisposition ein- und ausgeblendet werden können.

So erhebt sich die Frage, was denn geschehen könnte, wenn ihre eigenen Adressen einmal ausgeblendet sein sollten, wenn z.B. in den Bereich $C000...$FFFF RAM oder ein Festwertspeicher eingeblendet ist. Die Antwort ist: Eine anprechbare Kopie der Adressen $D500...$D504 befindet sich immer unter den Adressen $FF00...$FF04. Das ist eine bevorzugte Decodierung der MMU und des Adreßmodifiers. Selbst wenn ab $C000...$FF00 RAM disponiert ist, lesen wir an diesen Adressen die Register der MMU aus und können diese Adressen mit Wirkung auf die MMU beschreiben; sie sind also immer "sichtbar". Für die Praxis der Programmierung heißt es, daß der Computer damit beherrschbar

bleibt, auch in der Bedienung der Interfaces. Es heißt aber zugleich, daß wir hier bei $FF00 einen Sprung von 5 Speicherzellen im Kontinuum des Adreßraumes haben. Das ist zu berücksichtigen, wenn im Bereich ab $FF00 in jedweder Bank statt des dort normalerweise benutzten Kernels eigene Programmteile oder Daten des Benutzers angesiedelt werden sollten. - Die 5 Zellen bei $FF00 können als zweite Haustüren der Zellen bei $D500 angesehen werden: Selbst wenn der E/A-Bereich nicht eingeschaltet ist (s.u.), bleibt die MMU erreichbar.

Es werden jetzt die Bits des Konfigurationsregisters in ihren Steuerfunktionen besprochen. Dazu wird auch auf die nebenstehende Schemazeichnung hingewiesen. Es steuern:

Bits 7, 6 die aktive RAM-Bank. Sinnvoll sind hier nur 00 und 01 für die Bänke 0 und 1. Höhere Binärwerte ziehen eine Spiegelung in diese Bänke nach sich.

Bits 5,4 die Disposition für den Adreßbereich $C000...$FFFF
00= Einschaltung des Kernel ROMs
01= Interner ROM-Sockel
10= Externes ROM in der Cartridge
11= RAM in der aktivierten Bank
Bit 0=0 setzt sich für den Bereich $D000...$DFFF mit der Einblendung des E/A-Bereiches über eine Disposition wie vor hinweg.

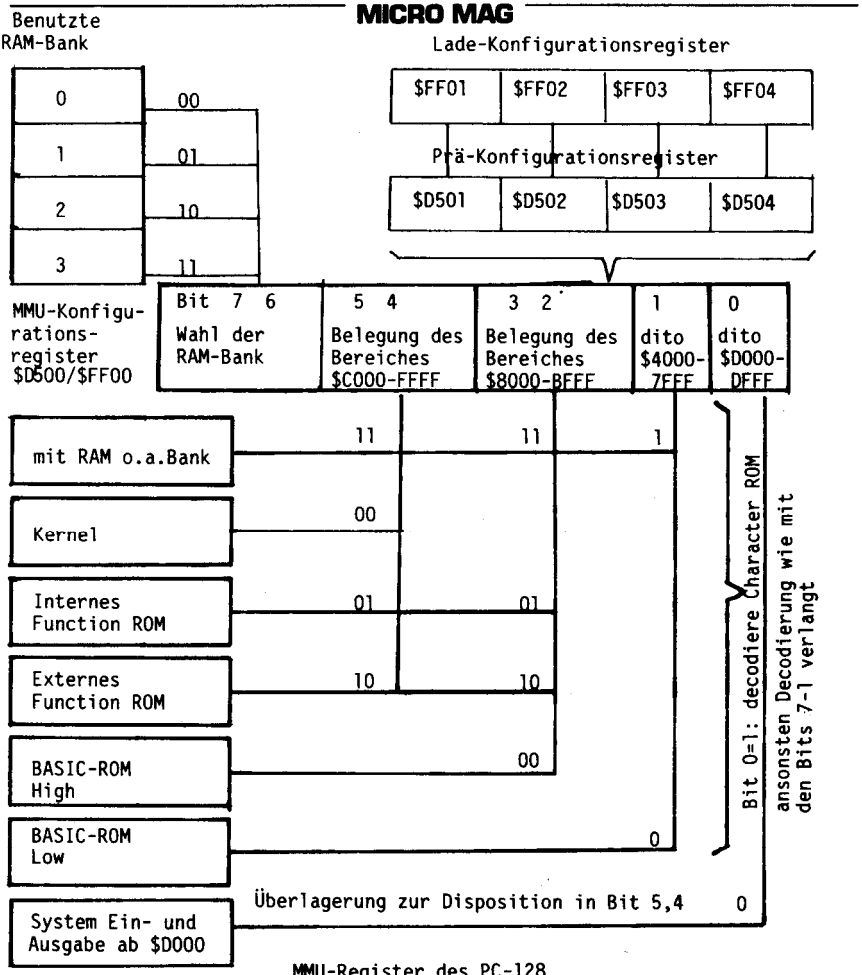
Bits 3, 2 die Disposition für den Adreßbereich $8000...$BFFF
00= Einschaltung des BASIC-ROMs High
01= Interner ROM-Sockel
10= Externes ROM in der Cartridge
11= RAM in der aktivierten Bank

Bit 1 die Disposition für den Adreßbereich $4000...$7FFF
0 = Einschaltung des BASIC-ROMs Low
1 = Einschaltung von RAM in der aktivierten Bank

Bit 0 Einschaltung des E/A-Bereiches $D000...$DFFF
0 = Einschaltung des E/A-Bereiches unter Überlagerung der in Bits 5, 4 getroffenen Disposition
1 = Keine Einblendung des E/A-Bereiches

Hierzu kurze Beispiele: 00 in der Konfiguration bedeutet: RAM-Bank 0, Kernel und gesamtes BASIC sowie der E/A-Bereich werden eingeschaltet. Das ist eine Konfiguration, in der das BASIC seine Routinen in den Festwertspeichern abarbeitet. - Die Konfiguration $0E bedeutet: RAM-Bank 0 wird aktiviert, der Kernel und der E/A-Bereich stehen zur Verfügung, von $0000 bis zur Adresse $BFFF steht RAM in der Zugriffsmöglichkeit, das sind 48 KB in Bank 0. Eine solche Umgebung kann die eines großen steuernden Assembler-Programmes sein, das seine Variablen mit einer anderen Konfiguration in Bank 1 unterhält.

**Präkonfigurationsregister in $D501...$D504**
**Trigger-Register in $FF01...$FF04**
Ein Beschreiben der Adresse $FF00 oder $D500 wirkt sich immer sofort auf die Konfiguration aus. Es wurde daher ein Mechanismus geschaffen, mit dem die Umschaltung der Bereichszuweisungen vereinfacht ausgeführt werden kann. Die Präkonfigurationsregister sind dabei 4 gleichberechtigte Merkzellen, deren Beschreiben zunächst nichts weiter auslöst. Erst wenn man einen beliebige Wert mit STA, STX oder STY in ein entsprechendes Trigger-Register bei $FF01... schreibt, dann wird der bei $D501...$D504 gemerkte Wert sofort in das Konfigurationsregister der MMU übertragen. Diese Trigger-Register werden auch Load Configuration Register genannt.




MMU-Register des PC-128
und Decodierung der Systembereiche

Man hat also die Wahl, entweder direkt in das sichtbare Konfigurationsregister zu schreiben oder aber den Trigger-Mechanismus mit vorgespeicherten Konfigurationen auszulösen. Dabei ist klar, daß ein Programm sich selbst mit dem Konfigurationswechsel ausschalten kann, so daß sich der Computer verläuft. Es bedarf daher an kritischen Stellen nicht nur einer sorgfältigen Planung, sondern auch weiterer Mechanismen für die kontrollierte Programmfortsetzung.

Als unkritisch kann man zunächst einmal Programme bezeichnen, die nebst ihren Daten mit 48 KB RAM und ohne BASIC auskommen. Für sie wählt man wie in obigem Beispiel die Konfiguration $0E, wenn das Programm in Bank 0 laufen soll, oder entsprechend $4E für Bank 1.

**Gemeinsame RAM-Bereiche, Common Area**
**RAM Configuration Register in $D506**
Mit der MMU wurde die Möglichkeit geschaffen, einen RAM-Bereich der Bank 0 in

andere RAM-Bänke einzublenden. Das ist dann ein gemeinsamer Bereich. Also: Auch wenn z.B. RAM-BAnk 1 aktiviert ist, liest und schreibt man tatsächlich in Bank 0, wenn ein Common-Bereich aktiviert wurde. Warum nun ist eine solche Einrichtung nützlich? - Der Leser weiß, das sowohl das Betriebssystem, Sprachen wie auch Anwenderprogramme Platz für eigene Variablen benötigen. Wenn nun insbesondere die Variablen des Betriebssystems nach einer Umschaltung der RAM-Bank nicht mehr erreichbar wären, dann würde keine Ein- und Ausgabe mehr klappen, weil diese auf Zeiger in der Bank 0 angewiesen sind. Es ist daher sinnvoll, einen Teil der Bank 0 mit solchen Parametern immer im Zugriff zu haben. Dem dient das RAM Configuration Register. Seine Bits steuern wie folgt:

Bits 7,6 Einblendung des VIC-Chips (40 Zeichen-Ausgabe)
00= Überlagerung in die Bank 0
01= in Bank 1, 10 und 11 vorgesehen für Bänke 2/3
Bits 5,4 00= Benutzung des ersten 256K-Blockes
01= Benutzung des zweiten usw.
Bits 3,2 Lage der Common Area
00= Keine Common Area, Bits 1 und 0 bleiben ohne Bedeutung
01= Gemeinsamer Bereich unten im RAM
10= Gemeinsamer Bereich oben im RAM
11= Gemeinsamer Bereich unten und oben
Bits 1,0 Größe des gemeinsamen Bereiches
00= 1 K
01= 4 K
10= 8 K
11= 16 K

Aus der dokumentierten Speichernutzung und aus Experimenten ergibt sich, daß man mit 4 KB Common Area auskommt, wenn man den Kernel und die E/A eingeschaltet hat und sein Assemblerprogramm z.B. in Bank 1 zur Ausführung bringt. In diesem Falle würde man eine RAM-Configuration von $45 nehmen. - Zusammenfassend kann man zu den gemeinsamen Bereichen sagen, daß sie den Transport von Parametern und von ganzen Zero Pages von Bank zu Bank zu vermindern helfen bzw. entbehrlich machen. Gleichwohl bleibt es dem Anwender unbenommen, einem Programm eine eigene Zero Page einzuräumen.

**Verschiebung der Zero Page und der Stack Page**

Die MMU macht es weiterhin möglich, zwei unabhängige Fenster über das RAM zu schieben, in denen die Zero Page bzw. die Stackseite enthalten sind. Hier wird mit einem neuartigen Offset gegenüber der normalen Lage gearbeitet: Es sind zu hinterlegen in $D507 der Offset gemessen in ganzen Pages (z.B. $0A), in $D508 die Bank, in der die Zero Page liegen soll, also z.B. $01. In diesem Beispiel würde sie bei $10A00 liegen, in Bank 1 ab Adresse $0600. - Das Beispiel kann erweitert werden. Nehmen wir einmal an, in Adresse $10A00/01 befinde sich der Pointer $2223 und es sei Y=0. Dann führt der Befehl LDA (00),y zum Laden aus der Adresse $2223.

Der gleiche Mechanismus kann auf die Lage des Stacks angewandt werden. Für ihn wird die gewählte Bank in $D50A und der in Pages gemessene Offset in $D509 des E/A-Bereiches hinterlegt. Und noch ein Hinweis: Was man in die Register für die gewählte Bank hineinschreibt, bleibt zwischengespeichert. Die Umschaltung der Zero Page oder der Stackpage tritt erst in dem Moment ein, wo man nach $D507 bzw. nach $D509 schreibt.

**Der Griff in die andere Bank**

Die CPU 8502 des PC-128 hat keine Befehle oder Adressierungsarten, mit denen sie in eine andere Speicherbank greifen könnte. Die Commodore-CPU 6509 in den Rechnern CBM 710/720 konnte wenigstens mit den Befehlen LDA (Pointer),Y und

STA (Pointer),Y Daten in anderen Banken direkt anfassen, wenn das Datenbankregister in Adresse $01 auf die Bank wies. Das Programmbankregister wurde durch den Inhalt der Zelle $00 gesteuert. In der 65xxx-Familie gibt es erstmals mit dem G65SC816 die Möglichkeit, Daten in beliebigen Banken anzufassen, ohne daß man als Programmierer auf Speichersegmentierungen achten muß. Bei der letzteren CPU kann man auch ohne besonderen Aufwand das Programm in einer anderen Bank fortsetzen oder Unterprogramme in anderen Banken aufrufen und zur Bank des Callers automatisch zurückkehren. Noch einfacher werden die Verhältnisse bei der CPU 68000 und ihren Geschwistern, bei der der Speicherraum ein Kontinuum von einigen Megabyte für Programme und Daten ist. Man braucht den Lesern wohl kaum die rhetorische Frage zu stellen, welche geradlinigere Architketur in Zukunft wohl mehr benutzt werden wird. - Nach diesem Vergleich muß man sagen, daß die CPU 8502 sich von der 6502 wesentlich nur durch ihre 7 Portpins unterscheidet und damit eine alte Architektur verwirklicht. Wenn der Computer PC-128 gleichwohl mehr als 64KB Speicher adressieren kann, so liegt das an der Memory Management Unit und am Adressmodifier.

Rechner wie der CBM 720 und der C-64 machten bereits klar, daß man wegen der Architektur jetzt eine noch deutlichere Unterscheidung zwischen Programmen und ihren Daten machen muß. Programme dürfen per Veränderung des Konfigurationsregisters sich nicht selbst aus der Programmfortsetzung abschalten. Im ersten Durchgang ist daher zu empfehlen, Programmcode und Daten wie bei den bisherigen 8 Bit-Maschinen ohne Umschaltung der Konfiguration zu disponieren. Das geht nur bis zu einem maximalen Umfang von Programm und Daten.

Der zweite gedankliche Ansatz besteht darin, Zugriffe zu Daten in anderen (verdeckten) Konfigurationen und Unterprogrammaufrufe in den gemeinsamen Speicherbereich zu verlegen. Dabei werden solche Griffe in andere Banken in der Common Area selbst wieder als Unterprogramme angelegt. Aufrufe von hierher führen dann auf den Folgebefehl in der Common Area zurück. - Das Betriebssystem des PC-128 macht von dieser Möglichkeit Gebrauch. Im gemeinsamen Bereich sind nach der Initialisierung des Computers folgende Dienstprogramme abgelegt, auf die wir noch zurückkommen:

```
$02A2 FETCH,  Lade den Akku mit dem Byte aus einer beliebigen Bank
$02AF STASH,  Speichere Akkuinhalt in eine beliebige Bank
$02BE CMPARE, Vergleiche Akku mit Byte in beliebiger Bank
$02CD JSRFAR, Führe ein Unterprogramm in einer fernen Bank aus
$02E3 JMPFAR, Programmsprung in beliebige ferne Bank
```

Man kann schon an dieser Stelle sagen, daß die Benutzung der vorstehenden Routinen mit etlichen Vorbereitungsarbeiten zur Übergabe der Parameter verbunden ist und daher Zeit und Code kostet. Das sind Erschwernisse, die durch die Rechnerarchitektur bedingt sind. Und man muß hinzufügen, daß JSRFAR zu einer Konfiguration zurückführt, die dem BASIC-Interpreter entspricht, nicht jedoch notwendig in die Umgebung von reinen Assemblerprogrammen. Wir stellen diese Routinen zunächst dar und knüpfen daran Überlegungen an, wie man evtl. einfacher arbeiten kann.

**Die indirekten Routinen**

Betrachten wir zunächst einmal das Laden eines Bytes aus einer anderen Bank in den Akku: Im Kernel fängt die Routine beim Label INDFET an. Der Akku enthält die Adresse des Pointers in der Zero Page, der benutzt werden soll. Diese Adresse wird in das RAM an der Befehlsstelle FETCHX+1 übertragen. Mit dem Register X muß man weiterhin die Nummer der gewünschten Speicherorganisation ausgewählt haben, deren Wert aus der Standard-Tabelle CONFIG ausgewählt wird. Auch will bedacht werden, daß der Pointer zunächst einmal vorbereitet sein

muß. Dann erfolgt der Weitersprung zur RAM-Routine FETCH. Dort wird die Konfiguration vorübergehend umgeschaltet, ein Byte wird in Zeile 52 per Adressierungsart LDA (Pointer),Y geladen. Die Programmfortsetzung erfolgt kontinuierlich in der Common Area im Zeilenbereich 49...54. - Etwa gleichartige Routinen haben wir mit INDSTA und INDCMP für das Abspeichern des Akkus und für den Vergleich. Andere Befehle, wie z.B. die arithmetischen und die logischen sind nicht implementiert. Auch müssen wir feststellen, daß das X-Register hier immer verändert wird, was zu weiterer Umsicht zwingt.

Komplizierter wird es beim Befehl JSRFAR (in Zeile 80). Er ruft zunächst den auch einzeln zu gebrauchenden Befehl JMPFAR als Unterprogramm auf. Dieser verlangt, daß für alle Register in der vorübergehenden Ablage ab Adresse $0003 Werte vorbesetzt sind, die dann auf den Stack für das RTI und in die Register geladen werden. Das RTI führt dann zur Programmausführung in der anderen Bank und zur Rückkehr aus JSR JMPFAR nach Zeile 81. Danach werden die Register abgespeichert. - Hier ist nun darauf hinzuweisen, daß in Zeile 90 die Standard-Konfiguration eingeschaltet wird, die dem BASIC entspricht. Sie dürfte in vielen Fällen nicht mit derjenigen identisch sein, die man für ein Assemblerprogramm haben möchte. Das läßt sich abändern, indem man in Zelle $02de einen anderen Konfigurationswert schreibt.

```
0002                                ;@ko"routinen pc-128"
0003 /* pc-128, durchgriff in andere speicherkonfigurationen
0004 label-namen gem. c-128 rom-listing, operating system
0005 */
0006 000000          bank      =$02              bankvorgabe
0007 000000          pc_high   =$03              vorgabe programmzaehler
0008 000000          sreg      =$05              prozessor-status
0009 000000          areg      =$06              akku
0010 000000          xreg      =$07              index x
0011 000000          yreg      =$08              index y
0012 000000          stkptr    =$09              stackpointer
0013 000000          pointer   =$ff              dummy-variable
0014 000000          fetch     =$002a2           ram-routine
0015 000000          mmucr     =$0ff00           konfigurationsregister
0017 000000                    *=$ff7d0          routinen im kernel
0018                           ;zum laden, abspeichern, vergleichen
0019 0ff7d0 8daa02   indfet    sta fetchx+1      pointer aktivieren
0020 0ff7d3 bdf0f7             lda config,x      auswahl der konfiguration mit
                                                  x
0021 0ff7d6 aa                 tax               x steuert das mmucr
0022 0ff7d7 4ca202             jmp fetch         fortsetzung im ram per lda (p
                                                 ointer),y
0024 0ff7da 48       indsta    pha               zeichen sichern
0025 0ff7db bdf0f7             lda config,x      ein zeichen speichern
0026 0ff7de aa                 tax               x steuert das mmucr
0027 0ff7df 68                 pla               zu speicherndes zeichen
0028 0ff7e0 4caf02             jmp stash         fortsetzung im ram per sta (p
                                                 ointer),y
0030 0ff7e3 48       indcmp    pha               ein zeichen vergleichen
0031 0ff7e4 bdf0f7             lda config,x      auswahl der konfiguration mit
                                                  x
0032 0ff7e7 aa                 tax
0033 0ff7e8 68                 pla               vergleichszeichen zurueck
0034 0ff7e9 4cbe02             jmp cmpare        fortsetzung im ram
0036 0ff7ec bdf0f7   getcfg    lda config,x      hole konfiguration 0...15
0037 0ff7ef 60                 rts
0039 0ff7f0 3f7fbfff config    .byt $3f,$7f,$bf,$ff alles auf ram-bank (0...3)
                                                  geschaltet
```

```
0040 0ff7f4 165696d6          .byt $16,$56,$96,$d6 ram-bank 0...3 mit internen
                                                   roms und i/o
0041 0ff7f8 2a6a2aea          .byt $2a,$6a,$2a,$ea ram-bank 0...3 mit externen
                                                   roms und i/o
0042 0ff7fc 06                .byt $06             ram-bank 0, kernel, internes
                                                   rom $8000, i/o
0043 0ff7fd 0a                .byt $0a             wie vor mit externem rom
0044 0ff7fe 01                .byt $01             ram-bank 0, kernel, basic, ze
                                                   ichengenerator-rom
0045 0ff7ff 00                .byt $00             ram-bank 0, kernel, basic, i/
                                                   o eingeblendet
0047 0ff800                   *=fetch              routinen im ram
0048                          ;laden eines zeichens, vorgabe in pointer und y
0048
0049 0002a2 ad00ff   fetch    lda mmucr            alte konfiguration merken
0050 0002a5 8e00ff            stx mmucr            neue konfiguration aktivieren
0051 0002a8 aa                tax                  merken jetzt in x
0052 0002a9 b1ff     fetchx   lda (pointer),y      vor-geladener pointer
0053 0002ab 8e00ff            stx mmucr            alte konfiguration
0054 0002ae 60                rts
 056                          ;speichern eines zeichens, vorgabe in pointer u
0056                          nd y
0057 0002af 48       stash    pha                  zu speicherndes zeichen
0058 0002b0 ad00ff            lda mmucr            alte konfiguration
0059 0002b3 8e00ff            stx mmucr            neue konfiguration
0060 0002b6 aa                tax                  gemerkte konfiguration
0061 0002b7 68                pla                  zu speicherndes zeichen
0062 0002b8 91ff              sta  (pointer),y     abspeicherung
0063 0002ba 8e00ff            stx mmucr            alte konfiguration
0064 0002bd 60                rts
0066                          ;vergleich eines zeichens, vorgabe in pointer u
0066                          nd y
0067 0002be 48       cmpare   pha                  zu vergleichendes zeichen
0068 0002bf ad00ff            lda mmucr            alte konfiguration
0069 0002c2 8e00ff            stx mmucr            neue konfiguration
0070 0002c5 aa                tax                  gemerkte konfiguration
0071 0002c6 68                pla                  zu vergleichendes zeichen
0072 0002c7 d1ff              cmp (pointer),y      vergleich
0073 0002c9 8e00ff            stx mmucr            alte konfiguration
0074 0002cc 60                rts
0076 /* aufruf eines unterprogrammes in einer anderen konfiguration
0077 die parameter muessen in der common page ab adresse $02
0078 vorgeladen sein
0079 */
0080 0002cd 20e302 jsrfar     jsr jmpfar           unterprogrammaufruf in andere
                                                    bank
0081 0002d0 8506              sta areg             erhaltenen akku merken
0082 0002d2 8607              stx xreg
0083 0002d4 8408              sty yreg
0084 0002d6 08                php                  dito status
0085 0002d7 68                pla
0086 0002d8 8505              sta sreg
0087 0002da ba                tsx                  dito stackpointer
0088 0002db 8609              stx stkptr
0089 0002dd a900              lda #00              konfiguration: kernel, basic,
0090 0002df 8d00ff            sta mmucr            ram-bank 0, i/o eingeschaltet
0091 0002e2 60                rts
```

```
0093 /* sprung in eine andere konfiguration
0094 */
0095 0002e3 a200     jmpfar    ldx #0              startdresse high, low
0096 0002e5 b503     jmpfar0   lda pc_high,x       und status transportieren
0097 0002e7 48                 pha                 und zwar aus den vorgaben im
                                                   common ram
0098 0002e8 e8                 inx
0099 0002e9 e003               cpx #3
0100 0002eb 90f8               bcc jmpfar0
0101 0002ed a602               ldx bank            gewuenschte konfiguration
0102 0002ef 20ecf7             jsr getcfg          holen
0103 0002f2 8d00ff             sta mmucr           umschaltung der konfiguration
0104 0002f5 a506               lda areg            besetzen der register
0105 0002f7 a607               ldx xreg
0106 0002f9 a408               ldy yreg
0107 0002fb 40                 rti                 aufruf
0107 0002fc                                        rti aufruf
0108 0002fc                    .end
```

**Erweiterung der Durchgriffs-Möglichkeiten**

Der Autor hat sich die Mühe gemacht, einen anderen Mechanismus zu entwerfen und auszuprobieren, der in der nachfolgenden Liste dargestellt wird. Bei BANK JSR und bei BANK BEF spielt folgende Überlegung mit: Ein JSR-Befehl legt die Rückkehradresse-1 auf dem Stack ab. Wenn man sich diese Adresse auf dem Stack beschafft, dann kann man relativ zu ihr auf Parameter Zugriff nehmen. Bei BANK JSR wird die Adresse aus dem nachfolgenden beabsichtigten JSR-Befehl beschafft und bis zum indirekten Sprung in Zeile 48 aufgearbeitet, der nach der Konfigurationsumschaltung in Zeile 47 in die neue Bank führt. Zuvor mußte aber auch die Rückkehradresse auf dem Stack um 3 erhöht werden, um der Verschiebung gerecht zu werden. - Der Test zu diesem Programm ist der Abschnitt TEST1 ab Zeile 104. Dort werden die Präkonfigurationsregister standig so eingerichet, daß das steuernde Programm in Bank 0 abläuft, daß es aber Unterprogramme in Bank 1 aufrufen kann. Die entscheidende Syntax des Aufrufes ist in den Zeilen 111...113: BANK JSR schaltet auf die Zielbank und dort auf die Beispielsadresse in der Zeile 112 ($4000) um. Nach dem Aufruf wird die ursprüngliche Konfiguration in Zeile 113 zurückgerufen.

Das Unterprogramm BANK BEF ab Zeile mit TEST2 geht ebenfalls von der genannten Überlegung aus. Hier wird aber ein ganzer 2-3 Byte langer Befehlscode in die Adressen $FC7A-FF7C transportiert und in der anderen Bank zur Ausführung gebracht. Wir können damit alle Zero Page und absolut adressierenden Befehle zur Ausführung in anderen Banken bringen. Auch hier ist die Prozedur der Parameterübergabe mechanisiert. - Damit ist es möglich, Daten in anderen Banken beliebig zu prüfen und zu bearbeiten. Sofern man eine eigene Zero Page einrichten möchte, so sollte man entsprechend in den Adressen $D507/08 (s.o.) disponieren. Man muß dann nur aufpassen, daß sie ggfs. nicht einen gemeinsamen Bereich bildet und daß Routinen des Kernels ihre gewohnten Zeiger wiederfinden.

In der Summe läßt sich feststellen, daß die CPU 8502 zwar keine eigenen Möglichkeiten hat, durch Bänke und Konfigurationen hindurch zu arbeiten. Das wird erst durch die Memory Management Unit ermöglicht. Routinen, wie die vorstehenden verbrauchen zwar nur wenig Platz in einem gemeinsamen Bereich, sie kosten aber einiges an Zeit und begleitendem Beiwerk. Wenn man die notwendige einfache Syntax bedenkt, dann ist Assembler-Programmierung auf dem PC-128 letztlich auch wieder nicht so schwer.

```
0001                                ;@ko"eigene 128rout"
0002 /* bank_jsr geht davon aus, da  die zielbank im
0003 preconfiguration register pcrc bei $d503 vorgeladen ist.
0004 und die aufrufende bank im register pcrd bei $d504
0005 der aufruf erfolgt so: jsr bank_jsr, dann jsr adresse
0006 dahinter dann sta lcrd
0007 diese routine muss in der common area liegen
0008 */
0009 000000          imparm    =$ce              pointer/sprungadresse
0010 000000          mmurcr    =$d506            ram configuration register
0011 000000          mmucr     =$ff00            mmu-kontrollregister
0012 000000          lcrc      =$ff03            lade-konfigurationsregister
0013 000000          lcrd      =$ff04            dito
0015 000000                    *=$fc00           ansiedlung im letzten gemeins
                                                 amen 1k-bereich
0016 00fc00          bank_jsr                    ;ein unterprogramm in anderer
                                                  bank ausfuehren
0017 00fc00 08                 php               status retten
0018 00fc01 48                 pha               akku
0019 00fc02 8a                 txa               index x
0020 00fc03 48                 pha
0021 00fc04 98                 tya               index y
0022 00fc05 48                 pha               4 items gerettet
0023 00fc06 ba                 tsx               stackpointer holen
0024 00fc07 18                 clc
0025 00fc08 bd0501             lda $105,x        alte return-adresse low
0026 00fc0b 85ce               sta imparm        pointer aufbauen
0027 00fc0d 6903               adc #3            zeige auf das 3. byte des fol
                                                 genden
0028 00fc0f 9d0501             sta $105,x        jsr-befehls = neue return-
0029 00fc12 bd0601             lda $106,x        adresse
0030 00fc15 85cf               sta imparm+1      pointer high
0031 00fc17 6900               adc #0            uebertrag einsammeln
0032 00fc19 9d0601             sta $106,x
0033 00fc1c a002               ldy #2            nun adresse aus dem jsr-befeh
                                                 l holen
0034 00fc1e b1ce               lda (imparm),y
0035 00fc20 48                 pha               low merken
0036 00fc21 c8                 iny
0037 00fc22 b1ce               lda (imparm),y    adresse high
0038 00fc24 85cf               sta imparm+1      pointer hat jetzt ausgedient
0039 00fc26 68                 pla
0040 00fc27 85ce               sta imparm        low, neuer pointer fertig
0041 00fc29 68                 pla
0042 00fc2a a8                 tay               register zurueckladen
0043 00fc2b 68                 pla
0044 00fc2c aa                 tax
0045 00fc2d 68                 pla
0046 00fc2e 28                 plp               status
0047 00fc2f 8d03ff             sta lcrc          bank/konfiguration umschalten
0048 00fc32 6cce00             jmp (imparm)      routine ausfuehren

0050 /* bank_bef = beliebigen zero page- oder absoluten
0051 befehl in einer anderen bank ausfuehren
0052 art des aufrufes: jsr bank_bef - befehl in anderer bank
0053 dann sta $ff04
0054 */
0055 00fc35 08       bank_bef  php               status retten
0056 00fc36 48                 pha               akku
```

```
0057 00fc37 8a                txa                 index x
0058 00fc38 48                pha
0059 00fc39 98                tya                 index y
0060 00fc3a 48                pha                 4 items gerettet
0061 00fc3b ba                tsx                 stackpointer holen
0062 00fc3c 18                clc
0063 00fc3d bd0501            lda $105,x
0064 00fc40 85ce              sta imparm          pointer aufbauen
0065 00fc42 6903              adc #3
0066 00fc44 9569              sta 105,x           neue return-adresse low
0067 00fc46 bd0601            lda $106,x
0068 00fc49 85cf              sta imparm+1
0069 00fc4b 6900              adc #0
0070 00fc4d 9d0601            sta $106,x          neuer return high
0071 00fc50 a003              ldy #03             nun den code aus folgendem be
                                                  fehl beschaffen
0072 00fc52 b1ce     bank_bef1 lda (imparm),y     low steht 2 bytes hinter alte
                                                  m return
0073 00fc54 9979fc            sta bank_bef3-1,y
0074 00fc57 88                dey
0075 00fc58 d0f8              bne bank_bef1       3 bytes transportieren
0076 00fc5a c94c              cmp #$4c            sprungbefehl?
0077 00fc5c f013              beq bank_bef2
0078 00fc5e c96c              cmp #$6c            indirekter sprung?
0079 00fc60 f00f              beq bank_bef2
0080 00fc62 c920              cmp #$20            jsr?
0081 00fc64 f00b              beq bank_bef2
0082 00fc66 290f              and #$0f            maskieren
0083 00fc68 c909              cmp #$09            befehl von 2 oder 3 bytes?
0084 00fc6a b005              bcs bank_bef2       3 bytes!
0085 00fc6c a9ea              lda #$ea            nop befehl als 3. byte einset
                                                  zen
0086 00fc6e 8d7cfc            sta bank_bef3+2
0087 00fc71 68       bank_bef2 pla                 register zurueck
0088 00fc72 a8                tay
0089 00fc73 68                pla
0090 00fc74 aa                tax
0091 00fc75 68                pla
0092 00fc76 28                plp                 status
0093 00fc77 8d03ff            sta lcrc            umschalten der bank/konfigura
                                                  tion
0095 00fc7a ea       bank_bef3 nop                 hier landet der transportiert
                                                  e
0096 00fc7b ea                nop                 code des auszufuehrenden befe
                                                  hles
0097 00fc7c ea                nop
0099 00fc7d 60                rts
0101 /* erprobung des bank_jsr. es wird erwartet, dass bei adresse $4000
0102 in bank 2 ein mit rts abgeschlossenes unterprogramm steht.
0103 */
0104 00fc7e ad00ff   test1     lda mmucr
0105 00fc81 ce00ff            dec mmucr           e/a-bereich voruebergehend ei
                                                  nblenden
0106 00fc84 8d04d5            sta $d504           praekonfigurationsregister d
0107 00fc87 a97f              lda #$7f            ram-bank 1
0108 00fc89 8d03d5            sta $d503           praekonfigurationsregister c
0109 00fc8c a90c              lda #$0c            lk ram oben und unten gemeins
                                                  am
0110 00fc8e 8d06d5            sta mmurcr          ram-konfiguration
```

```
0111 00fc91 2000fc              jsr bank_jsr        vorbereitung/austuehrung
0112 00fc94 200040              jsr $4000           dieses jsr-befehls in anderer
                                                     bank
0113 00fc97 8d04ff              sta lcrd            alte konfiguration einschalte
                                                    n
0114 00fc9a 00                  brk                 ende des test1
0116 /* erprobung eines beliebigen befehles in anderer bank
0117 es wird davon ausgegangen, dass die praekonfiguration
0118 nach test1 bereits ausgefuehrt wurde
0119 */
0120 00fc9b a999      test2     lda #$99            abzulegender wert
0121 00fc9d a202                ldx #2              index
0122 00fc9f 2035fc              jsr bank_bef        vorbereiten ausfuehren
0123 00fca2 9d0040              sta $4000,x         dieses befehles in anderer ba
                                                    nk
0124 00fca5 8d04ff              sta lcrd            alte konfiguration zurueck
0125 00fca8 00                  brk                 ende test2
0126 00fca9                     .end
```

Roland Löhr

# PC-128 High Way

Zum PC-128 von Commodore gibt es das Modul HIGH WAY in Form einer Cartridge mit 32 KB Festwertspeicher. Zusätzlich zum eigentlich schon sehr reichhaltigen Befehlssatz BASIC 7 des Computers enthält dieses Programmiersystem der Fa. SAS Berndt etwa 230 Befehle, die vor allem den BASIC-Programmierer in eine sehr komfortable Umgebung führen. Es würde sicher zu weit führen, hier alle neuen Möglichkeiten aufzulisten. So beschränken wir uns darauf, seine wesentlichen Merkmale zu besprechen. Der Leser kann das zugehörige etwa 200-seitige Handbuch unabhängig von der Cartridge beziehen (DM 20), wenn er sich im Vorwege ein umfassenderes Bild machen möchte.

Als wesentlich darf man zunächst kennzeichnen, daß HIGH WAY seine Befehle und die zu ihnen benötigten Parameterübergaben selbst auf dem Bildschirm erklären kann, und zwar in einer Form, bei der man meistens nur noch Zeilennummern hinzutippen muß, um ausführbare Programmzeilen zu erhalten. - Zweitens kann man den Aufruf von Programmteilen in einer symbolischen Form unter Verwendung von Labeln bewirken. Unterprogramme können mit CALL ("Name") und einer Parameterliste aufgerufen werden. In diesem Falle sind die im Unterprogramm benutzten Variablen lokaler Natur, sie werden nach der Abarbeitung wieder freigegeben. Damit kann man mit Modulen arbeiten, ohne jeweils die Namen der Variablen anpassen zu müssen. - Drittens können Programme in einer Reihenfolge abgearbeitet werden, die in einem Submit-File festgelegt wird. Es sind weiterhin Funktionen zum Mischen und Anhängen von Programmteilen im Floppy-Betrieb vorgesehen.

Den größten Teil der Erweiterungen findet man im Zusammenhang mit der Behandlung von Strings. Hier sind nicht nur das Suchen, Ersetzen, Einfügen und Extrahieren geregelt, sondern auch der Line Input mit beliebigen Zeichen oder GET$ mit der Abfilterung auf zulässige Zeichen der Eingabe. Arrays können sortiert und Größenvergleiche können ausgeführt werden, ebenso ein einfaches SWAP zwischen Strings. - Der Inhalt von BASIC-Zeilen kann in Strings überführt werden und umgekehrt, Strings kann man auch in sich nach rechts oder links rotieren.

Zahlreich sind auch die Funktionen hinsichtlich der Umwandlung von und in Strings: Binärzahl aus einem String bilden, einen String aus einer Binär- oder

Dezimalzahl bilden, Zeichenumsetzung Tabelle gegen Tabelle, Umschaltung der Schreibweisen, eine Fließkommazahl zum String machen usw.

Bei den Rechenoperationen können für die vier Grundrechenarten Genauigkeiten bis auf 100 Stellen eingestellt werden, Reste, Maxima und Minima sind implementiert. - Mit Bitbefehlen können einzelne Bits in langen Strings gesetzt, gelöscht und geprüft werden.

Bei den Kontrukten zur Programmsteuerung ist insbesondere die Fallentscheidung mit SELECT und CASE zu erwähnen. Konstrukte können mit EXIT verlassen werden. Bezüglich des Diskettenbetriebes sehen einige Funktionen insbesondere die Ansprache von relativen Dateien vor. - Weitere Funktionen betreffen die Bildschirmverwaltung, das Schaffen und Beschicken von Eingabefenstern und die graphischen Fähigkeiten. Alle graphischen Funktionen des BASIC 7 stehen auch auf dem hochauflösenden Bildschirm mit 80 Zeichen zur Verfügung. Man kann damit auch wählen, auf welchen Bildschirm die Graphik ausgegeben werden soll. - Zu den Fähigkeiten des Programmiersystems gehört ferner die Anlage und Verwaltung von ISAM/VSAM Dateien mit index-sequentiellem Zugriff.

Die vorstehenden Erwähnungen können nur einen ersten Eindruck von den vielseitigen Fähigkeiten des HIGH WAY geben. In der Summe darf man aber wohl wie folgt urteilen: Viele alltägliche und auch besondere Aufgabenstellungen, die man im landläufigen BASIC erst in weitläufigen Formulierungen und Abfragen lösen kann, oftmals nur unter Einsatz von Maschinenprogrammen, die lassen sich hier mit wenigen Befehlsworten und Parameterübergaben bearbeiten. HIGH WAY unterstützt dabei die Programmierung durch sein Bildschirm Informations System (BIS), dessen Auskünfte man mit dem Einsetzen von Zeilennummern zu ausführbarem Code machen kann. Es wird weiterhin die Programmierung in übersichtlichen Modulen und Entscheidungen gefördert. Und es sind schließlich Hilfen für das Editieren und Ausprüfen von Programmen geboten. Damit sind leistungsfähige Instrumente für die Programmierung des PC-128 geboten. Info und Bezug: SAS, Hermann-Josef Berndt, Langgasse 93, 5216 Niederkassel 5, Tel.: 0228 - 45 26 26.

Roland Löhr

# Betriebssystem 65816

Die Zeitschrift mc veröffentlichte in den Heften 2/86 und 3/86 einen Europakarten-Computer unter dem Titel "Das doppelte Lottchen". Ein weiterer Artikel ist für Heft 6/86 vorgesehen. - Die Hardware wurde von der Firma MCT Paul & Scherer in Berlin entwickelt, das Betriebsprogramm für die CPU G65SC816 vom Autor. Damit ist Anlaß gegeben, in einer losen Folge von Abschnitten einmal ausführlicher über die Programmierung dieser neuen Zentraleinheit und gesammelte Erfahrungen zu berichten.

**Hardware des mc-65816-Computers**

Das Computerboard kann wahlweise mit der CPU 68008 (Motorola) oder der G65SC816 von GTE Microcircuits bestückt werden, letztere mit 4 MHz angetrieben und in CMOS-Technik. Auch die Speicher-, Interface- und Dekodierungsbausteine können wahlweise in CMOS bestückt werden, so daß ein sehr stromsparender Aufbau möglich ist (92 mA ohne VMEbus-Interface). Interfacebausteine sind eine ACIA 6551 für den Terminalbetrieb an einer RS232-Schnittstelle und eine VIA 6522 zur Verfügung des Benutzers. Diese VIA kann auch mit den zusätzlichen Bausteinen 75160 und 75161 mit einem sehr einfachen Interface für den parallelen IEEE 488-Bus beschaltet werden, so daß Meßgeräteanordnungen und entsprechende Commodore-Floppys und Drucker betrieben werden können. Dazu nebenstehend der Beschaltungsvorschlag, wie er vom Betriebssystem unterstützt

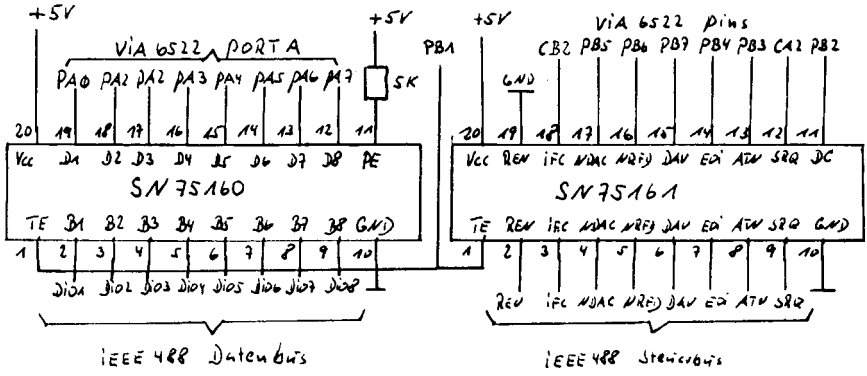



Abb.: Interface der System-VIA zum IEEE 488-Bus

wird. - Als RAM können die gängigen CMOS-Typen von 8..32 KB je nach Jumperung betrieben werden, ferner die EPROMs 2764...27512.

Diese Zeitschrift ging bereits früher auf die neue CPU G65SC816 ein. Kurz noch einmal ihre wichtigen Merkmale: Der 8-Bit Datenbus ist mit den Adreßbussignalen A16...A23 gemultiplext. Damit kann ein Speicherraum von 16 MB adressiert werden. Gegenüber der CPU 6502 ist die Beschaltung der Pins geändert, und es ist ein Latching der gemultiplexten Signale notwendig. - Nach einem kalten RESET meldet sich die CPU im Emulations-Modus, bei dem die Interruptvektoren NMI, IRQ und Reset wie beim 6502 adressiert und benutzt werden. Zusätzliche Vektoren liegen bei $FFF4 für den COP-Befehl und $FFF8 für das Eingangssignal ABORT. In diesem Modus haben die Register Akku, X und Y eine Verarbeitungsbreite von 8 Bit.

Der "ureigene Modus" der CPU wird durch die Befehlsfolge CLC und XCE hereingerufen. Das Carry-Bit im Status ist damit ein "Umklappregister" für den Modus. Im native mode, wie er englisch genannt wird, können die Register Akku, X und Y wahlweise in 8 oder 16 Bit Operandenbreite betrieben werden, und zwar in verschiedenen Kombinationen: a) alle Register in 8 Bit, b) alle Register in 16 Bit, c) Akku 8, Indexregister 16 Bit und d) Akku 16 und Indexregister 8 Bit breit arbeitend. Der entsprechenden Schaltung über Statusbits dienen die Befehle REP # und SEP #. Im native Mode werden 6 Ausnahmevektoren benutzt, die im Adressenbereich von $FFE4...$FFEF liegen müssen.

Neben der erhöhten Geschwindigkeit, dem gegenüber der 6502 erweiterten Befehlssatz der 65816 und verbesserten Adressierungsarten ist vor allen ihre Fähigkeit interessant, große Speicher zu adressieren. Das Hardware-Design ging daher davon aus, daß von Anfang an eine Erweiterung der CPU-Karte um Speicher und Interfaces möglich sein solle, und zwar ohne einen neuen exotischen Bus. Man entschied sich für den Industriestandard VMEbus, wie er von Motorola, den Zweitlieferanten und vielen mittelständischen Elektronikfirmen mit vorhandener Hardware unterstützt wird. Für den wahlweisen Betrieb der Karte mit einer CPU 68008 war das naheliegend, nicht jedoch für die CPU 65816, die einen synchronen Bus mit gleichmäßig schnell arbeitenden übrigen Bausteinen

erwartet, ohne ein Handshake der Bussignale mit DTACK. - Mit zusätzlicher Hardware, die u.a. den Phi2-Takt der CPU bei der Ansprache synchroner Bausteine nach Bedarf streckt, und weiteren Beschaltungen konnte auch der CPU 65816 ein asynchrones Busprotokoll auferlegt werden, das die Zusammenarbeit mit VMEbus-Karten ermöglicht. Dieses VMEbus-Interface ist im übrigen eine Option. Die Karte ist ohne ein solches "alleinstehend" als schneller Steuerungsrechner oder als kleines Entwicklungssystem einsatzbereit.

**Software-Design**
Wenn man eine CPU-Karte von den ersten Taktzyklen an mit einem Betriebssystem hochziehen will, dann sind zunächst einmal Planungen notwendig. Überraschungen lassen dann trotzdem nicht auf sich warten. Voraussetzungen sind zunächst einmal ein zweiter Computer mit Textsystem und einem Assembler für den neuen Befehlssatz und die Möglichkeit, EPROMs mit dem Zielcode zu programmieren. Die ersten Schritte bestehen dann in der Initalisierung des Reset-Vektors und der benötigten Interface-Bausteine, hier der ACIA für den Terminalbetrieb, bis man eine erste Reaktion des Systems wie erwartet erhält. Dann ist es an der Zeit, das gedankliche Gebäude weiter zu planen.

Die CPU benutzt die Speicherbank 0 (Adressen $0...$00FFFF) als Systembank. Hier erwartet sie ihren Resetvektor, ihren Stackbereich und hier legt sie ab $00 ihre Direct Page an (Zero Page bei 6502). Einige nützliche Befehle können nur in dieser Bank 0 zur Ausführung gebracht werden, so der indirekte Sprung JML auf einen Vektor mit 3 Adreßbytes. Mit ihm kann also eine Programmfortsetzung in einer beliebigen Bank erfolgen. Solche "weichen" Vektoren im RAM eignen daher hervorragend für die Umlenkung von Leistungen des Betriebssystems oder von Anwenderprogrammen. - Wegen solcher Überlegungen sollte die Bank 0 in einem System mit 65816 also möglichst ganz oder überwiegend eine RAM-Bank sein. Wie aber kann die CPU beim kalten Reset einen definierten Vektor in einer solchen Bank finden?

Der Designer der Karte entschloß sich zu dem Trick, das Betriebssystem, das in den Adressen ab $FF0000 arbeitet, beim Einschalten vorübergehend in die Bank 0 einzublenden. Mit dem Befehl JMP COLD1,L (langer Sprung) in Zeile 864 des nebenstehenden Programmauschnittes wird der Programmzähler der CPU auf die Bank $FF gesetzt, wo das EPROM normal dekodiert wird. Die drei Folgebefehle sprechen ein auf der Karte enthaltenes Flipflop an, das den Festwertspeicher künftig ab $FF0000 dekodiert. Damit ist der Speicher so konfiguriert, das in der Systembank 0 keine Programmbestandteile des Betriebssystems mehr gebraucht werden, es sei denn, man legt indirekte Vektoren dorthin (s.o.). Natürlich muß man jetzt oder etwas später dafür sorgen, daß 2 Sätze Ausnahmevektoren (für die Modi Emulation und native) in das höchste dekodierte RAM der Bank 0 gebracht werden, weil Interrupte sie dort erwarten. Das zur Initialisierung, soweit sie die Planung des Betriebsprogrammes betrifft.

**In 8 oder in 16 Bit arbeiten?**
Wenige Absätze zuvor wurde dargestellt, daß es für die Operandenbreite der Register im native mode, in den man ja möglichst hinein will, 4 verschiedene erlaubte Kombinationen gibt. Welche soll man wählen? Dazu die weiteren Überlegungen: Die Ein- und Ausgaben über die ACIA als Terminalbaustein erfolgen byteweise, ebenso diejenigen über das Interface zum IEEE 488-Bus, ferner die Entschlüsselung von Befehlstasten und alle Textverarbeitungen. Von daher bietet es sich an, im Betriebssystem grundsätzlich byte-orientiert zu arbeiten, wenigstens mit dem Akku in 8 Bit Breite.

Es bleibt dann die Frage, ob man die Indexregister in 16 Bit betreiben soll. Dagegen spricht, daß man für immer wiederkehrende Zwischenabspeicherungen von erreichten Registerständen doppelt soviel Speicherplatz und mehr Zeit

benötigt, wenn man, wie meistens, mit einer Registerbreite von 8 Bit auskommt. Einen Mischbetrieb sollte man wegen ggfs. etwas eleganter Abwicklungen nicht in Betracht ziehen, weil der Systemprogrammierer und der Anwender nicht unnötig Fehlermöglichkeiten ausgesetzt werden dürfen. Auch die Indexregister werden daher "native" in 8 Bit betrieben. - Das schließt nicht aus, daß man in Anwenderprogrammen und Sprachen streckenweise anders verfährt, bei der Heranziehung der vielen vorbereiteten Systemleistungen sollte man jedoch zum hier benutzten 8-Bit-Modus zurückkehren.

**Unterprogramme mit JSR oder JSL aufrufen?**
Mit dem Unterprogrammaufruf JSR bleibt man beim 65816 immer in der derzeit benutzten Programmbank, ob man nun aufruft JSR Label, JSR (Label) oder JSR (Label,X). Mit JSL Label kann man dagegen Unterprogramme in jeder beliebigen Bank erreichen. Diese sind dort mit dem Befehl RTL (Return from Subroutine Long) abzuschließen. In der Summe braucht diese Befehlsfolge zwar nur 1 Byte mehr, sie zieht aber höheren Zeitbedarf nach sich. Man könnte nun den Befehl JSR Label ganz aus dem Repertoire verschwinden lassen, damit man jedes Unterprogramm von überall her aufrufen und mit RTL verlassen kann. Das ist aus folgenden Gründen wohl doch zu weitgehend: Viele Unterprogramme haben nur eine "lokale" Bedeutung. Sie dienen dabei Erledigungen, die nur in einem lokalen Kontext von Interesse sind, oder sie sind nur Hilfsdienste zu Routinen, die global mit JSL aufrufbar sein müssen. Darunter versteht der Autor vor allem Dienste zur Ein- und Ausgabe und die Hauptroutinen des Monitor- und Editorprogrammes. - Um dem Benutzer nun eine einfache Richtschnur zu geben, welche Unterprogramme mit JSL aufzurufen sind, beginnen deren Namen einheitlich mit einem Unterstreichungsstrich, so z.B. _GETKEY oder _ANYKEY.

Man muß sich hinsichtlich der Mitbenutzung von Unterprogrammen des Betriebssystems (ob nun mit oder ohne JSR und JSL) auch eine Veränderung vor Augen halten, die in den letzten Jahren eingetreten ist: Früher waren die RAM-Speicher klein, so daß man durch die Mitbenutzung von Routinen etwas Platz sparen konnte. So aufgebaute Programme waren jedoch empfindlich gegen Veränderungen im Festwertspeicher, die man anderswo bezog, von der Dokumentationslage einmal abgesehen. Heute hat fast jeder Anwender die Möglichkeit, Veränderungen nach Wunsch mit einem EPROMmer selber vorzunehmen. Gleichwohl hat sich inzwischen ein Programmierstil als empfehlenswert erwiesen, in dem man die meisten benötigten Dienstleistungen selber in das Quellprogramm schreibt und nur wenige Dinge aus der Firmware mit dem Aufruf absoluter Adressen heranzieht. Solche globalen Routinen bezeichnet man auch als das BIOS, das BASIC Input Output System.

Da Softwarepakete niemals schon ganz fertig sind, so daß sich auch die Dokumentation ändern müßte, wurden im Betriebsprogramm des mc-65816 Computers etwa alle 50 Zeilen die Label der nachfolgenden Routinen als Byte-String in den Festwertspeicher geschrieben, wo man sie mit dem Memorybefehl bequem erkennen kann. Auch nach geringfügigen möglichen Verschiebungen dokumentieren die EPROMs sich damit selbst.

**Die Soft-Vektoren**
Bei früheren Computern, z.B. AIM 65 oder CBM 8032, liefen die Leistungen des Systems fast vollständig im Festwertspeicher ab, ohne einmal indirekt über das RAM zu springen, wo man Vektoren ja bekanntlich abändern kann. Das Betriebssystem des mc-65816 bietet hier große Beweglichkeit durch folgende RAM-Vektoren und ihre Adressen:

```
Vektoren/Sprungbefehle, Opcode $5C, im Native Mode:
  $200  COPLINKN   Sprung für COP-Befehl
  $204  BRKLINKN   dito für BRK-Befehl
```

$208 ABRTLINKN dito für Abort-Signal (Adreßfehler im System)
$20C NMILINKN dito für NMI-Signal
$210 IRQLINKN dito für IRQ-Signal

Vektoren/Sprungbefehle, Opcode $5C, im Emulations-Modus:
$214 COPLINKE Sprung für COP-Befehl
$218 ABRTLINKE für Abort-Signal
$21C NMILINKE für NMI-Signal
$220 IRQLINKE für IRQ-Signal

Verschiedene Vektoren und Befehle:
$224 MVN-Routine, abgeschlossen mit RTL, Block Move
$228 MVP-Routine, abgeschlossen mit RTL, Block Move
$22C MOLINK, Vektor für die Befehlsentschlüsselung des Monitors
$22F EDLINK, Vektor für die Befehlsentschlüsselung Editor
$232 DILINK, Ausgabevektor, Weitersprung zu OUTPUT
$235 PRINTLINK, Ausgabe Drucker, ggfs. Zeichenumsetzung
$238 IECOUTLINK, Ausgabe auf den IEEE-Bus
$23B IECINLINK, Eingabe vom IEEE-Bus
$23E USERLINKO, Ausgabe des Users
$241 USERLINKI, Eingabe des Users

Zu den Ausnahmevektoren ist zu bemerken, daß sie als Sprungbefehle zu auffangenden Routinen im Festwertspeicher führen. Die Befehle MVN und MVP wurden als vollständige mit RTL abgeschlossene Routinen ins RAM gelegt, weil solche Befehle im Festwertspeicher hinsichtlich Quell- und Zielbank zu starr sind. Per Programm muß man also nur noch z.B. nach $225 die Zielbank und nach $226 die Quellbank als Byte eintragen und dann aufrufen JSL $224.

MOLINK und EDLINK sind indirekte Zeiger. Sie werden mit JML (...) angesprungen, noch ehe irgendeine Befehlstaste im Monitor oder Editor entschlüsselt wurde. Damit ist man in der Lage, eine vollkommen neue Befehlsebene aufzubauen. Auch die Vektoren IECOUTLINK und IECINLINK werden vor Aktionen am IEEE-Bus mit JML übersprungen, so daß man auch hier andere Aktionen einflechten kann, zumal in den Zellen $2BD...$2C0 Kopien der Steuerung aufbewahrt werden, die an den Ports der VIA und ihren Richtungsregistern anzulegen ist.

**Stack, Direct Page und Common Area**

Obwohl man den Stack beliebig in der Bank 0 disponieren kann, z.B. an die höchsten freien Adressen, wurde er wie bei 6502 üblich bei $1FF belassen. Bei Bedarf von viel Stack kann man natürlich auch anders verfügen. Die Speicherseite 1 bleibt damit reine Stack-Seite.

Die Speicherseite 0 ist bei allen 6502-Computern zwischen Systemprogrammierern und Anwendern umkämpftes Gebiet gewesen, weil sich hier die besonderen Adressierungsarten unterbringen lassen. Der 65816 läßt es zu, die Direct Page überall in die Bank 1 zu legen. Damit tritt eine Entzerrung ein. So hatte der Autor denn keine Hemmungen, sogar den allgemeinen Eingabepuffer ab Adresse 3 in diese Page zu legen. In $00 ist der aktuelle Zeiger in diesen Puffer, in $01 die erreichte Stringlänge und in $02 das letzte eingetippte Zeichen. - Das Betriebssystem braucht viele Zeiger und auch Variablen für die Bewertung, die jetzt auch in 24 Bit angelegt werden müssen. Sie erstrecken sich von $53 bis einschließlich $84. Der Bereich bis hierhin wird als "common" (gemeinsam) bezeichnet. Die Grunddienste der Tastatureingabe, der Bewertung und der temporären Registerablage sowie der IEEE-Bus funktionieren immer in diesen ersten 133 Bytes relativ zur jeweiligen Lage der Direct Page.

So ist es z.B. möglich, dem Assembler die Direct Page ab absoluter Adresse $300 zur Verfügung zu stellen, wo er auf allgemeine Systemleistungen

zurückgreift und zudem eigene Variablen benutzt. In die Page drei muß für ihn nur der Anfangspointer des Editors übertragen werden. Ähnlich kann man für Anwenderprogramme disponieren. - Der im System enthaltene Editor und der Disassembler haben als Teilprogramme ihre Variablen in den absoluten Adressen ab $85...$A8, wo sie vom Assembler unberührt bleiben. Darüber ist bis $FF alles frei. Man kann auch so sagen: Wenn ein Anwenderprogramm die Grunddienste benutzt, dann hat es noch immer knapp 1/2 Direct Page für eigene Variablen frei, wo immer die Direct Page liegt.

**Speicherdisposition**

Page 4 wird vom Editor nur bei Such- und Ersetzungsfunktionen benutzt. Für die unterschiedlichen Betriebweisen beim Anwender kann man die Verfügbarkeit des Speichers wie folgt umreißen: Die Zero Page und die Page 1 (Stack) werden vom Betriebsystem benutzt. In alleinstehenden Systemen kann man je nach Nutzung der Systemroutinen ggfs. mehr als die Hälfte der Zero Page in Anspruch nehmen. Page 2 ist fast vollständig bis $C1 mit Vektoren und Systemvariablen belegt. Page 3 und Page 4 werden in Entwicklungssystemen vom Assembler und Editor benutzt. In einer solchen Umgebung wird man Anwenderprogramme oder einen Textspeicher ab Adresse $500 beginnen lassen. - In der Summe dürften damit in der Softwareplanung ausreichend Freiheiten für den Anwender offen gelassen worden sein..

```
0096                            ;** common-variablenbereich betriebssystem ***
0097 000000                     *=ramlow            variablen in der zero page
0098                            ;der eingabepuffer inbuff soll immer am beginn
0099                            ;der aktuellen direkt page/zero page liegen
0100                            ;manche variablen werden daher im jeweiligen um
0100                            feld
0101                            ;doppelt gefuehrt, jedoch nicht ineinander kopi
0101                            ert
0102                            ;der common-bereich erstreckt sich bis dp link
0104 000000          inbufp     *=*+1               zeiger in den eingabepuffer
0105 000001          stringlen  *=*+1               stringlaenge der eingabe
0106 000002          lastchar   *=*+1               zuletzt eingegebenes zeichen
                                                    gemerkt
0107 000003          inbuff     *=*+80              allg. eingabepuffer getstring
                                                    /assembler
0108 000053          inkanal    *=*+1               aktivierter eingabekanal (aus
                                                    ser keyboard)
0109 000054          outkanal   *=*+1               aktivierter ausgabekanal (aus
                                                    ser monitor)
0110 000055          tempkanal  *=*+1               vorgemerkter kanal nach jsl _
                                                    wohin?
0112                            ;********** ram fuer die systemverwaltung ****
0113 000056          dirpagreg  *=*+2               vorgabe direct page fuer _swi
                                                    t_dp routine
0114 000058          tempa      *=*+2               temporary accu
0115 00005a          tempx      *=*+2               temporary x
0116 00005c          tempy      *=*+2               temporary y
0117 00005e          temp       *=*+3               temporary allgemein
0118 000061          sprung     *=*+3               pointer fuer indir sprung
0119 000064          jump       =sprung             gleichsetzung mit sprung
0120 000064          umgebung   *=*+1               code des teilprogrammes, m=mo
                                                    nitor, e=editor
0122 000065          adres_von  *=*+3               low/high/bank, adresse 'von'
0123 000068          adres_bis  *=*+3               adresse 'bis'
0125 00006b          outlen                         ;gleichsetzung mit zbreit
0126 00006b          zbreit     *=*+1               zeilenbreite der ausgabe, nor
                                                    mal 80
```

```
0128 00006c          beflen    *=*+1              befehlslaenge disassembler/as
                                                  sembler
0130                           ;variablen fuer die bewertung
0131 00006d          iexpr     *=*+3              endwert einer bewertunglow/hi
                                                  gh/bank
0132 000070          zw_wert   *=*+3              feld fuer bewertungen
0133 000073          zexpr     =zw_wert           zwischenwert des ausdruckes
0134 000073          ziffer    *=*+1              zwischenablage b. zahlenwandl
                                                  ung
0135 000074          count     *=*+1              allg zaehler/ablage
0136 000075          carry     *=*+1              zwischenablage f. uebertraege
0137 000076          basis     *=*+1              zahlenbasis f. horner
0138 000077          trueflag  *=*+1              =1 b. erfolgreicher bewertung
0139 000078          truefl    =trueflag          true-flag bei bewertungen: =1
                                                  , wenn erfolgreich
0140 000078          keyflag   *=*+1              >$80: kein cr-echo bei getstr
                                                  ing
0142                           ;haupt-pointer des textsystems/editors
0143 000079          textpntr  *=*+3              allgemeine basisadresse fuer
                                                  textausgaben
0144 00007c          nowln     *=*+3              low/high/bank u.a. fuer _line
                                                  out geeignet
0145 00007f          edian     *=*+3              untere grenzadresse 1. textsp
                                                  eicher
0146                           ;zeiger-variablen monitor/assembler, common
0147 000082          parspt    *=*+3              pointer auf tabellen versch.
                                                  art
0148 000085          mesptr    =parspt            zeiger auf ausgabetexte
0149 000085          holadr    =parspt            temp. fuer adresseneinholung
0151                           ;****  ende des common-bereiches
0152 000085          dp_link                      beginn der dedizierten direkt
                                                   pages
```

**Fallstricke in der Programmierung**

Programm- und Datenbereich

Das Betriebssystem und der optionale Assembler bewegen sich im bei $FF0000 dekodierten EPROM. Wie schon dargelegt, verläßt man den Code dieses Festwertspeichers nicht mit Befehlen wie JMP Label oder JSR Label. Sprünge des Programmes in andere Banken können erfolgen mit JMP Label.L (langer Sprungbefehl mit Bankangabe), JSL Label (Unterprogrammaufruf in beliebiger Bank) und JML (Pointer mit 3 Bytes in Bank 0). Mit JML kann über die Bankadresse im Pointer ein Weitersprung in eine beliebige Bank erfolgen. Bei allen drei Befehlen (Opcodes $5C, $22 un $DC) wird der Programmzähler in seinen Bits 16...23 auf die Zielbank gesetzt, in der das Programm jetzt weiterläuft, bis ein ähnlich verändernder Befehl, ein RTL (Rückkehr von einem lang aufgerufenen Unterprogramm) oder eine Ausnahmebedingung eintritt. Der Programmierer bedenkt diese Dinge fast automatisch, so daß hier kaum Fehlermöglichkeiten entstehen, zumal man nicht so oft zwischen Programmbänken hin- und herspringt.

Etwas hinterlistig wird es, wenn wir Parameter ansprechen wollen, die im Speicher der aktuellen Programmbank stehen. Dort stehen z.B. Texte zur interaktiven Benutzerführung, Tabellen mit erlaubten Befehlstasten und Vergleichsparameter der verschiedendsten Art. Beim 6502 waren wir immer in einer Bank. Wenn wir dort z.B. sagten CMP $FF00,X, so hatten wir im allgemeinen einen Parameter im Festwertspeicher angesprochen. Nicht so beim 65816: Der vorstehende Befehl spricht immer einen Operanden in der aktuellen DATENBANK an. Und diese ist die Bank 0 laut Reset des 65816, wenn man sie nicht durch eine Befehlsfolge verändert hat.

Daten und Parameter in anderen Bänken als 0 können wir nur durch Befehle mit einer "langen" Adressierungsart ansprechen, etwa in den Formulierungen LDA Label.L, LDA LABEL,LX (Nach-Indizierung mit Register X), LDA (Label,L) (Pointer mit 3 Bytes in der Direct Page) und LDA (Label,L),Y (Pointer 3 Bytes, Nach-Indizierung mit Y). Eine weitere Möglichkeit besteht darin, das Datenbankregister vorübergehend umzuschalten, etwa mit der Befehlsfolge PHB (alte Datenbank auf dem Stack sichern), LDA #neue Datenbank, PHA (1 Byte) und schließlich PLB (Hineinziehen des Direktwertes über den Stack in das Datenbankregister). Nach der Aktion muß man nur noch PLB programmieren, um die alte Datenbank zurückzurufen.

Es ist also eine Frage der persönlichen Aufmerksamkeit, daß man die langen Adressierungen (oder eine Datenbankumschaltung) wählt, wenn man Parameter außerhalb der Bank 0 ansprechen möchte. Bei letzterem Verfahren mag es aber vorkommen, daß soeben noch zur Verfügung stehende Parameter, z.B. aus einer Eingabezeile, nicht mehr in der neuen Bank stehen, so daß man von hier "zurückbaggern" muß. - Der Teufel sitzt aber wie immer im Detail: Man überträgt bewährte Quelltext-Module auf die neue CPU, ohne jede einzelne Zeile auf ihre Tauglichkeit in der neuen Umgebung anzusehen und erlebt dann die Ansprache ungeeigneter Operanden. Der Autor darf freimütig berichten, daß er schon in solche Fallen geriet und daß die Befreiung aus ihnen Zeit kostete.

**Besonderheiten bei Push und Pull**

Wenn man im 6502-Modus der CPU ist (Emulation), dann wirken auch die Befehle Push on Stack und Pull from Stack auf jeweils 1 Byte. Im ureigenen Modus (native) hängt es von der für den Akku und die Indexregister eingestellen Operandenbreite ab, ob ein Byte oder ob ein Wort logisch richtig auf den Stack transportiert oder von dort geholt wird. Es gibt aber Stackbefehle, die auch bei sonst vorherrschender Operandenbreite von 16 Bit immer nur 1 Byte im Verkehr mit dem Stack transportieren. Es sind dieses PHP und PLP für das Statusregister, PHB und PLB für das Datenbankregister und PHK für das Stacken des Programmbank-Registers (zu diesem Befehl gibt es kein Pendant mit einem Rücktransport).

Weiterhin: Bei einem Unterprogrammaufruf mit JSL werden 3 Bytes der Rückkehradresse-1 auf den Stack gebracht, also auch die Programmbank. - Gemischt wird es dann wieder bei den Programmausnahmen (Interrupte). Im 6502-Modus (Emulation) werden wie üblich 3 Bytes mit der Rückkehradresse-1 und dem alten Status auf den Stack gebracht (im BRK-Bit auf dem Stack erkennt man, ob ein BRK-Befehl zur Ausnahme führte oder ein IRQ). - Im ureigenen Modus wird 1 Byte mehr auf den Stack transportiert, nämlich zu allererst das Programmbank-Register der unterbrochenen Routine, dann die Rückkehradresse-1 und schließlich der alte Status. Diese logisch richtige Besonderheit muß man bedenken, wenn man einen Interrupt auffängt oder wenn man im Monitor-Programm den Go-Befehl implementiert, bei dem ein Programm mit vorbesetzen Registern gestartet wird. Die Lage wird dadurch erleichtert, daß es für beide Modi der CPU getrennte Ausnahmevektoren gibt, so daß man weiß, woher man kommt. Wir gehen noch darauf ein.

**"Lang" wirkende Entscheidungen**

Der Leser wird aus früheren Veröffentlichungen erinnern, daß der Autor sich für das Verzweigungsprinzip ON...GOTO mit klaren Entscheidungen und Strukturen einsetzt. In Assemblersprache hatten wir dafür die neueren Befehle JMP (Label,X). Jetzt ist beim 65816 der Befehl JSR (Label,X) hinzugekommen. Beide erlauben eine Sprungleiste, die in Abhängigkeit vom Register X benutzt wird, das gewissermaßen das "ON" bildet. Mit beiden Befehlen kann man die aktuelle Programmbank nicht verlassen, und es gibt keinen weiteren Befehl, der das mit einer indirekten und indizierten Adressierung zulassen würde. Der 68000 von Motorola ist da seit Jahren beweglicher. Bei ihm gibt es z.B. die Adressierung

JMP d(Adreßregister,Indexregister). Das Adreßregister weist samt einem möglichen Offset d auf die Basis einer Sprungleiste. Ein zweites Register der CPU dient als Index und positioniert innerhalb der Sprungleiste auf das richtige "ON". Entscheidungen fallen hier also leichter.

Gleichwohl gibt es einen Trick mit dem Befehl PHK (Push Programmbank-Register), der im Monitorprogramm in Zeile 981 benutzt wurde:

```
0932                            ;***** hauptverarbeitungsschleife des monitors
0933 ff04af 4d41494e            .byt 'mainl'
0934 ff04b4 18        mainl     clc                 native mode
0935 ff04b5 fb                  xce
0936 ff04b6 c230                rep #$30            alle register in 8 bit
0937 ff04b8                     .xl6
0938 ff04b8 58                  cli                 clear any interrupt
0939 ff04b9 d8                  cld
0940 ff04ba a2ff01    main      ldx #$1ff
0941 ff04bd 9a                  txs                 init stack to $1ff
0942 ff04be                     .r08
0943 ff04be e230                sep #$30
0944 ff04c0 a900                lda #00
0945 ff04c2 8556                sta dirpagreg
0946 ff04c4 8557                sta dirpagreg+1
0947 ff04c6 6453                stz inkanal         systemeingabe setzen
0948 ff04c8 22ba03ff            jsl _swit_dp        direct page immer mormalisier
                                                    en
0950 ff04cc a94d                lda #'m'            befehlsebene verankern
0951 ff04ce 8564                sta umgebung
0952 ff04d0 22b402ff            jsl _crlf           _output return
0953 ff04d4 a93c                lda #'<'            klammere geholten befehl in <
                                                     >
0954 ff04d6 225d02ff            jsl _output
0955 ff04da 229400ff            jsl _getkeyg        hole zeichen
0956 ff04de dc2c02              jml (molink)        softvektor f. erweiterung
0957 ff04e1 4d4f4e43            .byt 'moncmd?'
0958 ff04e8 297f      moncmd?   and #$7f
0959 ff04ea 225d02ff  maing     jsl _output
0960 ff04ee 48                  pha                 rette kommandozeichen
0961 ff04ef a93e                lda #'>
0962 ff04f1 225d02ff            jsl _output
0963 ff04f5 68                  pla                 hole zurueck
0964 ff04f6 a21f                ldx #monbef         count of commands
0965 ff04f8 dfaa08ff  mcm2      cmp comb,lx
0966 ff04fc f00b                beq main3
0967 ff04fe ca                  dex
0968 ff04ff 10f7                bpl mcm2
0970                            ;unbekannter befehl
0971 ff0501 a93f                lda #'?'
0972 ff0503 225d02ff            jsl _output
0973 ff0507 80ab                bra mainl
 0975                            ;have valid command
 0976 ff0509 8596      main3     sta monbefehl       merke die taste
 0977 ff050b 8a                  txa
 0978 ff050c 0a                  asl a               a 2 bytes (addr)
 0979 ff050d aa                  tax
 0981 ff050e 4b                  phk                 program bank register to stac
                                                     k for rtl
 0982 ff050f fcd008              jsr (moncom,x)      fuehre monitor-routine aus
 0983 ff0512 4cb404              jmp mainl           stelle emulation wieder her
```

Wenn ein Unterprogramm aufgerufen werden soll, das mit RTL endet, dann kann es statt mit JSL auch mit JSR (Label,X) aufgerufen werden, wenn zuvor das Programmbank-Register mit PHK auf den Stack gebracht wurde. Wesentlich ist nur, daß RTL später die 3 richtigen Bytes auf dem Stack findet.

**Die Befehle BRK und COP**

In den Datenblättern finden wir BRK und COP als 2-Byte Befehle ausgewiesen. Bezüglich des BRK ist das sicher ein neuer Hinweis, und wir fragen, was das bedeutet. Die Antwort lautet eigentlich nur ganz einfach, daß der durch die Programmunterbrechnung auf den Stack gerettete Programmzählerstand um 2 größer ist als die Adresse des BRK-Befehles. Und das gilt ebenso für die Programmunterbrechnung durch den Befehl COP. Hinter dieser Konstruktion steht eine von den Designern des Chips offensichtlich nicht konsequent verfolgte Überlegung:

Das auf den Opcode (00 für BRK) folgende 2. Byte wird "Signatur" genannt. Es war wohl geplant, um den beiden Befehlen einen Parameter mitgeben zu können, mit dem man nach einem BRK oder Coprozessorbefehl sofort eine logische Verzweigung gemäß der Signatur vornimmt. Diese Überlegung ähnelt der Einrichtung der numerierten Traps (TRAP #) beim 68000. Beim Chip 65816 wurde daraus jedoch eine Fehlkontruktion: Der auf dem Stack abgelegte Programmzählerstand zeigt auf die Adresse hinter der Signatur. Es bedarf nun eines ziemlichen Aufwandes, um das Byte der Signatur in ein Register zu laden, um es auswerten zu können. Zunächst muß man zum Zählerstand auf dem Stack $FFFFFF hinzuaddieren und das Ergebnis in einen Pointer mit 3 Bytes bringen, damit er auf die Adresse der Signatur weist. Dann muß man die Signatur indirekt aus dem Pointer laden. Erst dann kann man abhängig von der Signatur eine Entscheidung fällen.

Der Autor hat das Arbeiten mit so numerierten Breaks schon einmal ausprogrammiert und war wegen des doch erheblichen Aufwandes, der ja auch einen Zeitbedarf hat, damit nicht zufrieden.

**Break und Go**

Beim Ausprobieren eines Programmes setzt man an geeigneten Punkten den BRK-Befehl ein, um an diesen Stellen eine Registeranzeige zur Kontrolle zu erhalten. Andererseits möchte man einen Programmteil an einem bestimmten Punkt mit definiert vorgeladenen Registern starten, um Auswirkungen beobachten zu können. In der neuen Umgebung des 65816 kommen wegen seiner zahlreichen möglichen Betriebsweisen z.T. recht komplizierte Einflußgrößen gegenüber dem 6502 hinzu, wenn man einen BRK-Befehl auffängt oder ein Programm definiert startet. Es ist am besten, wenn man dazu den nebenstehenden Ausschnitt aus der Assembler-Liste studiert.

Im 6502-Emulationsmodus benutzen der BRK-Befehl und der von außen kommende Interrupt IRQ denselben Vektor, und man kann nur am Status auf dem Stack sehen, ob des Break-Bit gesetzt ist. Von ihm abhängig teilt man in die BRK- und in die IRQ-Hantierung auf. Beide Unterbrechungen werden auf das Label IRQ_BRK in Zeile 1231 zugeleitet. Wenn es ein IRQ war, dann wird in Zeile 1235 zu dessen Hantierung verzweigt. Ein BRK geht zum Label BRK_ROUE über, wo zunächst der Modus native eingestellt wird. Durch den Befehl XCE in Zeile 1239 erhalten wir aber zugleich im Carry-Bit (=1) einen Merker, welcher Modus vorher herrschte, so daß wir jetzt auf die allgemeine BRK-Behandlung bei BRK_ROUO überleiten können.

Ein BRK im Modus native wird nach BRK-ROU in Zeile 1243 vektoriert. Durch die 4 ersten Befehle gewinnen wir im Carry-Bit (=0) ebenfalls einen Merker zum Modus. Wenn wir jetzt die erreichten Registerstände in der Direct Page

```
1231 ff0716 855e      irq_brk   sta temp          irq oder brk,emulation mode
1232 ff0718 68                  pla               status
1233 ff0719 48                  pha
1234 ff071a 8910                bit #%00010000    break-flag gesetzt?
1235 ff071c f0eb                beq irq_roul      nein
1236 ff071e a55e                lda temp
1238 ff0720 18        brk_roue  clc
1239 ff0721 fb                  xce               weiter im native modus, carry
                                                   zeigt herkunft
1240                            ;rep #$30 mit carrry set
1241 ff0722 8004                bra brk_rou0
1243 ff0724           brk_rou                     entry native mode
1244 ff0724 fb                  xce               gewinne modus
1245 ff0725 08                  php               carry clear = modus der cpu
1246 ff0726 fb                  xce               umschaltung rueckgaengig
1247 ff0727 28                  plp
1248 ff0728           brk_rou0                    gemeinsam native und emulatio
                                                   n
1249 ff0728                     .r16              register 16 bit
1250 ff0728 c230                rep #$30          16 bit modus
1251 ff072a 48                  pha               akku retten, native mode
1253 ff072b 7b                  tdc               dito direct page register
1254 ff072c 48                  pha
1256 ff072d a90000              lda #00           direct page zur sicherheit be
                                                   i 0000
1257 ff0730 5b                  tcd
1258 ff0731 68                  pla
1259 ff0732 858a                sta sav_dp        alte direct page
1260 ff0734 68                  pla               alten akku
1261 ff0735 8594                sta sav_a
1262 ff0737 8692                stx sav_x         indexregister
1263 ff0739 8490                sty sav_y
1264 ff073b                     .r08              register wieder 8 bit
1265 ff073b e230                sep #$30
1266 ff073d 8b                  phb               alte datenbank merken
1267 ff073e a900                lda #0
1268 ff0740 48                  pha               datenbank jetzt 00 setzen
1269 ff0741 ab                  plb
1271 ff0742 68                  pla
1272 ff0743 858c                sta sav_db
1273 ff0745 68                  pla               alten status
1274 ff0746 8585                sta sav_ps
1275 ff0748 08                  php               carry-status mit dem modus
1276 ff0749 68                  pla
1277 ff074a 2901                and #1            maskiere carry als merker mod
                                                   us
1278 ff074c 8586                sta sav_ps+1      zeigt emulationsmodus
1279 ff074e 68                  pla               pc_low
1280 ff074f 8587                sta sav_pc
1281 ff0751 68                  pla               pc_high
1282 ff0752 8588                sta sav_pc+1
1283 ff0754 a586                lda sav_ps+1      welcher modus war?
1284 ff0756 f004                beq brk_rou01     native, noch bank holen
1285 ff0758 6489                stz sav_pc+2
1286 ff075a 8003                bra brk_roul
1287 ff075c 68        brk_rou01 pla
1288 ff075d 8589                sta sav_pc+2      programmbank
```

```
1289 ff075f          brk_roul                    fuer stackpointer:
1290 ff075f c230                 rep #$30
1291 ff0761 ba                   tsx             hole stackpointer ins registe
                                                 r
1292 ff0762 868e                 stx sav_sp      stack ist nun ausgerauemt
1293 ff0764 e230                 sep #$30
1294 ff0766 22b402ff             jsl _crlf
1295 ff076a a000                 ldy #0
1296 ff076c a270                 ldx #m8-mesbas  break anzeigen
1297 ff076e 221203ff             jsl _mesout
1298 ff0772 221c05ff             jsl _regdis     registerleiste anzeigen
1299 ff0776 4cba04               jmp main        zum monitor gehen
```

abspeichern wollen, um sie später anzuzeigen, so müssen wir uns vor Augen halten, daß ein Anwenderprogramm durchaus die Lage der Direct Page verändert haben kann. In den Zeilen 1257 und 1258 wird sie daher definiert auf $0000 gelegt, damit nun die Register abgespeichert werden können, und zwar in der Reihenfolge alte Direct Page, Akku, X, Y, alte Datenbank und der vom BRK-Befehl auf den Stack gebrachte Status. Bis hierhin ist das Carry-Bit, das über den früheren Modus Auskunft gibt, nicht verändert worden. Es wird in den Zeilen ab 1275 isoliert und abgespeichert. Schließlich folgen der vom Stack entnommene Programmzähler und der Stackpointer vor dem BRK in die Abspeicherung. Nun können alle Register angezeigt werden (JSL _REGDIS). - Man sieht, daß das Retten der Register etwas aufwendig ist, wenn man vorhandene Information nicht zerstören will.

**Der Go-Befehl** mit der Übergabe vorbesetzter Register verlangt einen ähnlichen Aufwand. Dabei ist klar, daß der aus einem Break gewonnene oder sonstwie per Monitorprogramm gesetzte Status erst zuallerletzt übernommen werden darf, damit er nicht durch die übrigen Vorbereitungen zum Go-Befehl verändert wird.

Beim Label _GO_BEF in Zeile 1055 sind wir im Modus native. Mit REP #$30 werden alle Register auf die Breite von 16 Bit gebracht. Nun wird der Stackpointer so übertragen, wie er am Ende der Prozedur sein soll, nachdem alle noch folgenden Push- und Pullbefehle erledigt sind. Nun kommt der geplante Status auf den Stack (dort wird er als letztes Item dann entnommen), danach ein Status, in den das geplante Emulationsbit in die Bitposition des Carry-Flag hineingeodert wird. Es folgen die geplante Datenbank und die Lage der Direct Page. Beide dürfen jetzt noch nicht aktiv sein, denn es sind noch die jetzt nicht mehr benötigten Register Akku, X und Y vorzuladen.

Nun kann die Entnahme vom Stack erfolgen: Direct Page Register, Datenbank und ein Status mit dem Emulationsbit im Carry-Flag. Dieses wird mit XCE aktiviert, wobei natürlich etwas in das Carry kommt, was dort vielleicht nicht hingehört. Nun kann der geplante Status mitsamt dem geplanten Carry-Flag vom Stack entnommen werden. Der geplante Programmzähler steht in drei Zellen der Bank 0. Der Befehl JML (SAV_PC) überträgt ihn in die CPU. Weil JML immer über einen Pointer in Bank 0 indirekt springt, ist es gleichgültig, welche Datenbank wir geplant haben.

```
1054 ff05ba 5f474f5f             .byt '_go_bef'
1055 ff05c1          _go_bef                     ;programmstart mit vorbesetzt
                                                 en registern
1056                             ;wir sind native, 8 bit
1058 ff05c1 c230                 rep #$30
1059 ff05c3 a68e                 ldx sav_sp      stackpointer alt
1060 ff05c5 9a                   txs             uebertragen
1062                             ;einsprung, wenn rtl-routinen benutzt werden, r
1062                             eturn to main
1063 ff05c6 e230    _go_rtl      sep #$30        8 bit register
```

```
1064 ff05c8 a585          lda sav_ps          alter status davor
1065 ff05ca 48            pha
1066 ff05cb 29fe          and #$fe            jetzt das emulation bit dazu-
                                              odern
1067 ff05cd 0586          ora sav_ps+1
1068 ff05cf 48            pha                 auch auf den stack
1070 ff05d0 a58c          lda sav_db          alte datenbank
1071 ff05d2 48            pha
1073 ff05d3 c230          rep #$30            num 16 bit
1074 ff05d5 a68a          ldx sav_dp          alte direkt page
1075 ff05d7 da            phx
1076 ff05d8 a692          ldx sav_x           alte indexregister
1077 ff05da a490          ldy sav_y
1078 ff05dc a594          lda sav_a
1080 ff05de 2b            pld                 2 byte direct page register
1081 ff05df ab            plb                 1 byte, datenbank via stack r
                                              estituiert
1082 ff05e0 28            plp                 status mit emulation bit
1083 ff05e1 fb            xce                 modus herstellen
1084 ff05e2 28            plp                 sav_ps wieder herstellen
1085 ff05e3 dc8700        jml (sav_pc)        springt immer per bank 00
```

**Befehlsentschlüsselung, ON...GOSUB**

Im Abschnitt "Lang wirkende Entscheidungen" wurde bereits erwähnt, daß man mit dem Befehl JSR (Label,X) mit RTL endende Unterprogramme aufrufen kann, wenn man zuvor mit dem Befehl PHK das Programmbank-Register auf den Stack gebracht hat, das vom RTL gebraucht wird. In den nebenstehenden Zeilen 953...983 wird die Anwendung dieses Prinzips in der Haupt-Warteschleife nebst Entschlüsselung im Monitor gezeigt. In Zeile 955 wird ein Befehls-Tastendruck abgefordert, der in Groß- oder Kleinbuchstaben erfolgen darf. Eine Zeile später erfolgt vor jeglicher Dekodierung bereits der Sprung JML über einen Vektor im RAM, der auf Zeile 958 gerichtet ist, wo die normale Programmfortsetzung erfolgt, wenn der Anwender sie nicht vorher umgelenkt hat. Bei MCM2 erfolgt der Vergleich mit einer Tabelle erlaubter Befehlstasten. Wird eine solche erkannt, so geht es bei MAIN3 mit der Multiplikation von X mit 2 weiter. X weist jetzt relativ zur Basis einer Vektortabelle ausführender Routinen mit Namen MONCOM, so daß nach dem erwähnten PHK (Programmbank) JSR (MONCOM,X) zur Programmverzweigung gegeben werden kann.

Um jedoch die herrschende Programmbank mit einer ähnlichen Konstruktion wie JSR (Label,X) verlassen zu können, sind aufwendigere Programmierungen notwendig.

**Umschalten des Direct Page Registers**

Das Direct Page Register (2 Byte) wird im 16 Bit-Modus entweder mit TCD vom Akku C (mit den Hälften A und B) her umgeladen oder über den Stack mit PLD. Um nun eine Lösung zu finden, die die alte Betriebsweise der CPU automatisch wiederherstellt, wurde die Routine SWIT DP geschrieben (ab Zeile 0809). Sie rettet zunächst den alten "normalen" Status, besorgt sich dann per XCE den Modus der CPU im Carry Bit und rettet den so (ggfs. veränderten) Status ebenfalls. Dann kann auf den 16 Bit-Modus umgeschaltet werden. Der Akku wird aus den Zellen $56/57 (DIRPAGREG) geladen und in das Direct Page Register per TCD übertragen. Der Benutzter hat also vor dem Aufruf im DIRGPAGREG seine Disposition hinterlegen müssen. Hernach kann der Modus der CPU und der richtige Carry-Status wieder über den Stack zurückgewonnen werden. - Wenn es auch einfacher ginge: Diese Routine hat den Vorteil der allgemeinen Anwendbarkeit.

```
0809 ff03ba          _swit_dp                    ;umschalten direct page regis
                                                  ter
 0810 ff03ba                      .rl6            register 16 bit
 0811 ff03ba 08                   php             alten carry-status retten
 0812 ff03bb fb                   xce             alten modus herausholen
 0813 ff03bc 08                   php             retten
 0814 ff03bd 18                   clc             ;native mode fuer init direct
                                                   page
 0815 ff03be fb                   xce
 0816 ff03bf c230                 rep #$30        a=16, x=16
 0817 ff03c1 a556                 lda dirpagreg   direct page einrichten nach v
                                                  orgabe
 0818 ff03c3 5b                   tcd             ins direct page register
 0819 ff03c4                      .r08
 0820 ff03c4 28                   plp             alten modus
 0821 ff03c5 fb                   xce             wiederherstellen
 0822 ff03c6 28                   plp             alten carry-status
 0823 ff03c7 6b                   rtl
 0825 ff03c8 54454143             .byt 'teach'
 0826                             ;anlernen der acia
 0827 ff03cd 18        teach      clc
 0828 ff03ce fb                   xce             native
 0829 ff03cf                      .x16
 0830 ff03cf c210                 rep #$10
 0831 ff03d1 a90b                 lda #$0b
 0832 ff03d3 8f1200fd             sta acial_com,l command register, rapport lad
                                                  en
 0833 ff03d7 a20300    teach0     ldx #3
 0834 ff03da bf0e04ff  teach1     lda baudtable,lx
 0835 ff03de 8f1300fd             sta acialc,l
 0836 ff03e2 af1000fd  teach2     lda aciald,l    datenregister
 0837 ff03e6 af1100fd  teach20    lda acials,l    statusregister
 0838 ff03ea 2908                 and #8
 0839 ff03ec f0f8                 beq teach20
 0840 ff03ee af1000fd             lda aciald,l    datenregister
 0841 ff03f2 c90d                 cmp #cr
 0842 ff03f4 d00b                 bne teach3
 0843 ff03f6 af1100fd             lda acials,l    statusreg
 0844 ff03fa 2907                 and #7
 0845 ff03fc d003                 bne teach3
 0846 ff03fe e230                 sep #$30
 0847 ff0400 6b                   rtl
 0849 ff0401 da        teach3     phx
 0850 ff0402 a250c3               ldx #50000
 0851 ff0405 ca        teachloop  dex
 0852 ff0406 d0fd                 bne teachloop
 0853 ff0408 fa                   plx
 0854 ff0409 ca                   dex
 0855 ff040a 30cb                 bmi teach0
 0856 ff040c 80cc                 bra teach1
 0858 ff040e 181a1c1e baudtable   .byt $18,$1a,$1c,$1e 1200-9600 baud
 0859 ff0412                      .r08
```

**Initialisierung der Baudrate**

Die Erfahrung zeigt, daß die Inbetriebnahme der seriellen Schnittstelle RS232 immer wieder mit Überraschungen verbunden ist, die von der Art der elektrischen Signaldurchführung abhängen. Zugleich muß man sich aber vor Augen halten, daß die benutzten Terminals oder die als Terminals benutzten Computer

auf unterschiedliche Übertragungsgeschwindigkeiten eingestellt sind. Es wurde daher ein Lernprogramm TEACH ab Zeile 0827 implementiert, das die benutzte ACIA automatisch auf eine Baudrate zwischen 1200 und 9600 einstellt. Die Grundidee ist folgende: Das Terminal soll nach dem RESET dauernd ein CR senden. Die ACIA des Computers ist im Command Register mit dem beabsichtigten Modus (Rapport) vorgeladen worden und wird nun in einer Schleife ab Label TEACHO laufend mit einer von 4 Baudraten geladen. Danach wird im Datenregister gefragt, ob ein CR empfangen wurde. Wenn ja, dann steht die Baudrate fest und es kann weitergehen. Anderenfalls wird eine Warteschleife bei TEACHLOOP benutzt, ehe mit der nächsten Baudrate ein weiterer Versuch stattfindet. - Die praktische Erfahrung zeigt, daß die richtige Baudrate mit diesem Algorithmus augenblicklich gebootet wird.

**Das Interface zum IEEE 488-Bus**

In dieser Zeitschrift wurden schon im Heft 19 (Juni 1981) und nachfolgend Routinen und ein Interface für den Betrieb verschiedener Computer am GPIB-Bus veröffentlicht, besonders in Hinsicht auf den Betrieb der parallelen CBM-Floppys als Massenspeicher. Diese Anregungen waren Vorbild für viele gleichartige Implementierungen. Es lag daher nahe, den mc-65816 Computer auch mit einem solchen Interface in Hard- und Software zu versehen, damit er einen Massenspeicher haben und damit er als Steuercomputer mit dem Meßgerätebus zusammenwirken kann. Zwischenzeitlich gibt es nun Bausteine für den Anschluß, die eine wesentlich einfachere Verdrahtung zulassen. Dazu nebenstehend die Skizze des Interfaces mit den Bausteinen 75160 und 75161 von TI. Man sieht, daß die an den Ports der benutzten VIA 6522 angelegten Signale gradlinig über diese Bausteine weiter auf den Bus gelegt werden können. Nicht benutzt ist eine Steuerung des Remote Enable (REN) und (softwaremäßig) eine Auswertung der Interruptanforderung SRQ durch Geräte.

Die Vereinfachung der Hardware war zugleich auch Anlaß, die Software in noch übersichtlichere Module mit merkfähigen Namen zu gliedern, um die Verständlichkeit und die Übersicht zu befördern. Bezeichner wie IECINIT, NEUTRAL, SETINPUT und SETOUPUT oder SETPINHIGH und SETPINLOW dürften dabei für sich sprechen. Ferner auch z.B. DAVHIGH (zum Setzen) und NDACHIGH? (zum Abfragen). - Für die Signale am Port B als Steuerport wurden Namen gewählt, die dem IEEE-Bezeichnungen entprechen z.B. DAV=$80. Mit diesen Zuweisungen ist es dann möglich, die logischen Befehle des Schaltens und der Abfrage in einer sehr mnemonischen Form zu schreiben, z.B. LDA #DAV für das Hinzuodern, LDA #$FF-DAV für das maskierende Ausschalten am Port und BIT #DAV für die Abfrage des Signals. - Die Zeilen ab Nr. 1452 sind als physikalische Busroutinen zum Schalten von Signalzuständen anzusehen.

```
1400                        ;iecbus-routinen fuer mct-computer mit 65816
1401                        ;copyright 1986 by roland loehr, backup 4,2,86
1402                        ;paralleler bus, angeschlossen ueber system-via
1402                         6522
1403                        ;und treiber 75160 (daten) und 75161 (steuerbus
1403                        )
1405                        ; erklaerung der via-register
1406 ff0970      portb      =via              port
1407 ff0970      porta      =via+1
1408 ff0970      ddrb       =via+2            datenrichtung
1409 ff0970      ddra       =via+3
1410 ff0970      t1cl       =via+4            timer 1 counter low
1411 ff0970      t1ch       =via+5            counter high
1412 ff0970      acr        =via+$a           auxiliary control register
1413 ff0970      pcr        =via+$c           peripheral control register
1414 ff0970      ifr        =via+$d           interrupt flag register
```

```
1416                                ;signalbelegung: porta ist datenport fuer diol-
1416                                dio, bidirektional
1417                                ; pb0=nc
1418                                ; pb1=te-control, 0=lesen, 1=schreiben
1419                                ; pb2=dc-control, auf 0 gehalten
1420                                ; pb3=atn,        attention
1421                                ; pb4=eoi         end or identify
1422                                ; pb5=ndac
1423                                ; pb6=nrfd
1424                                ; pb7=dav
1425                                ; ren=remote enable, fest auf gnd geschaltet
1426                                ; ca2=srq in      nicht programmiert, interrupt
1426                                faehig
1427                                ; cb2=ifc         interface clear, per taste 'z
1427                                ', monitor
1428                                ; pe=pull up enable, mit 1k nach +5v geschaltet
1428
1430                                ; operatoren des programms fuer bussignale an d
1430                                en pins
1431 ff0970          te_contr     =2
1432 ff0970          dc_contr     =4
1433 ff0970          atn          =8
1434 ff0970          eoi          =$10
1435 ff0970          ndac         =$20
1436 ff0970          nrfd         =$40
1437 ff0970          dav          =$80
1438 ff0970          oneshot      =0                  timer 1 oneshot modus fuer ac
                                                     r
1439 ff0970          outport      =dav+eoi+atn+dc_contr+te_contrzu ausgaengen in d
                                                     drb
1440 ff0970          inport       =ndac+nrfd+atn+dc_contr+te_contreingabeport in d
                                                     drb
1442                                ; logische operatoren fuer datenverkehr
1443 ff0970          lisn         =$20                devices-nummer hinzu-odern
1444 ff0970          tlk          =$40                dito
1445 ff0970          open         =$f0                dito
1446 ff0970          close        =$e0                dito
1447 ff0970          unlisn       =$3f
1448 ff0970          untlk        =$5f
1450 ff0970                       .r08                mit registern in 8 bit
1451                                ; unterprogramme zum busverkehr
1452 ff0970 4e455554              .byt 'neutral'
1453 ff0977 a9ff     neutral      lda #$ff            datenport auf ausgang setzen
1454 ff0979 8dc002                sta ddracopy
1455 ff097c 8dbe02                sta portacopy       kopie f. anpassung seriell
1456 ff097f 2cc102                bit inhibit         paraleller iec-bus?
1457 ff0982 3008                  bmi neutrall        nein
1458 ff0984 8f2300fd              sta ddra,1          und neutral senden
1459 ff0988 8f2100fd              sta porta,1         neutral senden
1460 ff098c 60       neutrall     rts
1461 ff098d 207709   setoutput    jsr neutral         port a ausgang, neutral
1462 ff0990 a99e                  lda #outport        port b zum ausgang machen
1463                                ;ndac und nrfd bleiben eingang
1464 ff0992 2cc102                bit inhibit
1465 ff0995 3004                  bmi setoutp1
1466 ff0997 8f2200fd              sta ddrb,1
1467 ff099b 8dbf02   setoutp1     sta ddrbcopy
1468 ff099e a9fa                  lda #%11111010      dav, eoi, atn te_contr high
1469 ff09a0 2cc102                bit inhibit
```

```
1470 ff09a3 3004              bmi setoutp2       port nicht veraendern
1471 ff09a5 8f2000fd          sta portb,1
1472 ff09a9 8dbd02   setoutp2 sta portbcopy
1473 ff09ac 60                rts
1475 ff09ad a96e     setinput lda #inport        ddrb auf eingabe setzen
1476 ff09af 2cc102            bit inhibit
1477 ff09b2 3004              bmi setinput1
1478 ff09b4 8f2200fd          sta ddrb,1
1479 ff09b8 8dbf02   setinput1 sta ddrbcopy
1480                           ;dav und eoi bleiben eingaenge
1481 ff09bb a9f8              lda #%11111000     ndac + nrfd high senden
1482 ff09bd 2cc102            bit inhibit
1483 ff09c0 3004              bmi setinput2
1484 ff09c2 8f2000fd          sta portb,1
1485 ff09c6 8dbd02   setinput2 sta portbcopy
1486 ff09c9 a900              lda #0
1487 ff09cb 2cc102            bit inhibit
1488 ff09ce 3004              bmi setinput3
1489 ff09d0 8f2300fd          sta ddra,1         datenrichtung eingang
1490 ff09d4 8dc002   setinput3 sta ddracopy
1491 ff09d7 60                rts
1493 ff09d8 207709   iecinit  jsr neutral        interface initialisieren
1494 ff09db 208d09            jsr setoutput
1495 ff09de 60                rts
1497 ff09df a900     _timeout? lda #oneshot      menge durchgaenge in tempa vc
                                                 rladen !!
1498 ff09e1 8f2a00fd          sta acr,1          one shot mode
1499 ff09e5 a9ff     timeout0 lda #$ff           16 ms bei 4 mhz je oneshot
1500 ff09e7 8f2400fd          sta t1cl,1         zugleich ins latch
1501 ff09eb 8f2500fd          sta t1ch,1         start timer
1502 ff09ef af2d00fd timeout1 lda ifr,1          interrupt-flag t1?
1503 ff09f3 8940              bit #$40
1504 ff09f5 f0f8              beq timeout1
1505 ff09f7 c658              dec tempa
1506 ff09f9 d0ea              bne timeout0       restart
1507 ff09fb 6b                rtl
1508 ff09fc 41544e4c          .byt 'atnlow'
1509 ff0a02 a9f7     atnlow   lda #$ff-atn       maskieren
1510 ff0a04 200e0a            jsr setpinlow      setzen
1511 ff0a07 a905              lda #5             ca. 38 mikrosek warten
1512 ff0a09 221d0bff          jsl _wait
1513 ff0a0d 60                rts
1514 ff0a0e 2f2000fd setpinlow and portb,1       atn low setzen
1515 ff0a12 2cc102            bit inhibit
1516 ff0a15 3004              bmi setpinlo1
1517 ff0a17 8f2000fd          sta portb,1
1518 ff0a1b 8dbd02   setpinlo1 sta portbcopy
1519 ff0a1e 60                rts
1520 ff0a1f a908     atnhigh  lda #atn           atn high setzen
1521 ff0a21 202b0a            jsr setpinhig      signal setzen
1522 ff0a24 a989              lda #137
1523 ff0a26 221d0bff          jsl _wait          ca. 960 mikrosek. warten
1524 ff0a2a 60                rts
1525 ff0a2b 0f2000fd setpinhig ora portb,1
1526 ff0a2f 2cc102            bit inhibit
1527 ff0a32 3004              bmi setpinhi1
1528 ff0a34 8f2000fd          sta portb,1
1529 ff0a38 8dbd02   setpinhi1 sta portbcopy
1530 ff0a3b 60                rts
```

```
1531 ff0a3c a9ef      eoilow    lda #$ff-eoi
1532 ff0a3e 80ce                bra setpinlow
1533 ff0a40 a910      eoihigh   lda #eoi
1534 ff0a42 80e7                bra setpinhig
1535 ff0a44 af2000fd eoilow?   lda portb,1        ist eoi low?
1536 ff0a48 8910                bit #eoi
1537 ff0a4a d003                bne eoilowl       kein endbyte
1538 ff0a4c ce4402              dec eoiflag
1539 ff0a4f 60        eoilowl   rts
1540 ff0a50 a97f      davlow    lda #$ff-dav
1541 ff0a52 80ba                bra setpinlow
1542 ff0a54 a980      davhigh   lda #dav
1543 ff0a56 80d3                bra setpinhig
1544 ff0a58 af2000fd davlow?   lda portb,1        ist dav low?
1545 ff0a5c 8980                bit #dav
1546 ff0a5e d0f8                bne davlow?       nein, warten
1547 ff0a60 60                  rts
1548 ff0a61 af2000fd davhigh?  lda portb,1        ist dav low?
1549 ff0a65 8980                bit #dav
1550 ff0a67 f0f8                beq davhigh?      nein, warten
1551 ff0a69 60                  rts
1552 ff0a6a a920      ndachigh  lda #ndac
1553 ff0a6c 80bd                bra setpinhig
1554 ff0a6e a9df      ndaclow   lda #$ff-ndac
1555 ff0a70 809c                bra setpinlow
1556 ff0a72 af2000fd ndachigh? lda portb,1
1557 ff0a76 8920                bit #ndac
1558 ff0a78 f0f0                beq ndachigh
1559 ff0a7a 60                  rts
1560 ff0a7b a940      nrfdhigh  lda #nrfd
1561 ff0a7d 80ac                bra setpinhig
1562 ff0a7f a9bf      nrfdlow   lda #$ff-nrfd
1563 ff0a81 4c0e0a              jmp setpinlow
1564 ff0a84 af2000fd nrfdlow?  lda portb,1
1565 ff0a88 8940                bit #nrfd
1566 ff0a8a d0f8                bne nrfdlow?      warten bis low
1567 ff0a8c 60        nrfdlowl  rts
```

Die nächste hierachische Ebene darüber beginnt bei Zeile 1579 mit SENDEOI (ein Byte mit EOI-Signal senden), SENDBYTE und hernach bei Zeile 1604 mit GETBYTE. Es werden einzelne Bytes gesendet und empfangen, und man sieht am Ablauf der einzelnen Routinen sehr deutlich, wie das Handshake aufeinanderfolgt. - Dieses als Voraussetzung für immer höher aufsteigende Funktionen wie z.B. die Kommandos ILISTEN, UNLISTEN TALK und UNTALK. Noch darüber stehen in Zeile 1703 und 1870 die Teilprogramme IECOPEN und XCLOSE zum Öffnen und Schließen von Dateien.

```
1568 ff0a8d 226c00ff samplebus jsl _anykey?      abbruch mit escape
1569 ff0a91 9007                bcc samplebul
1570 ff0a93 c91b                cmp #escape
1571 ff0a95 d003                bne samplebul
1572 ff0a97 4cb404              jmp main1         bumms!
1573 ff0a9a af2000fd samplebul lda portb,1
1574 ff0a9e 2960                and #nrfd+ndac
1575 ff0aa0 c940                cmp #nrfd         nrfd high und ndac low?
1576 ff0aa2 d0e9                bne samplebus
1577 ff0aa4 60                  rts
1578 ff0aa5 53454e44            .byt 'sendeoi'
1579 ff0aac           sendeoi                     ;byte mit eoi low senden
1580 ff0aac 203c0a              jsr eoilow
```

```
1581 ff0aaf ad4702    sendbyte   lda iecbyte        aus der vorbereitung
1582 ff0ab2 49ff      sendbyted  eor #$ff           byte invertieren und senden
1583 ff0ab4 8dbe02               sta portacopy
1584 ff0ab7 dc3802               jml (iecoutlnk)    f. seriellen bus ggfs.
1585 ff0aba 8f2100fd sendbytel  sta porta,l
1586 ff0abe 208d0a               jsr samplebus
1587 ff0ac1 20500a               jsr davlow         data valid
1588 ff0ac4 20840a               jsr nrfdlow?       kommt handshake?
1589 ff0ac7 20720a               jsr ndachigh?
1590 ff0aca 20540a               jsr davhigh
1591 ff0acd a9ff                 lda #$ff           sende neutral
1592 ff0acf 8dbe02               sta portacopy
1593 ff0ad2 8f2100fd             sta porta,l
1594 ff0ad6 a909                 lda #9
1595 ff0ad8 221d0bff             jsl _wait          ca. 65 mikrosek. warten
1596 ff0adc 2c4402               bit eoiflag
1597 ff0adf 1009                 bpl sendbytel
1598 ff0ae1 20400a               jsr eoihigh        zuruecksetzen
1599 ff0ae4 9c4402               stz eoiflag
1600 ff0ae7 20290b               jsr unlisten
1601 ff0aea 60        sendbytel  rts
1603 ff0aeb 47455442             .byt 'getbyte'
1604 ff0af2 9c4402    getbyte    stz eoiflag        get byte from bus
1605 ff0af5 dc3b02               jml (iecinlink)    ggfs. f. serielle busroutinen
1606 ff0af8 207b0a    getbytel   jsr nrfdhigh
1607 ff0afb 206e0a               jsr ndaclow        bin empfangsbereit
1608 ff0afe 20580a               jsr davlow?        data valid?
1609 ff0b01 207f0a               jsr nrfdlow
1610 ff0b04 af2100fd             lda porta,l        get data-byte
1611 ff0b08 49ff                 eor #$ff           invertieren
1612 ff0b0a 8d4702               sta iecbyte
1613 ff0b0d 20440a               jsr eoilow?        setzte ggfs eoiflag
1614 ff0b10 206a0a               jsr ndachigh       lesen ist fertig
1615 ff0b13 20610a               jsr davhigh?
1616 ff0b16 206e0a               jsr ndaclow
1617 ff0b19 ad4702               lda iecbyte        datenbyte im akku
1618 ff0b1c 60                   rts
1620 ff0b1d 22280bff  _wait      jsl _waitl         wait in software, vorgabe im
                                                    accu
1621 ff0b21 22280bff             jsl _waitl         je durchlauf gut 7 mikrosek.
1622 ff0b25 3a                   dec a
1623 ff0b26 10f5                 bpl _wait
1624 ff0b28 6b        _waitl     rtl
1625 ff0b29 a93f      unlisten   lda #unlisn
1626 ff0b2b 48        unlistenl  pha
1627 ff0b2c 208d09               jsr setoutput
1628 ff0b2f 20020a               jsr atnlow
1629 ff0b32 68                   pla
1630 ff0b33 20b20a               jsr sendbyted      direkt senden
1631 ff0b36 201f0a               jsr atnhigh
1632 ff0b39 60                   rts
1634 ff0b3a a95f      untalk     lda #untlk
1635 ff0b3c 80ed                 bra unlistenl
1637 ff0b3e 20020a    ilisten    jsr atnlow         geraetenummer muss in iecdevi
                                                    ce sein
1638 ff0b41 ad4602               lda iecdevice
1639 ff0b44 0920                 ora #lisn
1640 ff0b46 20b20a               jsr sendbyted      direkt senden
1641 ff0b49 60                   rts                ggfs. sekundaeradresse nachsc
```

```
1643 ff0b4a bd5b02   olddevice lda open_dev,x     log. file x wieder einsetzen
1644 ff0b4d 8d4602             sta iecdevice     mit alten parametern
1645 ff0b50 bd6b02             lda open_sec,x
1646 ff0b53 8d4802             sta ieckanal
1647 ff0b56 60                 rts
1648 ff0b57 54414c4b           .byt 'talk'
1649 ff0b5b 20020a   talk       jsr atnlow        geraetenummer muss in iecdevi
                                                  ce sein
1650 ff0b5e ad4602             lda iecdevice
1651 ff0b61 0940               ora #tlk
1652 ff0b63 20b20a             jsr sendbyted     direkt senden
1653 ff0b66 ad4802             lda ieckanal      sekundaeradresse
1654 ff0b69 0960               ora #$60
1655 ff0b6b 20b20a   talkl     jsr sendbyted
1656 ff0b6e 201f0a             jsr atnhigh
1657 ff0b71 20ad09             jsr setinput      interface umsteuern
1658 ff0b74 60                 rts
1660 ff0b75 20020a   statussub jsr atnlow        geraetenummer muss in iecdevi
                                                  ce sein
1661 ff0b78 ad4602             lda iecdevice
1662 ff0b7b 0940               ora #tlk
1663 ff0b7d 20b20a             jsr sendbyted     direkt senden
1664 ff0b80 a96f               lda #$6f          fehlerkanal
1665 ff0b82 206b0b             jsr talkl
1666 ff0b85 20f20a             jsr getbyte
1667 ff0b88 8d4702             sta iecbyte
1668 ff0b8b c930               cmp #'0'          okay?
1669 ff0b8d 60                 rts
1670 ff0b8e 20750b   status?   jsr statussub
1671 ff0b91 d015               bne statusf       fehler
1672 ff0b93 203a0b             jsr untalk
1673 ff0b96 60                 rts
1674 ff0b97 22b402ff statusmes jsl _crlf         fehler auf iecbus
1675 ff0b9b 20f20a   statusmsl jsr getbyte
1676 ff0b9e 225d02ff           jsl _output
1677 ff0ba2 2c4402             bit eoiflag
1678 ff0ba5 10f4               bpl statusmsl     weitere ausgabe
1679 ff0ba7 60                 rts
1680 ff0ba8 20970b   statusf   jsr statusmes     anzeige statustext
1681 ff0bab 20290b             jsr unlisten      bus normalisieren
1682 ff0bae 20a00d             jsr talk_end
1683 ff0bb1 4cb404             jmp main1         zum monitor
1685 ff0bb4 a2e4     open_err  ldx #m19-mesbas    fehleranzeige
1686 ff0bb6 221203ff           jsl _mesout
1687 ff0bba 4cb404             jmp main1         fataler abbruch
1689                           ;x=log. kanal, akku=sekundaeradresse bei aufruf
1689
1690                           ;variable iecdevice=atives geraet, muss vorher
1690                          bestimmt sein
1691 ff0bbd e010     iopensub  cpx #16            zu hohe nummer?
1692 ff0bbf b0f3               bcs open_err
1693 ff0bc1 3c4b02             bit open_log,x    file open?
1694 ff0bc4 30ee               bmi open_err
1695 ff0bc6 9d6b02             sta open_sec,x
1696 ff0bc9 8d4802             sta ieckanal      aktiver kanal
1697 ff0bcc ad4602             lda iecdevice     zuordnung wie beim aufruf!!
1698 ff0bcf 9d5b02             sta open_dev,x    aktives geraet zum log. kanal
1699 ff0bd2 de4b02             dec open_log,x    kanal besetzt
```

```
1700 ff0bd5 8e4902              stx iecopnchn       aktiver kanal, nummer
1701 ff0bd8 60                  rts
1702 ff0bd9 4945434f            .byt 'iecopen'
1703 ff0be0 20bd0b   iecopen    jsr iopensub        oeffne kanal
1704 ff0be3 9c4402              stz eoiflag         normal beginnen
1705 ff0be6 208d09              jsr setoutput       zur sicherheit
1706 ff0be9 20020a              jsr atnlow
1707 ff0bec a940                lda #64             ca 1 sek warten
1708 ff0bee 8558                sta tempa
1709 ff0bf0 a900                lda #oneshot        menge durchgaenge in tempa vo
                                                    rladen !!
1710 ff0bf2 8f2a00fd            sta acr,1           one shot mode
1711 ff0bf6 a9ff     iecopntm   lda #$ff            16 ms bei 4 mhz je oneshot
1712 ff0bf8 8f2400fd            sta t1cl,1          zugleich ins latch
1713 ff0bfc 8f2500fd            sta t1ch,1          start timer
1714 ff0c00 af2000fd iecopntml  lda portb,1
1715 ff0c04 2920                and #ndac
1716 ff0c06 f01b                beq iecopen00       okay
1717 ff0c08 af2d00fd            lda ifr,1           interrupt-flag t1?
1718 ff0c0c 8940                bit #$40
1719 ff0c0e f0f0                beq iecopntml
1720 ff0c10 c658                dec tempa
1721 ff0c12 d0e2                bne iecopntm        restart
1722 ff0c14 a2a1                ldx #ml2-mesbas
1723 ff0c16 221203ff            jsl _mesout         text: kein geraet
1724 ff0c1a 20290b              jsr unlisten        bus normalisieren
1725 ff0c1d 20a00d              jsr talk end
1726 ff0c20 4cb404              jmp main1           abbruch mit fehler
1728 ff0c23 ad4602   iecopen00 lda iecdevice
1729 ff0c26 0920                ora #lisn           device hear
1730 ff0c28 20b20a              jsr sendbyted       direkt
1731 ff0c2b ad4802              lda ieckanal        sekundaeradresse
1732 ff0c2e 09f0                ora #open           operator vom typ $fx
1733 ff0c30 20b20a   iecopen0  jsr sendbyted       direkt senden
1734 ff0c33 201f0a              jsr atnhigh         ende kommando
1735 ff0c36 a000                ldy #0
1736 ff0c38 b90300   iecopen1  lda inbuff,y        kommandostring
1737 ff0c3b f01d                beq iecopenf        fehler, leerstring kann sein
1738 ff0c3d 8d4702              sta iecbyte
1739 ff0c40 b90400              lda inbuff+1,y      folgt 00 als ende?
1740 ff0c43 d00f                bne iecopen2
1741 ff0c45 ce4402              dec eoiflag
1742 ff0c48 203c0a              jsr eoilow          mit eoi senden
1743 ff0c4b ad4702              lda iecbyte
1744 ff0c4e 20af0a              jsr sendbyte
1745 ff0c51 4c290b              jmp unlisten        rts dort
1746 ff0c54          iecopen2                       ;normales byte
1747 ff0c54 20af0a              jsr sendbyte
1748 ff0c57 c8                  iny
1749 ff0c58 80de                bra iecopen1
1750 ff0c5a 60       iecopenf   rts                 kurz-schluss
1752 ff0c5b 48       iecrdopen  pha                 ieckanal lesen programm oder
                                                    sequ.
1753 ff0c5c da                  phx                 log. kanal
1754 ff0c5d 208d09              jsr setoutput       programm lesen
1755 ff0c60 22720cff            jsl _iecstrng       dateiname usw.
1756 ff0c64 fa                  plx
1757 ff0c65 68                  pla
1758 ff0c66 20e00b              jsr iecopen
```

```
1759 ff0c69 208e0b            jsr status?
1760 ff0c6c 9c4402            stz eoiflag
1761 ff0c6f 4c5b0b            jmp talk          rts dort
1763 ff0c72 a2c3     _iecstrng ldx #ml5-mesbas   text: string
1764 ff0c74 221203ff          jsl _mesout
1765 ff0c78 22b000ff          jsl _getstrng     iec-kommandostring holen
1766 ff0c7c 4cb402            jmp _crlf         rtl dort
```

Die Hauptbefehlsebene ist ab Zeile 1769 bei IECMAIN erreicht. Von hier aus werden die Haupttätigkeiten im Floppy-Betrieb angesprungen, nämlich DIRECTORY (Inhaltsverzeichnis anzeigen) IECREAD, IECWRITE und die Ansprache des Kommandokanals 15 in IECKANL15. Die Routine zur Anzeige des Inhaltsverzeichnisses fußt dabei auf der Veröffentlichung in Heft 27 (P. Rix). - Bei den lesenden und schreibenden Routinen für ganze Files muß man zwischen solchen für Programme und für Texte (Editor) unterscheiden. Bei Programmen muß in den ersten beiden Bytes die Ladeadresse berücksichtigt werden. Es ist beim Lesen von Programmen allerdings ab Zeile 1993 ein verschiebliches Laden vorgesehen, wenn die Variable INS_FLAG negativ ist. Dann wird der an anderer Stelle bereits vorgeladene Zeiger ADRES_VON benutzt. - Bei einem Adreßraum von 256 Bänken auf 65816-Rechnern muß natürlich in Zeile 1983 manuell disponiert werden, in welche Bank geladen werden soll. Man könnte natürlich auch ein Aufzeichnungsformat wählen, das die Bankinformation enthält. Es würde aber zur Inkompatibilität mit dem üblichen CBM-Format führen.

```
1768 ff0c7f 5f494543          .byt '_iecmain'
1769 ff0c87          _iecmain                  ;hauptebene iec-bus-programme
1770 ff0c87 229d06ff          jsl _device?      welches geraet?
1771 ff0c8b a28e     iecmain0 ldx #ml1-mesbas   welche funktion: ld$k ?
1772 ff0c8d 221203ff          jsl _mesout
1773 ff0c91 229400ff          jsl _getkeyg      antwort?
1774 ff0c95 8d4a02            sta ieccmd        befehl merken
1775 ff0c98 225d02ff          jsl _output
1776 ff0c9c 22b402ff          jsl _crlf
1777 ff0ca0 a205              ldx #5            befehle-1
1778 ff0ca2 ad4a02            lda ieccmd        befehlstaste
1779 ff0ca5 dfba0cff iecmain1 cmp iectab,lx
1780 ff0ca9 f009              beq iecmain2
1781 ff0cab ca                dex
1782 ff0cac 10f7              bpl iecmain1
1783 ff0cae 22b402ff          jsl _crlf
1784 ff0cb2 80d7              bra iecmain0      unbekannt!
1786 ff0cb4 8a       iecmain2 txa               gefundenes item
1787 ff0cb5 0a                asl a
1788 ff0cb6 aa                tax
1789 ff0cb7 7cc00c            jmp (iecwords,x)  sprungverteiler
1791 ff0cba 4c44244b iectab   .byt 'ld$kr',escape gueltige tasten
1791 ff0cbf 1b
1792 ff0cc0 9f0e080e iecwords .wor iecread,iecwrite,directory,ieckanl15 exekut
                                                iernde routinen
1792 ff0cc4 d50cc30d
1793 ff0cc8 380fb404          .wor _iecreset,main1
1795 ff0ccc 44495245          .byt 'directory'
1796 ff0cd5 208d09   directory jsr setoutput     directory lesen
1797 ff0cd8 22720cff          jsl _iecstrng     kommando holen
1798 ff0cdc a900              lda #0            sek. adresse
1799 ff0cde aa                tax               x=0 log kanal
1800 ff0cdf 20e00b            jsr iecopen
1801 ff0ce2 208e0b            jsr status?       fehler=abbruch
1802 ff0ce5 205b0b            jsr talk          mit umschaltung auf empfang
1803 ff0ce8 a005              ldy #5
```

```
1804 ff0cea 20f20a   dir1    jsr getbyte
1805 ff0ced 88               dey
1806 ff0cee d0fa             bne dir1
1807 ff0cf0 22b402ff         jsl _crlf
1808 ff0cf4 ad4702           lda iecbyte
1809 ff0cf7 22e802ff         jsl _hex_out       drive-#
1810 ff0cfb 20f20a           jsr getbyte
1811 ff0cfe 20f20a           jsr getbyte
1812 ff0d01 22c402ff         jsl _spaceout
1813 ff0d05 20f20a   dir2    jsr getbyte        kopfzeile
1814 ff0d08 f006             beq dir3
1815 ff0d0a 225d02ff         jsl _output
1816 ff0d0e 80f5             bra dir2
1817 ff0d10 22b402ff dir3    jsl _crlf
1818 ff0d14 a01b             ldy #27
1819 ff0d16 a92d             lda #'-'           unterstreichen
1820 ff0d18 225d02ff dir4    jsl _output
1821 ff0d1c 88               dey
1822 ff0d1d d0f9             bne dir4
1823 ff0d1f 22b402ff         jsl _crlf
1824 ff0d23 20f20a   dir5    jsr getbyte        zeile mit eintrag
1825 ff0d26 2c4402           bit eoiflag
1826 ff0d29 305a             bmi dirend
1827 ff0d2b c901             cmp #1
1828 ff0d2d d0f4             bne dir5
1829 ff0d2f 20f20a           jsr getbyte
1830 ff0d32 20f20a           jsr getbyte
1831 ff0d35 8571             sta zw_wert+1
1832 ff0d37 20f20a           jsr getbyte
1833 ff0d3a 8570             sta zw_wert        zur zahlenumwandlung
1834 ff0d3c f8               sed                dezimal
1835 ff0d3d 646d             stz iexpr
1836 ff0d3f 646e             stz iexpr+1
1837 ff0d41 a010             ldy #16
1838 ff0d43 18               clc
1839 ff0d44 a56e     dir6    lda iexpr+1
1840 ff0d46 656e             adc iexpr+1
1841 ff0d48 856e             sta iexpr+1
1842 ff0d4a a56d             lda iexpr
1843 ff0d4c 656d             adc iexpr
1844 ff0d4e 856d             sta iexpr
1845 ff0d50 2671             rol zw_wert+1
1846 ff0d52 2670             rol zw_wert
1847 ff0d54 88               dey
1848 ff0d55 10ed             bpl dir6
1849 ff0d57 a56d             lda iexpr
1850 ff0d59 22e802ff         jsl _hex_out
1851 ff0d5d a56e             lda iexpr+1
1852 ff0d5f 22e802ff         jsl _hex_out
1853 ff0d63 22c402ff         jsl _spaceout      blockzahl fertig
1854 ff0d67 20f20a   dir7    jsr getbyte
1855 ff0d6a f013             beq dir9
1856 ff0d6c c920             cmp #space
1857 ff0d6e f0f7             beq dir7
1858 ff0d70 225d02ff         jsl _output
1859 ff0d74 20f20a   dir8    jsr getbyte
1860 ff0d77 f006             beq dir9
1861 ff0d79 225d02ff         jsl _output
1862 ff0d7d d0f5             bne dir8
```

```
1863 ff0d7f 22b402ff dir9      jsl _crlf
1864 ff0d83 809e             bra dir5
1865 ff0d85 20a00d   direnD   jsr talk_end
1866 ff0d88 6b               rtl
1867                         ;jmp main1
1869 ff0d89 58434c4f         .byt 'xclose'
1870 ff0d8f bd5b02   xclose   lda open_dev,x     close log. kanal m. alten par
                                                ametern
1871 ff0d92 8d4602           sta iecdevice      kanal-nr. in x
1872 ff0d95 bd6b02           lda open_sec,x
1873 ff0d98 8d4802           sta ieckanal
1874 ff0d9b a900             lda #0
1875 ff0d9d 9d4b02           sta open_log,x     tabelle offener files=geschlo
                                                ssen
1876 ff0da0 203a0b   talk_end jsr untalk
1877 ff0da3 20290b   writ_end jsr unlisten
1878 ff0da6 203e0b           jsr ilisten
1879 ff0da9 ad4802           lda ieckanal
1880 ff0dac 09e0             ora #close
1881 ff0dae 20b20a           jsr sendbyted
1882 ff0db1 201f0a           jsr atnhigh
1883 ff0db4 20290b           jsr unlisten
1884 ff0db7 ae4902           ldx iecopnchn      nummer log. kanal
1885 ff0dba a900             lda #0
1886 ff0dbc 9d4b02           sta open_log,x     tabelle offener files=geschlo
                                                ssen
1887 ff0dbf 20d809           jsr iecinit        neutraler bus
1888 ff0dc2 60               rts
1890 ff0dc3          ieckan115                  ;kommandokanal 15 oeffnen
1891 ff0dc3 208d09           jsr setoutput
1892 ff0dc6 22720cff         jsl _iecstrng      kommando holen
1893 ff0dca 203e0b           jsr ilisten
1894 ff0dcd a90f             lda #$0f
1895 ff0dcf 8d4802           sta ieckanal
1896 ff0dd2 a96f             lda #$6f
1897 ff0dd4 20300c           jsr iecopen0
1898 ff0dd7 208e0b           jsr status?
1899 ff0dda 20750b           jsr statussub      anzeigen, was gelaufen ist
1900 ff0ddd ad4702           lda iecbyte        sollte meistens '0' sein
1901 ff0de0 225d02ff         jsl _output
1902 ff0de4 209b0b           jsr statusms1
1903 ff0de7 20a00d           jsr talk_end
1904 ff0dea 6b       ieckan116 rtl
1906 ff0deb 20e00b   iecwrsub jsr iecopen
1907 ff0dee 9c4402           stz eoiflag
1908 ff0df1 203e0b   iecwrsub1 jsr ilisten
1909 ff0df4 ad4802           lda ieckanal
1910 ff0df7 0960             ora #$60
1911 ff0df9 20b20a   iecrdsub jsr sendbyted      save
1912 ff0dfc 201f0a           jsr atnhigh
1913 ff0dff 60               rts
1915 ff0e00 49454357         .byt 'iecwrite'
1916 ff0e08          iecwrite                   ;schreibe auf iec-bus
1917 ff0e08 a564             lda umgebung
1918 ff0e0a c945             cmp #'e'           vom editor?
1919 ff0e0c f044             beq iecwrited      ja
1920 ff0e0e 225e01ff         jsl _from_to       bereich holen f. save program
1921 ff0e12 9001             bcc iecwritp       bereich nicht ueberschritten
1922 ff0e14 6b               rtl
```

```
1923 ff0e15 22720cff iecwritp  jsl _iecstrng       kommando holen
1924 ff0e19 208d09             jsr setoutput
1925 ff0e1c a901               lda #1              cbm-kanal f. save
1926 ff0e1e aa                 tax                 log. kanal=1
1927 ff0e1f 20eb0d             jsr iecwrsub        listener aufrufen write
1928 ff0e22 208e0b             jsr status?         verbleibt im output mode
1929 ff0e25 20f10d             jsr iecwrsubl       listener wieder rufen
1930 ff0e28 a565               lda adres_von
1931 ff0e2a 20b20a             jsr sendbyted       sal
1932 ff0e2d a566               lda adres_von+1     sah, startadresse auf disk
1933 ff0e2f 20b20a             jsr sendbyted
1935 ff0e32 a000      iecwritp0 ldy #0
1936 ff0e34 b765               lda (adres_von,1),yprogrammbyte
1937 ff0e36 8d4702             sta iecbyte
1938 ff0e39 a901               lda #1
1939 ff0e3b 206003             jsr addvon          adres_von=adres_von+1
1940 ff0e3e 20ac03             jsr dumpende?
1941 ff0e41 900a               bcc iecwritp1       nein
1942 ff0e43 20ac0a    iecwriend jsr sendeoi
1943 ff0e46 20290b             jsr unlisten        fertig
1944 ff0e49 20a30d             jsr writ_end        close
1945 ff0e4c 6b                 rtl
1946 ff0e4d 20af0a    iecwritp1 jsr sendbyte        normal senden
1947 ff0e50 80e0               bra iecwritp0
1949 ff0e52           iecwrited                     ;aus den editor saven
1950 ff0e52 22720cff           jsl _iecstrng       kommando holen
1951 ff0e56 ad4602             lda iecdevice       sequentiell schreiben
1952 ff0e59 aa                 tax                 log. kanal
1953 ff0e5a 20eb0d    iecwrseq  jsr iecwrsub       listener rufen, einsprung f a
                                                   ndere kanaele
1954 ff0e5d ad4602             lda iecdevice       sonderbehandlung drucker, ger
                                                   aet 4
1955 ff0e60 c904               cmp #4
1956 ff0e62 f003               beq iecwrse1
1957 ff0e64 208e0b             jsr status?
1958 ff0e67 20f10d    iecwrse1  jsr iecwrsubl       listener wieder rufen
1960 ff0e6a a000               ldy #0              editor-dump bis zum ende
1961 ff0e6c b77c      iecwrite1 lda (nowln,1),y
1962 ff0e6e 8d4702             sta iecbyte
1963 ff0e71 f00a               beq iecwritec       ende
1964 ff0e73 c8        iecwrite2 iny                 y=1
1965 ff0e74 207110             jsr nowlnundy       nowln=nowln+1
1966 ff0e77 a000               ldy #0
1967 ff0e79 b77c               lda (nowln,1),y     folgt der schluss?
1968 ff0e7b d00e               bne iecwrite3
1969 ff0e7d ce4402    iecwritec dec eoiflag
1970 ff0e80 22430eff           jsl iecwriend       aufraeumen
1971 ff0e84 224c00ff           jsl _sysout         normalausgabe
1972 ff0e88 4c0012             jmp _edicmd
1973 ff0e8b 20af0a    iecwrite3 jsr sendbyte        normal senden
1974 ff0e8e 80dc               bra iecwrite1
1976 ff0e90 49454352           .byt 'iecreadv0'
1977 ff0e99 a9ff      iecreadv0 lda #$ff            verschiebliches laden, taste
                                                   v
1978 ff0e9b 859e               sta ins_flag
1979 ff0e9d 800c               bra iecreadvv
1980 ff0e9f a564      iecread   lda umgebung        vom monitor oder editor?
1981 ff0ea1 c945               cmp #'e'
1982 ff0ea3 f063               beq iecreaded       editor
```

```
1983 ff0ea5 22ac06ff iecreadp  jsl _bank?          welche ladebank?
1984 ff0ea9 649e               stz ins_flag        lesen an alte adresse
1985 ff0eab a900     iecreadvv lda #0              lesekanal cbm f. programme
1986 ff0ead aa                 tax                 log. kanal
1987 ff0eae 205b0c             jsr iecrdopen
1988 ff0eb1 20f20a             jsr getbyte         sal
1989 ff0eb4 249e               bit ins_flag        skip b. verschieb. laden
1990 ff0eb6 3002               bmi iecreadvl
1991 ff0eb8 8565               sta adres_von       in den pointer
1992 ff0eba 20f20a   iecreadvl jsr getbyte
1993 ff0ebd 249e               bit ins_flag        skip b. verschieb. laden
1994 ff0ebf 3002               bmi iecreadv2
1995 ff0ec1 8566               sta adres_von+1
1996 ff0ec3 22b402ff iecreadv2 jsl _crlf
1997 ff0ec7 a200               ldx #ml-mesbas
1998 ff0ec9 221203ff           jsl _mesout
1999 ff0ecd 22c402ff           jsl _spaceout
2000 ff0ed1 22d302ff           jsl _adrout         ladeadresse anzeigen
2001 ff0ed5 a000               ldy #0              f. adressierung indirekt
2002 ff0ed7 20f20a   iecreadpl jsr getbyte         ladeschleife
2003 ff0eda 9765               sta (adres_von,1),yabspeichern mit bank
2004 ff0edc a901               lda #1
2005 ff0ede 206003             jsr addvon          pointer=pointer+1
2006 ff0ee1 2c4402             bit eoiflag         ende?
2007 ff0ee4 10f1               bpl iecreadpl       weiter!
2008 ff0ee6 208d09             jsr setoutput
2009 ff0ee9 20a00d             jsr talk_end        close
2010 ff0eec a209               ldx #m3-mesbas
2011 ff0eee 221203ff           jsl _mesout
2012 ff0ef2 22c402ff           jsl _spaceout
2013 ff0ef6 22d302ff           jsl _adrout         endadresse+1 anzeigen
2014 ff0efa 22b402ff           jsl _crlf
2015 ff0efe 6b                 rtl                 ende des programmladens
2017                           ;einlesen in den editor, zeile um zeile
2018                           ;folgende routine nur initialisierend benutzen
2019                           ;fuer folgezeilen dann iecreade0
2020 ff0eff 49454352           .byt 'iecreaded'
2021 ff0f08 a902     iecreaded lda #2              lesekanal cbm sequentiell
2022 ff0f0a aa                 tax                 log. kanal=2
2023 ff0f0b 205b0c             jsr iecrdopen
2024 ff0f0e a000     iecreade0 ldy #0              f. adressierung indirekt
2025 ff0f10 20f20a   iecreadel jsr getbyte         ladeschleife
2026 ff0f13 990300             sta inbuff,y        abspeichern in den puffer
2027 ff0f16 f018               beq iecreadez       ende textfile
2028 ff0f18 c04f               cpy #79             pufferlaenge?
2029 ff0f1a b006               bcs iecreade2       voll
2030 ff0f1c c8                 iny
2031 ff0f1d eaaeeaea           .byt $ea,$ae,$ea,$ea,$ea
2031 ff0f21 ea
2032 ff0f22 c90d     iecreade2 cmp #cr
2033 ff0f24 d0ea               bne iecreadel       weiter!
2035 ff0f26 88                 dey
2036 ff0f27 8401               sty stringlen
2037 ff0f29 a900               lda #0              bufferabschluss
2038 ff0f2b 990300             sta inbuff,y
2039 ff0f2e 18                 clc                 kein file-ende
2040 ff0f2f 6b                 rtl
```

```
2041 ff0f30 208d09   iecreadez jsr setoutput
2042 ff0f33 20a00d             jsr talk_end       close
2043 ff0f36 38                 sec                markierung
2044 ff0f37 6b                 rtl
2046 ff0f38 20d809   _iecreset jsr iecinit        reset auf ifc geben, taste r
2047 ff0f3b a9c0               lda #$c0           manual output low to cb2
2048 ff0f3d 8f2c00fd           sta pcr,l
2049 ff0f41 a905               lda #5
2050 ff0f43 8558               sta tempa
2051 ff0f45 22df09ff           jsl _timeout?      warten
2052 ff0f49 a900               lda #0
2053 ff0f4b 8f2c00fd           sta pcr,l          nimm reset zurueck
2054 ff0f4f 6b                 rtl
```

# Editorial

Das vorliegende Heft erscheint wesentlich später als ursprünglich geplant. Der Herausgeber bittet die Leser um Verständnis dafür, daß dabei einerseits die deutlichen Veränderungen am Markt der Mikrocomputer mitgespielt haben und andererseits etwas auch persönliche Gründe. - Im Bereich der Personal Computer ist festzustellen, daß die Firma Commodore sich von der früher sehr erfolgreichen Linie der Geräte 8032, 8096 usw. getrennt hat. Die Linie 610/710 nebst Zubehör wird durch einen Distributor als Restposten billig abgestoßen. Als wichtiger Neuzugang im Bereich der 65xxx-Geräte ist seit dem Herbst 1985 nur noch der PC-128 zu nennen. In seiner Leistungsfähigkeit reicht er durchaus in den Bereich der Personal Computer, es ist aber unwahrscheinlich, daß er in diesem Bereich noch nennenswerte Erfolge erzielen wird. Immerhin muß man sich vor Augen halten, daß es bei einem Preis von inzwischen schon unter DM 700 für das Gerät und bei einer Vermarktung über Kaufhäuser kaum Softwarehäuser geben wird, die dafür 'seriöse' Software zu entwickeln bereit sind. Andererseits darf der Rechner wegen zahlreicher am Markt erhältlicher Bücher als gut dokumentiert gelten. So darf man ihn als durchaus empfehlenswertes Gerät bezeichnen, wenn man sich der Mühe einer eigenen Programmierung unterziehen will. - Die CPU 8502 des Rechners und die von ihr abhängige Architektur sind ein Fingerzeig, daß man mit einem Adreßbus von 16 Bit auch bei 65xx an eine Grenze gestoßen ist. Mit den Beschreibungen und Vorschlägen in diesem Heft hofft der Herausgeber den Gebrauch dieses Rechners etwas zu befördern.

Seitens der großen Hersteller ist noch kein Rechner mit der moderneren CPU G65SC816 am Markt, wenn es auch Gerüchte gibt, daß sich da etwas tun könnte. Die CPU wurde bereits in früheren Heften dieser Zeitschrift vorgestellt. In diesem Heft nun wird ausführlich über ihre Programmierung berichtet, und zwar am Beispiel des in der Zeitschrift mc veröffentlichten Europakarten-Computers der berliner Firma MCT und seines Betriebssystems. Wenn man die hier dargestellten beispielhaften Programmblöcke einmal genauer studiert, so wird man feststellen, daß die CPU zwar nicht alle Wünsche eines Programmierers erfüllt, daß sie aber gut zu hantieren ist, wenn man erst einmal auf ihre Besonderheiten hingewiesen wurde. Insofern sind Rechner mit 65816 durchaus eine Angelegenheit für 'Aufsteiger', die ihre Erfahrungen und Programm-Module in einer leistungsfähigeren Umgebung weiterbenutzen möchten.

Im Bereich der Computer mit CPU 6xxxx hat es seit Ende 1985 einen deutlichen Durchbruch gegeben. Zwar gab es bis dahin schon zahlreiche Fertiggeräte mit einer CPU 68xxx, sie waren aber relativ teuer. Auch wurden zahlreiche Platinen- und Nachbaucomputer mit einer solchen CPU angeboten. Die Wende trat aber erst mit dem Erscheinen der preiswerten Atari 520ST-Linie ein. Es wurden in wenigen Monaten sicher mehr Geräte verkauft, als seinerzeit in Jahren z.B. vom AIM 65 oder von CBM 80xx. Dieser Erfolg trat ein, obwohl es anfangs kaum Software für die Rechner gab. Überzeugend waren zunächst das Preis-Leistungsverhältnis, das hochauflösende und gestochen scharfe Monitorbild, die Benutzeroberfläche mit GEM und wohl auch die Tatsache, daß ein Speicher von 512 KB bzw. 1 MB ein freizügiges Arbeiten verspricht, ohne daß man schon bei der Anschaffung an Erweiterungen mit Zusatzkarten denken muß. - Die Software-Situation hat sich seither deutlich gebessert. Es gibt soviele Assembler, Sprachcompiler und Anwenderprogramme, daß

man schon nicht mehr die Zeit findet, um sie auszuprobieren. Und auch hier findet man einen deutlichen Preisabstand z.B. zu vergleichbarer Software für die IBM-PCs. - Interfaces für den Betrieb weiterer Peripherie sind im Moment allerdings noch rar. Und man darf wohl auch sagen, daß es zwar Programme gibt, aber noch zu wenig Dokumentation und Darstellungen, wie man den 68000 im Atari programmiert. - Dieses Heft macht daher einen Anfang zur Assembler-Programmierung des Atari, und zwar mit einem vollständigen Programm zum Empfang seriell empfangener Daten und zum Abspeichern derselben auf der Floppy. Programme dieser Art sind sicher schon erhältlich, aber wohl kaum dokumentiert. Dieses erste Beispiel soll dem Leser daher zeigen, wie einfach man erstens mit dem Befehlssastz des 68000 umgehen kann und zweitens, welche gut vorbereiteten Hilfen in den Funktionen des Betriebssystems GEMDOS zu finden sind. Dieser Verbund zwischen Betriebssystem und Programmierung soll uns auch in künftigen Ausgaben beschäftigen. Der von anderen 6xxxx-Systemen herkommende Leser wird dabei feststellen, daß man mit wenig Code und möglichst vielen Bibliotheksfunktionen auch in dieser zumeist neuen Umgebung gut zurecht kommt.

Mit den Themen PC-128, CPU 65816 und 68000-Assembler auf dem Atari geht dieses Heft auf aktuelle Entwicklungen ein. Es war dabei das Ziel, heute oder später verwertbare Information aus der eigenen Beschäftigung mit dem Stoff zu vermitteln. Man möge darin aber auch den Ehrgeiz sehen, nicht in den Tenor vieler Zeitschriften einfallen zu wollen, die etwas 'über' Geräte und ihre Programmierung in Ansätzen berichten, ohne sich lange genug 'mit' dem Stoff beschäftigt zu haben. Diese Beschäftigung kann, wie es die letzten Monate gezeigt haben, langwierig sein, weil es in der Branche nach wie vor an ausreichender Dokumentation mangelt oder weil sie auch fehlerhaft ist. - In diesem Sinne sind die Leser auch weiterhin gebeten, aus ihren Erfahrungen Hinweise und Artikel zum MICRO MAG beizusteuern. In der Sicht des Herausgebers ist nach wie vor viel Informationsarbeit zu leisten, und man darf davon ausgehen, daß vieles noch verbessert werden kann, was heute zwar läuft, aber eben nicht so befriedigend. - Das nächste Heft dieser Zeitschrift ist zur Herausgabe im Juli 1986 geplant.

# Bücher

**Hoffmann, P., Nicoloff, T.: MS-DOS Anwenderhandbuch.** Verlag McGraw Hill, Hamburg 1985, 250 S. DM 45,--, ISBN 3-89028-028-5. Das Buch stellt alle Versionen des Microsoft-DOS dar (Versionen 3.0 und 3.1 im Anhang). Wir finden folgende Gliederung des Stoffes: Einführung, Arbeiten mit dem MS-DOS, Befehle, MS-DOS für Fortgeschrittene, Fehlerbehandlung, Beschreibung von Anwendungsprogrammen. Das Buch gibt damit eine Übersicht, man darf von ihm jedoch keine tiefergehenden Hinweise zu Anwendungstechniken erwarten.

**Willen, D.C., Krantz, J.I.: IBM PC/XT Assembler-Programmierung CPU 8088.** Verlag te-wi, München 1985, 330 S. DM 66,--, ISBN 3-921803-59-4. Der Text verbindet eine Einführung in die Programmierung des 8088 mit einer detaillierten anwendungsbezogenen Beschreibung des Computers IBM PC/XT. Die Wirkung aller Befehle kann in Systembildern verfolgt werden, BIOS, DOS, Macroassembler, EDLIN, ASM und LINK werden im Einsatz gezeigt. Die wesentlichen Systemkomponenten werden beschrieben. Die Programmierung befaßt sich mit Ein- und Mehrfarbgrafiken, Druckeransteuerung, serielle und parallele E/A, Interruptbehandlung, akustische Effekte usw. Im Anhang ist der Befehlssatz dokumentiert. - Das Buch zeigt damit Beispiele für die mögliche Programmierung insbesondere der Hardware. Weniger jedoch, als es der Titel erwarten läßt, wird auf die Assemblersprache eingegangen und wie sie, in Variationen, zur Lösung verschiedener Aufgaben benutzt werden kann.

**Lien, D. A.: BASIC-Programmierung PC-10/PC-20.** te-wi Verlag, München 1985, 488 S., DM 59,-, ISBN 3-921803-66-7. Dieses Buch gilt den 'kompatiblen' von Commodore. Mit über 70 am Gerät erprobten Übungsbeispielen gibt der amerikanische Autor einen didaktisch hervorragenden BASIC-Kurs. Vom Tastenfeld und der Bedienung ausgehend,werden die Grundlagen eines Programmentwurfes und sämtliche Befehle mit Beispielen beschrieben, auch Merge, Chain, Poke usw. für 'Fortgeschrittene'. Der Text ist klar verständlich geschrieben und in der druckmäßigen Gestaltung übersichtlich gegliedert.

**Newman, W.M., Sproull, R.F.: Grundzüge der interaktiven Computergrafik.** Bei Mc Graw-Hill, Hamburg 1986, 580 S., DM 58,--, ISBN 3-89028-015-3. Das Gebiet der Computergrafik gewinnt an Bedeutung. Das Buch der Autoren gilt als Standardwerk in der englischen Sprache. Es informiert, unterstützt durch zahlreiche Illustrationen, über alle Möglichkeiten der Computergrafik und auch die neuesten Entwicklungen, einschließlich der dreidimensionalen Grafik. Es enthält die mathematischen Konzepte ebenso wie praktische Anleitungen zum Entwurf eigener Grafiksysteme und Anwendungsprogrammen. Als besonders wertvoll mögen sich auch die etwa 550 Literaturhinweise am Schluß des Textes erweisen. Es wird wirklich eine große Fülle des Stoffes und eine hervorragende Übersicht geboten.

**Schineis, R., Braun, U.M., Demgensky, N.: C 128 ROM-Listing, Operating System.** Bei Markt & Technik, Haar 1986, 450 S., DM 49,--, ISBN 3-89090-221-9. Auf etwa 300 Seiten wird das disassemblierte und nachkommentierte Betriebssystem des Computers dargestellt. Dabei finden wir nicht nur, soweit als möglich, Commodore-Labels, sondern auch Speicherbelegungen und dokumentierte Signale und Registerbelegungen für die im Einsatz befindlichen Chips. Außerdem werden die wichtigen Systemroutinen beschrieben und Beispiele für ihren Einsatz gegeben. - Wer sich intensiver mit dem PC-128 befassen will, wird auf dieses klar aufgemachte Buch nicht verzichten können. Es mag sogar auch noch den späten Erwerbern eines CBM 610 oder 710 aus der Dokumentationsnot helfen: Wenn auch die Bank- und Segmentumschaltung in den Computern unterschiedlich gelöst ist und ebenso der IEC-Bus (seriell beim PC-128), so wird man doch weite Parallelen in der Ablauflogik finden.

**Gerits, Schieb, Thrun: Commodore 128 Intern.** Verlag Data Becker, Düsseldorf 1985, 507 S., DM 69,--, ISBN 3-89011-098-3. Es ist eines der schnell herausgegebenen Bücher aus diesem Verlag. In der Lesbarkeit und Kommentierung des Betriebssystems hält es keinen Vergleich mit dem vorerwähnten Titel aus. Dafür wird mehr auf die Systembausteine eingegangen: VIC, CIA, SID und VDC 8563.

**Brückmann, Englisch, Gerits: Atari ST Intern.** Verlag Data Becker, Düsseldorf 1985, 464 S. DM 69,-, ISBN 3-89011-119-X. Es war eines der ersten Bücher zum Atari 520ST mit den Hauptabschnitten Die ICs, Die Schnittstellen, Das Betriebssystem. Auf etwa 160 Seiten ist dabei ein disassembliertes und nachkommentiertes BIOS abgedruckt, dessen Studium durchaus seinen Wert hat, auch wenn das TOS heute (sicher etwas verändert) in Festwertspeichern und an anderer Stelle residiert. Die Beschreibung des Betriebssystems mit BIOS, XBIOS und GEMDOS entspricht in etwa der Dokumentation im Entwicklungspaket von Atari und dem sog. Hitchhiker's Guide. Trotz verschiedener Ungenauigkeiten in diesen Quellen, die sich bis in dieses Buch hin fortgepflanzt haben, kann man es als Assembler-Programmierer durchaus in die Hand nehmen, wenn man den Inhalt kritisch behandelt.

**Szczepanowski, G.: Das große GEM-Buch zum Atari ST.** Data Becker, Düsseldorf 1985, 459 S., DM 49,--, ISBN 3-89011-125-4. Nach einer kurzen Übersicht zum Atari 520ST wird der Versuch gemacht, kurze Einführungen in den 68000-Assembler und die Sprache C zu geben. Zusammen mit Hinweisen auf den Linker und Direktiven des Assemblers werden dafür nur etwa 65 Seiten benötigt, viel zu wenig, um auch nur ein oder zwei Themen gründlich darzustellen. Die letzten etwa 400 Seiten dienen der Aufzählung nebst Parameterübergaben der VDI- und AES-Bibliothek des GEM. Dabei sind einige kurze Anwendungsbeispiele enthalten, und zwar in Assembler und in 'C'. - Die Quelle für die Beschreibung der Funktionen ist nicht zu erkennen, es muß sich jedoch um eine Veröffentlichung von Digital Research zum GEM handeln. In diesem Buch, das mehr eine Aneinanderreihung von Themen ist, wird die Darstellung von Zusammenhängen vermißt, auch Grafiken hätten dem Stoff gut getan. Die Programmierung von GEM-Funktionen bleibt damit eine zeitaufwenige Angelegenheit, weil man soviel noch selbst ausprobieren muß. Dem Anfänger wird daher mehr empfohlen, sich zunächst mit TOS-Funktionen zu befassen.

Roland Löhr

# Sprungleiste CBM 610/710

Im Zusammenhang mit dem Ausverkauf von Commodore-Computern der Typen 710 und 610 durch die Fa. Völkner sind auf den Herausgeber vermehrt Anfragen zugekommen, was zu diesen Geräten noch als Dokumentation, Programmen und ggfs. als Erweiterungsmodul zu Verfügung steht. Hierzu: Das RAM/ROM/I/O-Listing der Fa. Hard und Soft GmbH ist dort beim Verleger, hier und auch in der sonstigen Distribution total ausverkauft und mit keinem Exemplar mehr nachlieferbar. Um gleichwohl den spät hinzukommenden Betreibern etwas Hilfe zu geben, wird aus vorg. Buch die Sprungleiste des Kernels (hinten im F-ROM) mit den grundlegenden Routinen des Betriebssystems abgedruckt. Dazu noch folgende Hinweise: Es soll etwa 4 Versionen der Festwertspeicher und des Betriebsprogrammes gegeben haben. Man muß also damit rechnen, daß sich Adressen noch etwas verschoben haben. Weiterhin: Viele Routinen springen indirekt über das RAM. Die Programmfortsetzung findet man in diesen Fällen durch die Betrachtung der dort niedergelegten Vektoren. - Die Logik der neueren CBM-Betriebssysteme und der BASIC-Versionen 4 und 7 dürfte auf Strecken vergleichbar sein. Insofern sollte man auch in Betracht ziehen, z.B. folgendes Buch heranzuziehen: C 128 ROM-Listing, Operating System, ISBN 3-89090-221-9, bei Markt & Technik, DM 49.

Eine Platine für die Cartridge, $2000, $4000, $6000 für CMOS-RAM oder EPROM 2764/27128, ist beziehbar bei Hennig Grawe, Sterzenweg 29, 8193 Ammerland. Ggfs. folgt noch ein Expansion Adapter für die Systemerweiterung RAM/EPROM/IO.

Software für diese Rechner ist außer in dieser Zeitschrift wohl kaum oder gar nicht veröffentlicht worden. Folgende Themen und Hefte des MICRO MAG mögen noch interessant sein: Commodore CBM 710, BASIC 4 (Nr. 32), Commodore Chips, CBM 710 Spezial (33), CBM 710 Spezial, Datenbank (36), CBM 6x0 und 7x0 als Terminal, Terminalbetrieb mit CBM 710 (37), GMA-Text für CBM 720 (39), RENUMBER und FIND, Intelligentes Terminal (43), AUTO für CBM 710/720 (44), Texteditor CBM 720 (45).

```
5255  ff6c  4c 9d fe  knwsys jmp txjmp       transfer-of-execution jumper
5256  ff6f
5257  ff6f
5258  ff6f  4c ca fb  kvrese jmp vreset      netz ein/aus reset-vector
5259  ff72

5261  ff72  4c 33 fe  kipcgo jmp ipcgo       mon-befehl 'z' umsch. auf coprozessor
5262  ff75
5263  ff75
5264  ff75  4c 22 e0  kfunky jmp jfunky      vector für funktionstasten
5265  ff78
5266  ff78
5267  ff78  4c ab fc  kipcrq jmp iprqst      vector für ipc-anforderung
5268  ff7b
5269  ff7b
5270  ff7b  4c fb f9  kioini jmp ioinit      initialisierung ein-/ausgabe
5271  ff7e
5272  ff7e
5273  ff7e  4c 04 e0  kcint  jmp jcint       initialisierung bildschirm
5274  ff81
5275  ff81
5276  ff81  4c 00 f4  kalloc jmp tttop       allocation-routine
```

```
5279  ff84  4c a9 fb  kvecto jmp vector      ein-/ausgabe-vectoren lesen /
                                             schreiben
5280  ff87
5281  ff87
5282  ff87  4c a2 fb  kresto jmp restor      standard ein-/ausgabe-vectoren setzen
5283  ff8a
5284  ff8a
5285  ff8a  4c 60 f6  klkups jmp lkupsa      geräteadr. über sekundäradr. suchen
5286  ff8d
5287  ff8d
5288  ff8d  4c 78 f6  klkupl jmp lkupla      geräte- sekundäradr. über log. file#
                                             suchen
5289  ff90
5290  ff90
5291  ff90  4c 5a fb  kstmsg jmp setmsg      meldung des operat.syst. ausgeben
5292  ff93
5293  ff93
5294  ff93  6c 24 03  ksecnd jmp (isecnd)    addrl sekundäradr. nach listen
                                             ausgeben
5295  ff96
5296  ff96
5297  ff96  6c 26 03  ktksa  jmp (itksa)     addrl akt. sekundäradr. auf iec-bus
                                             ausgeben
5298  ff99
5299  ff99
5300  ff99  4c 78 fb  kmemtp jmp memtop      höchste speichergrenze
                                             lesen/schreiben
5301  ff9c
5302  ff9c
5303  ff9c  4c 8d fb  kmembt jmp membot      unterste speichergrenze
                                             lesen/schreiben
5304  ff9f
5305  ff9f
5306  ff9f  4c 13 e0  kscnky jmp jkey        tastatur lesen/in puffer schreiben
5307  ffa2
5308  ffa2
5309  ffa2  4c 74 fb  ksetmo jmp settmo      zeitüberwachung iec-bus ein
5310  ffa5
5311  ffa5
5312  ffa5  6c 28 03  kacptr jmp (iacptr)    addrl zeichen von iec-bus nach ac
5315  ffa8  6c 2a 03  kciout jmp (iciout)    addrl ac auf iec-bus ausgeben
5316  ffab
5317  ffab
5318  ffab  6c 2c 03  kuntlk jmp (iuntlk)    addrl untalk auf iec-bus ausgeben
5319  ffae
5320  ffae
5321  ffae  6c 2e 03  kunlsn jmp (iunlsn)    addrl unlisten auf iec-bus ausgeben
5322  ffb1
5323  ffb1
5324  ffb1  6c 30 03  klistn jmp (ilistn)    addrl listen auf iec-bus ausgeben
5325  ffb4
5326  ffb4
5327  ffb4  6c 32 03  ktalk  jmp (italk)     addrl talk auf iec-bus ausgeben
5328  ffb7
5329  ffb7
5330  ffb7  4c 4a fb  kreast jmp readst      ein/ausgabe-status lesen / schreiben
5331  ffba
5332  ffba
5333  ffba  4c 43 fb  kstlfs jmp setlfs      log. filenummer, geräte- und sek.adr.
                                             eintragen
5334  ffbd
5335  ffbd
5336  ffbd  4c 34 fb  kstnam jmp setnam      länge und adr. des filenamens
                                             eintragen
5337  ffc0
5338  ffc0
5339  ffc0  6c 06 03  kopen  jmp (iopen)     addrl open file-vector
5340  ffc3
5341  ffc3
5342  ffc3  6c 08 03  kclose jmp (iclose)    addrl close-vector
```

```
5345  ffc6  6c 0a 03  kchkin jmp (ichkin)     addrl open eingabekanal-vector

5348  ffc9  6c 0c 03  kckout jmp (ickout)     addrl open ausgabekanal-vector
5349  ffcc
5350  ffcc
5351  ffcc  6c 0e 03  kclrch jmp (iclrch)     addrl close kanal-vector
5352  ffcf
5353  ffcf
5354  ffcf  6c 10 03  kbasin jmp (ibasin)     addrl eingabe akt. kanal-vector
5355  ffd2
5356  ffd2
5357  ffd2  6c 12 03  kbsout jmp (ibsout)     addrl ausgabe akt. kanal-vector
5358  ffd5
5359  ffd5
5360  ffd5  6c 1a 03  kload  jmp (iload)      addrl lade vom file-vector
5361  ffd8
5362  ffd8
5363  ffd8  6c 1c 03  ksave  jmp (isave)      addrl save auf file-vector
5364  ffdb
5365  ffdb
5366  ffdb  4c 0e f9  ksttim jmp settim       interne uhr stellen
5367  ffde
5368  ffde
5369  ffde  4c e6 f8  krdtim jmp rdtim        interne uhr ablesen
5372  ffe1  6c 14 03  kstop  jmp (istop)      addrl prüfe stoptaste-vector
5373  ffe4
5374  ffe4
5375  ffe4  6c 16 03  kgetin jmp (igetin)     addrl zeichen aus tastatur-puffer
                                              vector
5376  ffe7
5377  ffe7
5378  ffe7  6c 18 03  kclall jmp (iclall)     addrl alle files schliessen-vector
5379  ffea
5380  ffea
5381  ffea  4c 79 f9  kudtim jmp udtim        interne uhr bedienen
5382  ffed
5383  ffed
5384  ffed  4c 10 e0  kscror jmp jscror       max.anzahl: spalten=xr, zeilen=yr
5385  fff0
5386  fff0
5387  fff0  4c 19 e0  kplot  jmp jplot        akt. x,y-pos. des cursors lesen /
                                              schreiben
5388  fff3
5389  fff3
5390  fff3  4c 1c e0          jmp jiobas      basisadr. i/o-gerät nach xreg/yreg
5391  fff6
5392  fff6
5393  fff6  ===> ac in 6509 execution register <===
5394  fff6
5395  fff6  85 00     kgbye   sta e6509       6509 execution register
5396  fff8  60                rts
5397  fff9
5398  fff9
5399  fff9  01                .byte $01
5400  fffa
5401  fffa
5402  fffa  ===> 6509 hardware-nmi-vektor <===
5403  fffa
5404  fffa  31 fb     hanmi   .word tabend     ind. sprung nmi-vector (von $fffa)
5405  fffc
5406  fffc
5407  fffc  ===> 6509 hardware-reset-vektor <===
5408  fffc
5409  fffc  97 f9     hares   .word start      system reset routine
5410  fffe
5411  fffe
5412  fffe  ===> 6509 hardware-irq-vektor <===
5413  fffe
5414  fffe  d6 fb     hairq   .word nirq       interrupt-routine (irq)
```

# Assembler

**Neu konzipierter Assembler für 6502, R65C02, 65802/65816**
**lauffähig auf AIM 65 und kompatiblen, ferner auf CBM 720**

Deutsche interaktive Bedienerführung, Fehlerhinweise in deutschem Klartext. Einstellbare Befehlssätze für die CPU-Typen mit Fehlerhinweis bei nicht implementierten Befehlen/Adressierungen. Schnelldurchgang mit alten Parametern möglich. Pass 2 kann mit neuen Ausgabeparametern zeitsparend wiederholt werden. Symboltafel alphabetisch geordnet. Symbole dürfen 9 Zeichen lang sein (Unterstreichungszeichen, Fragezeichen erlaubt), dadurch aussagefähige Label. Listbilder in diesem Heft. Übliche Eingabemedien des AIM mit bes. Floppy-Kanal (CBM 720: Floppy + Editor). E/A läuft über Sprungleiste am Programmbeginn. Im Texteditor können Kommandofiles benutzt werden, die Quelltexte von verschiedenen Eingabemedien verbinden. Codeablage und Offset-Ablage ein- und ausschaltbar. Erweiterte logische Operandenverknüpfung. - Eine vorhandene Symboltafel kann zusätzlichen Quelltexten zur Verfügung gestellt werden (nützlich bei engem Speicher). Speziell auf CBM 720 großzügiges Arbeiten: Quelltext im RAM resident bis 128 KB, 1 Bank nur für die Symboltafel (bis ca. 4600 Symbole), 1 Bank für Code-Ablage. Code kann aber auch an beliebige Stellen/Bänke abgelegt werden. Profi-Werkzeug mit Dokumentation. 8 KB-Programm für AIM in 2 Eproms, für CBM 1 Eprom für Bank 15-Erweiterung oder auf Diskette LP 8250. DM 350,-

In Vorbereitung: Assembler für PC 128 (bitte anfragen).

**Cross-Assembler für 6800/6802/6803/6303**

für die CPU-Typen von Motorola und Hitachi, für den AIM 65 und kompatible Systeme. Erzeugt aus den Mnemonics von Motorola Maschinencode. Zwei-Pass-Assembler mit gewohnten Komfort, Syntax jedoch nach 6502, 2 EPROMs mit Dokumentation DM 300,--

**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,--

**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

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



**Band 1:**

1. Einleitung
2. Was Sie benötigen
3. Starten von APPLE WORKS
4. Der Schreibtischmanager
5. Datenbank
6. Rechenblatt

A1 Anschluß der Festplatte ProFile
A2 APPLE II Easy Pieces Referenz
A3 Druckeranpassungen
A4 DOS 3.3 Konvertierungen
A5 APPLE WORKS Disketten sichern/kopieren
A6 Hilfsfunktionen nach Programmteilen

**Band 2:**

1. Einleitung
2. Was Sie benötigen
3. Starten von APPLE WORKS
4. Der Schreibtischmanager
5. Textbearbeitung
6. Datenfernübertragung

A1 . . . A6 wie Band 1, dazu:
A7 Modemkabel für APPLE IIe, IIc
A8 SuperSerialCard: Einstellung
A9 ASCII-Textdateien aus anderen Dateiformaten für ACCESS II
A10 DATEX-P20 F Verzeichnis
A11 Deutsche/Englische Menübilder von ACCESS II

**Von Botta/Lange/Zimmermann je 264 Seiten, Softcover, je DM 49,–**

**APPLE WORKS auf APPLE II, IIe, IIc:**
verwandelt APPLE-II-Computer in einen Elektronischen Schreibtischmanager mit:
Texterstellung . . . Edition, Briefarchiv, Ausdruck etc.
Datenarchivierung . . . Kontoführung, Buchhaltung etc. Formblattkalkulation . . . Bilanzen, VisiCalc-Dateien etc. Datenfernübertragung . . . Mailbox, Rechnerkopplung etc.

- ist ein erfolgreicheres Integrationspaket als LOTUS auf IBM PC!
- ist auf 1 MByte Speichererweiterungen Ihres APPLE II vorbereitet!
- erschließt Ihnen die Zukunftstechnik MAILBOX!
- ist ebenso einfach zu bedienen wie APPLE WRITER:
  Kein Befehlsstudium . . . Einfachste Menüführung . . . Sofortige Anwendbarkeit

**te-wi's APPLE WORKS SYSTEMBÜCHER 1+2 zeigen Ihnen:**

- Sämtliche APPLE WORKS Funktionen an Beispielen aus der Wirtschaft
- Das Wechseln zwischen Text/Rechenblatt/Datenarchiv/Dfü
- Umfassende Systeminformationen zu Dateikonvertierung, Druckeranpassung etc.

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

# MICRO MAG

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

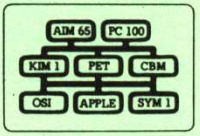


**Bezugsbedingungen:** Abonnement ab laufender Ausgabe für 6 Hefte DM 54,-- im Inland, bzw. DM 59,-- im Ausland (surface mail). Luftpostzustellung auf Anfrage. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu 4 Wochen vor deren Ablauf. - Nachliefermöglichkeiten siehe unten.

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 und 16 der Zeitschrift. Es erfolgt keine Neuauflage.

Lieferbar: 'Das Buch 7-13 des 65xx MICRO MAG' zu DM 42,-- sowie die Hefte 14, 15 und 17-45 zu DM 7,80/St. (ab 10 St. in einer Sendung: DM 6,--/St.).

**Mathe-ROM nach P. Rix** für FORTH und Einbindung in Assembler-Programme, mit Dokumentation DM 124,30.

**Assembler für 6502/65C02/65802/65816 (drei Befehlssätze!) sowie für MC6805 und für 6800/6802/6303 (drei Befehlssätze): Siehe im Heftinneren.**

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