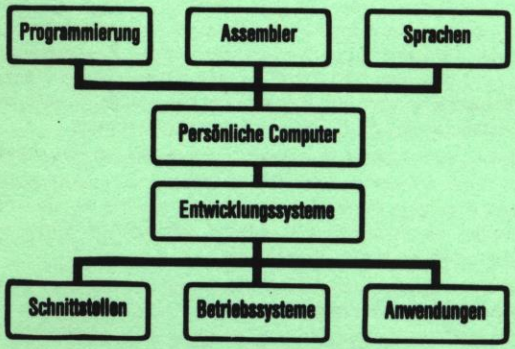
MICRO MAG

Vormals: 65xx MICRO MAG

DM 9,50 Nr. 37 Juni 1984



Inhaltsverzeichnis

Editorial

Im siebten Jahrgang trägt diese Zeitschrift jetzt den verkürzten Namen 'MICRO MAG', um zum Ausdruck zu bringen, daß die Themen wie bisher schon über den erfolgreichen Computerkreis 65xx hinausreichen. Denn wir stehen erst am Anfang der mikro-elektronischen Entwicklung, sowohl in der Hardware, wie auch in der Firm- und Software. - Nach den Jahren der Pionierzeit dürfen wir heute allerdings schon froh sein, Systeme mit leistungsfähigen Programmen beschaffen zu können, die auf Anhieb und ohne viel eigenes Zutun laufen und uns dann die Arbeit erleichtern.

In diesem Heft hat die Sprache FORTH einen neuerlichen Schwerpunkt. Es beginnt mit dem nachdenklichen Aufsatz von Andreas Goppold zum 'Paradigma der interaktiven Programmierung', der der jüngst in Hamburg gegründeten FORTH-Gesellschaft (s.u.) als Anregung und als Arbeitspapier zugeleitet wurde. Die ernsthaften Diskussionen in ihren Arbeitskreisen und die Vorbereittungen lassen schon jetzt erkennen, daß FORTH und ein möglicherweise auf die hiesigen Verhältnisse angepaßter Abkömmling sich auf dieser Seite des großen Ozeans deutlicher bemerkbar machen und die Unterstützung der FORTH-Gesellschaft haben werden.

Es geht nicht nur darum, daß FORTH als maschinennahe höhere Programmiersprache dichten und schnell ausführenden Code erzeugt und beliebig erweiterbar ist. Es geht vor allem auch darum, daß sein Compiler (im Gegensatz zu anderen Sprachen) anpaßbar ist. Damit sind für den Arbeitskreis 'Fünfte Dimension' die Ansatzpunkte gegeben, die besagte 'interaktive Programmierung' aus dem Kern der Sprache heraus weiter zu verfolgen.

Unvorhergesehene Ereignisse haben die Fertigstellung und den Versand dieses Heftes etwas verzögert. Der Herausgeber bittet um Verständnis.

Beiträge der Leser zu dieser Zeitschrift sind weiterhin gerne gesehen. Über die Themenwahl stimme man sich im Vorwege möglichst kurz ab.

○□○

FORTH-Gesellschaft gegründet: Ende April wurde mit Zentrum Hamburg die FORTH-Gesellschaft gegründet, einerseits als ein 'Chapter' der FORTH Interest Group (FIG), andererseits mit der Absicht, diese Sprache und evtl. einen auf die hiesigen Verhältnisse angepaßten Abkömmling hier zu fördern. Als Anlaufstelle für interessierte Betreiber wurde das Common Interface Alpha, Horst-Günter Lynsche benannt, Holstenstr. 191, 2000 Hamburg 50. Tel. Anfragen werktags von 12-19 Uhr, Tel.: 040 - 43 50 70. Inzwischen ist die erste Ausgabe des Kommunikationsorganes erschienen, die 'VIERTE DIMENSION'. Ihr Bezug kostet für zunächst 4 Ausgaben im Jahr DM 23. Die Mitgliedschaft in der Gesellschaft ist offen. Sie sieht sich als den Kristallisationskern für eine spätere bundesweite Organisation. Sie befördert mit ihren Informationen die Implementierung und den Gebrauch der Sprache FORTH sowohl für lernende und private Zwecke, als auch für den professionellen Einsatz. Die Mitgliedschaft ist offen, auch für fördernde Mitglieder und Institutionen. - Gleich bei der Gründung wurden daneben verschiedene Arbeitsgruppen gebildet, die einzelne weiterrreichende Projekte bearbeiten, z.B. die Gruppe 'Leibnitz' und die 'Fünfte Dimension'. Erstere bemüht sich um eine dem Deutschen angepaßte aktuelle Sprachversion, die andere .denkt voraus', um weitere Möglichkeiten zu erschließen.

Es finden regelmäßige Treffen am vierten Sonnabend im Monat statt.

Andreas Goppold, 2000 Hamburg 26

Das Paradigma der interaktiven Programmierung

Interaktivität ist heute das Schlagwort der Computer-Industrie. Computer haben sich von den batch-orientierten der Anfangsjahre über Time-Sharing zu den heutigen benutzerorientierten Systemen mit Video-Terminal, Maus, Berührbildschirm, Sprachein-/ausgabe und anderen interaktiven Methoden entwickelt. Die Anwendersoftware der heutigen Microcomputer wird immer weiter in Richtung Interaktivität ausgebildet. Die Kritereien für Interaktivität wandeln sich. War früher der Bildschirm und die Tastatur das einzige Medium, so unterstützen heutige Softwaresysteme auch die Maus und andere Hilfsmittel. Der Einsatz dieser Techiken, die, wie die Spracheingabe, sogar von einem speziellen 32-Bit Prozessor Gebrauch machen müssen (TI-PC), entwickelt sich zunehmend zu der entscheidenden Schwelle zwischen der alten Generation der 8-Bit Computer und der nächsten Generation der 16-Bit-Systeme. In dem Adreßbereich der 8-Bit-Systeme ist es nicht mehr möglich, den Code unterzubringen, den eine solche Benutzerunterstützung benötigt, und grafische Möglichkeiten, wie die von LISA, sind nur mit der Prozessorkraft eines MC 68000 und darüber zu bewältigen.

Programmieren ist nicht interaktiv

Wesentlich weniger interaktiv sind aber die Mittel, mit denen diese Softwaresysteme hergestellt werden: Die Programmiersprachen. Hier haben die neuen Hilfsmittel noch überhaupt keinen Eingang gefunden. Noch immer muß ein Programm zuerst mit einem Editor geschrieben und korrigiert und dann mit einem Compiler übersetzt werden, gelinkt werden. Und erst dann kann es ausgetestet werden. Die einzelnen Arbeitsschritte werden zwar interaktiv am Computer gestartet, wie das Editieren, das Compilieren etc., es sind aber soviele Zwischenschritte zwischen der Codierung des Programms und seinem eigentlichen Lauf, und die einzelnen Schritte dauern so lange, daß die ganze Verarbeitung eher Ähnlichkeit mit einem manuell gesteuerten Job-Control-System hat, als mit der vielbeschworenen Interaktivität des Personal Computer. Die Computerindustrie steht also vor der paradoxen Situation, daß die Anwendungssoftware der vierten Generation noch mit der Produktionssoftware der dritten geschrieben wird. Die Folgen für die Produktivität des Programmierers sind offensichtlich:

Wenn ein Programm erst einmal erstellt ist und als Objektcode vorliegt, dann kann man nicht schnell etwas an dem Objektcode ändern, um etwa einen Fehler zu verbessern, sondern man muß ganz zurück in den Editor, den Quellcode modifizieren und dann wieder den ganzen Zyklus von Editieren, Compilieren und Linken durchlaufen. Außerdem kann man einen Fehler nicht dadurch finden, daß man das Programm an einem beliebigen Punkt anhält, seine einzelnen Schritte durchspielt und genau die Fehlerstelle ausfindig macht. Es fehlt die Lupe, mit der man das Programm betrachten kann, oder, wie es fachlich heißt, die TRACE-Möglichkeit. Das Problem bei compilierten Sprachen ist, daß der Objekt-Code des Programms durch die Compilation keine Ähnlichkeit mehr mit der Quellversion hat. Es macht daher keinen Sinn, die einzelnen Schritte der Objektversion zu verfolgen.

BASIC hat in der Anfangszeit der Microcomputer eine ungeheuere Beliebtheit erlangt, weil es interaktiv ist. TRACE ist möglich (man denke an den Film TRON, der ein spekulativer Bericht eines TRACE in einem Supercomputer ist). BASIC hat hunderttausenden einen leichten, einfachen Einstieg in die Computerwelt verschafft. Sein Nachteil, daß es für größere Anwendungen zu langsam und wegen fehlender Strukturierungsmöglichkeiten zu unübersichtlich ist, fiel bei Anwendungen auf den frühen Microcomputern mit maximal 8 oder 20 K Speicherplatz nicht auf. So hat BASIC zwar das interaktive Programmieren eingeleitet, aber heute ist die professionelle Programmierung wieder auf die Methoden der Groß-EDV übergegangen, nicht viel anders als vor 10 Jahren.

Der Computer wird zu wenig zur interaktiven Unterstützung des Programmierens selber benutzt. Die Hauptarbeit der Programmiersprachen-Entwicklung ging in Richtung auf Modularität und logische Klarheit (PASCAL, MODULA, ADA). Die Methodik der batch-Arbeitsweise von Codieren, Compilieren und Linken wurde beibehalten. Zwar gibt es heute schon Ansätze einzelner Anbieter,

diese Situation zu verbessern, so etwa das MICRO FOCUS COBOL, dessen Animator-Funktion eine sehr komfortable interaktive Programmentwicklung zuläßt, aber im wesentlichen liegt dieses Feld noch brach.

Interaktive Unterstützung des Programmierens ist mit Compilersprachen wegen der starken Umwandlung des Quellcodes problematisch. Nur Sprachen, die einen interpretierten Zwischencode erzeugen (P-Code PASCAL oder CBASIC) sind für den interaktiven Einsatz besser geeignet. Es gibt ein Programmiersystem, das die interaktiven Eigenschaften in bisher nicht gekannter Konsequenz weiterentwickelt hat, und dieses System ist FORTH.

: FORTH INTERAKTIVES PROGRAMMIEREN ;

Hier ist als FORTH-Definition die Charakteristik des FORTH aufgezeigt. FORTH ist ein Programmiersystem, das speziell auf Interaktivität hin optimiert ist, und das durch seine offene Definition unbeschränkt weiter in diese Richtung hin ausgebaut werden kann.

Wichtig ist zu bemerken, daß das Element der Interaktivität nicht etwa ein Strukturelement der Sprach-Syntax ist, sondern auf Kriterien beruht, die der reinen Syntax-Betrachtung entgehen. Interaktivität ist ein Systemfaktor. Es ist mehr eine Potentialität, eine Eigenschaft, die entwickelt werden muß, und die deshalb in Diskussionen über die jeweiligen Vorzüge und Nachteile eines Programmiersystems auch oft unter den Tisch fällt. Daher kann man sagen: Der wichtigste Faktor und der wirklich wirksame Faktor an FORTH ist, daß FORTH über FORTH hinaus weiterentwickelt werden kann und werden muß. FORTH hat als Programmiersprache sicher seine Probleme, aber auf die Programmiersprache, also die Syntax, kommt es hier überhaupt nicht an, bei FORTH schon allein deswegen nicht, weil seine Syntax (im gravierenden Unterschied zu allen anderen Sprachen) als veränderbar definiert ist. FORTH enthält damit einige Prinzipien, die in dieser Mischung ein äußerst effektives Beispiel für eine interaktive Ausrichtung der Programmier-Methodologie bilden, wie sie bisher noch nicht formuliert worden ist.

Empirisches Programmieren

Vorteilhaft an FORTH ist zudem, daß es sehr leicht ist, die hier aufgeführten Argumente selbst nachzuprüfen. Der Empirie sind alle Wege offen. FORTH braucht keine 200.000 DM-Maschine, wie LISP (als einzige vergleichbare Sprache), um überhaupt zum Laufen gebracht zu werden, sondern steht auf jedem Home-Computer zur Verfügung. Alle Eigenschaften von FORTH lassen sich leicht nachprüfen, indem man sich an den Computer setzt und es ausprobiert.

Effizienzkriterien und Programmiermethodologie

Der Ursprung der Programmierung kann mit einem Wort charakterisiert werden: Institutionell. Computer waren bis vor ein paar Jahren sehr teure Maschinen, deren Nutzung nach sehr strengen Effizienzkriterien eingeteilt wurde. Die Strukturen der Programmiersprachen reflektieren diesen Ausgangspunkt. Früher wurden Programmiersprachen dahingehend entwickelt, daß der Programmierer so wenig Zeit wie möglich auf der Maschine verbrauchen mußte und so viel Arbeit wie möglich als Kopfarbeit dazu verwendete, um das Problem 'im stillen Kämmerchen' oder am Schreibtisch maximal vorzubereiten. Computerzeit war teuer, und es konnte sich kein Computerbetreiber leisten, kostbare Maschinenzeit mit Experimenten oder gar Herumspielen zu vergeuden. Demgemäß war auch die gesamte Programmierung auf dieses Kriterium der Effizienz hin strukturiert, und sie ist es auch heute noch, in einer Zeit wo sich die Kosten umgekehrt haben, und Hardware ein Zehntel der Softwarekosten ausmacht.

Das Aufkommen der Personal Computer brachte vor allem in der ersten Generation der 'Hacker' eine Zeit des 'Herumprobierens' am Computer, etwas, was man heute eher 'empirisches Programmieren' nennen müßte. Diese Methodik wurde aber abgelehnt, obwohl sie durchaus ihre Ergebnisse gebracht hatte, so etwa VISICALC, das Programm, das den APPLE-Computer in die Konzernetagen brachte, und damit den Microcomputern den Weg zur allgemeinen Akzeptanz bereitete.

Es ist erst in wenigen Arbeiten untersucht worden, daß die Produktivität der Programmierung durch empirische Vorhehensweise steigen kann (Weinberg, 'Psychology of Programming'), und interaktive Programmierung war bisher eher ein Anathema für alle Verfechter strukturierter Logik des Programmierens. Und genau das ist FORTH. Es ist ein empirisches Programmiersystem.

FORTH - Das Ein-Mann-Programmiersystem

Die Geschichte von FORTH unterscheidet sich so gravierend von der Entstehung der anderen Programmiersprachen, daß sie auch hier erwähnt werden soll. Sie ist charakteristisch für seine speziellen Qualitäten. FORTH ist als Ein-Mann-Produktivitätssystem entstanden. Es ist das Werk einer einzigen Person, Charles Moore. Später beschrieb er sich und das Ergebnis seiner Arbeit als 'der erste computerunterstützte Programmierer'. Er hatte ein System geschaffen, mit dem er in maximale Interaktion treten konnte oder, wie man heute sagen würde, mit dem er in eine geschlossene feedback-loop treten konnte. In der Biologie heißt dieses Konzept Symbiose oder Co-Evolution. Er trat mit seinem System in Interaktion und verbesserte es dynamisch. Er bemerkte, daß er mit seinen eigenen Fähigkeiten an seiner Kreation gewachsen war und gab sofort das feedback, indem er diese Eigenschaft des Systems weiterentwickelte oder wieder einschmolz. Eine solche Vorgehensweise war natürlich im Bereich der institutionellen Programmierung undenkbar. Diese Methodik ist aber die Wurzel von FORTH und sie erklärt alles, was FORTH heute ist.

Die Systemeigenschaften von FORTH sind in meinem Artikel 'FORTH - ein Programmiersystem ohne Grenzen' /1/ beschrieben. Aber diese Eigenschaften sind sozusagen nur die Mosaiksteine, die zusammen das Bild ergeben: Die Transparenz und die Möglichkeiten des TRACE, der Datenverkehr über den Stack, die Methodik der FORTH-Definition, die durch den Return-Stack möglich wird. - Hier soll mehr das daraus resultierende System-Bild betrachtet werden, vor allem in Hinsicht auf eine weitere mögliche Evolution dieser Konzepte. FORTH hat als wohl einziges Programmiersystem eine Möglichkeit der Evolution, da es eben nicht in seiner Syntax festgelegt ist. Und man muß nicht so ängstlich, wie etwa in COBOL, darauf bedacht sein, mit den Programmen, die vor 20 Jahren geschrieben wurden, noch kompatibel zu sein. In FORTH ist eine ganz andere Denkweise in dieser Richtung möglich: Es ist heute wesentlich billiger, sämtliche 100 Mrd.-Dollar-Investitionen in bestehenden COBOL-Programmen abzuschreiben, weil die Kosten der Wartung höher sind, als sämtliche Systeme von Grund auf neuzuschreiben, in einer neuen Methodologie, wie etwa der von FORTH.

Kriterien der Interaktivität

Interaktivität ist eine Eigenschaft, die graduell erscheint. Ein Videoterminal mit Tastatur ist interaktiv. Die Maus ist interaktiver. Wo liegen die Kriterien, und was ist maximale Interaktivität? Es ist klar, daß Interaktivität auf einem Zusammenspiel von Systemparametern beruht. Wichtige Faktoren sind hierbei die Geschwindigkeit und der Grad der Unterstützung. Frühere Systeme wurden speziell auf Effizienz in der Computerausnutzung optimiert und boten daher minimale Unterstützung für den Programmierer. Heutige Microcomputer-Systeme sind auf maximale Benutzerunterstützung ausgerichtet, werden aber noch so programmiert wie früher. Unterstützung kostet ihren Preis in Speicher und Maschinenzeit. Viele der heutigen Systeme drohen trotz der höheren Kapazität ihrer Prozessoren an den vielen Hilfsleistungen zu ersticken. APPLE's LISA ist dafür ein Beispiel. Software-Engineering nimmt eine neue Komponente an, die sich sonst eher in den Bereichen der Kunst aufhält: Ästhetik. Eine gelungene Mischung von Komponenten, wie Gewürze in einer Speise.

Der Bildschirm

Das heute wesentliche Feld der Interaktivität ist die Bildschirmdarstellung von Information. Heute wie vor zehn Jahren hat man zur Informationsdarstellung einen Bildschirm mit der Standard-Kapazität von 2000 alphanumerischen Zeichen zur Verfügung. Das ist dieselbe Fläche, wie bei IBM 3270-Bildschirmen. Trotz der gewaltig gestiegenen Prozessorleistung, die nun einem einzigen Bildschirm zur Verfügung steht, hat sich dieser Parameter nicht geändert. Es ist lediglich ein Graphik-Modus hinzugekommen mit etwa 500 mal 700 Punkten mit der Tendenz in Richtung 1000x1000 Punkte. Das ist aber nur ein statischer Wert. Um einen solchen Bildschirm aufzubauen, braucht der Prozessor ein bis zwei Sekunden. Mit schnell wechselnden Bildschirminhalten ist sogar der Hochleistungsprozessor MC 68000 der LISA überfordert..

Trotz der offensichtlichen Einschränkungen des Bildschirms ist dieser noch das schnellste Interaktionsmedium des Personal Computer. Eingaben sind notorisch langsam. Tastaturen sind immer

noch am schnellsten, aber schwer zu beherrrschen. Maus und Spracheingabe sind bequem aber erlauben nur sehr geringe Datenübertragungsraten.

Eine Theorie der Ikonen

Was ist der Effekt von Ikonen in der LISA oder in den Menues des MacIntosh? Eine Ikone ist ein graphisches Zeichen, also ein Zeichen, das aus einem vielfach größeren Zeichenvorrat ausgewählt wird, als es beim ASCII-Zeichensatz der Fall ist. Bei letzterem sind das eben nur etwa 100 Zeichen, Blockgraphik mit eingerechnet sind es etwa 200 Zeichen. Bei der Auflösung der LISA-Graphik sind es etwa 1000 Zeichen. Damit ist der Informationsinhalt einer Ikone auch wesentlich höher als der eines Namens (ebenso der nötige technische Aufwand).

Die Beziehung einer Ikone zu einem Ideogramm ist offensichtlich. Ein Ideogramm und eine Ikone drücken in einem Zeichen eine komplexe Bedeutung aus. In einer Ikone ist es eine bildliche Projektion, in einem Ideogramm mehr eine logische.

In FORTH ist ebenfalls ein ideographisches oder ikonisches Element enthalten. Bei FORTH ist es zwar kein graphisches Zeichen, das den Namen einer Definition darstellt, sondern ein Wort, das FORTH-Wort. Worte sind in gewissem Sinne Ideogramme, da sie ja ein Superzeichen bilden, also beim Lesen nicht als eine Folge von Buchstaben begriffen werden, sondern eben als Begriff.

Chinesische Ideogramme beziehen ihren Informationswert aus mehr Variablen als Worte der lateinischen Schrift. Im Chinesischen ist die Position maßgeblich, ob über, unter oder nebeneinander. Und auch die Nuancierung des Pinselstrichs ist bedeutsam. In der lateinischen Schrift gibt es nur noch die Folge, also die Aneinanderreihung von Buchstaben als Informationsträger. Nun sind FORTH-Worte zwar alphabetische Konstruktionen, aber ihre Positionierung ist relevant. Wer Anweisungen über die optimale Strukturierung einer FORTH-Definition liest, der merkt, welch' wichtige Bedeutung die Positionsinformation für die Lesbarkeit des Programms bekommen kann. - Es ist also aus dieser Sicht kein sehr großer Unterschied zwischen den Ikonen der LISA-Methodik und der FORTH-Programmierung. Die Gemeinsamkeit liegt aber woanders.

Der magische Screen

Das Wesentliche am Ikonengedanken ist zu erkennen, wenn man ein PASCAL-Programm, also einer modernen Programmiersprache, ansieht und im Vergleich dazu ein LISA-Menue. Der unterschied ist, daß in dem Programm die Information zwar (bei guter Programmierung) logisch strukturiert ist, aber sich diese Struktur über Seiten und Seiten im Programmlistung hinzieht. Ein LISA-Menue oder überhaupt ein Menue gibt die Information, die es enthält, auf einen Blick bzw. auf einem Screen des Video-Displays wieder. - Es gibt eine hierarchische Unterteilung von Menues, die nacheinander durchlaufen werden, wo immer ein kompletter Informationssatz mit einem Blick zu erfassen ist und eine Auswahl getroffen werden kann. Die logische Unterteilung der Information ist bei PASCAL wesentlich weiträumiger und bringt einen besonderen Nachteil mit sich, daß nämlich die relevante Information, die man sucht, über viele Seiten Programmlisting verteilt ist. Es erfordert einen höheren Aufwand, sie herauszufinden.

Es ist das Prinzip der FORTH-Programmierung, eine Unterscheidung in das WAS und WIE eines Programms zu machen und diese Unterscheidung auch räumlich zu trennen. Das heißt abstrakt: Eine Kontrollstruktur und eine Verarbeitungsstruktur. Auch in PASCAL ist eine solche Unterteilung zwar möglich durch Module und Blocks, aber sie ist mit der Verarbeitungstruktur vermischt.

Eleganz durch Einschränkung

In FORTH hat sich der Programmierer durch die Besonderheit des Forth-Virtual memory systems daran gewöhnt, seine Information in 'Screens' unterzubringen, Blöcken von 1000 Zeichen mit 16 Zeilen mal 64 Zeichen, noch ganz im Stil der ersten Dampfmicros, als der Computer noch maximal tausend Zeichen vom seinem sowieso sehr beschränkten Speicher für ein memory mapped Video erübringen konnte. — Von diesen Screens waren aus Gründen der Übersichtlichkeit noch etwa 2/3 als Füllraum freizulassen, so daß nur etwa 300 Zeichen zur Darstellung der relevanten Information übrig blieben. Dies hat die Programmierung in FORTH in eine sehr ausgeprägte Strukturierung geführt (Nebenfaktor: Bei den frühen Microcomputern gab es auch keinen Drucker, bestenfalls ein ausrangiertes Teletype, so daß man tunlichst nicht viele Programmlistings machen wollte). - Diese Struktur heißt: Komprimiere die relevante Information auf einen Screen von 300 Zeichen und sorge dafür, daß Du dein Problem lösen kannst, ohne das Listing all' der anderen

Screens zur Verfügung zu haben. Eine außerordentlich wirksame Methode des Trainings im strukturierten Programmieren. Sie führte dazu, daß man nicht nur strukturiert programmierte (GOTO gibt es in FORTH nicht), sondern auch noch, daß man hierarchisch programmieren lernte.

Hierarchische Programmierung

Das heißt: Das Problem mußte nicht nur maximal in Module unterteilt werden, sondern diese Module mußten auch noch in ihren Kommunikationsanforderungen in 300 Zeichen erschöpfend beschrieben werden. Bei Programmiersprachen, die mit Variablen arbeiten, ist dies einfach nicht machbar, aber mit dem Stack-Konzept war es möglich. Zudem war nur mit dem Return-Stack die Methode der geschachtelten Definitionen praktikabel. So fügten sich also viele verschiedene Bestandteile, von denen die meisten Einschränkungen waren, zu einem Bild.

Hierarchische Programmierung heißt: Zu der Modularisierung in Funktionen auch noch eine Unterteilung in logische Ebenen von Funktionen. Eine logische Typisierung. Allein Funktionen gleichen logischen Typus' können miteinander in Kommunikation treten. Wenn man keine Variablen verwendet ist nur der Stack als Datenverkehrsmedium zwischen FORTH-Worten (Unterprogrammen, ähnlich C-Funktionen) vorhanden, die in derselben FORTH-Definition vorkommen, die also auf derselben hierarchischen Ebene stehen. M.a.W.: Es ist ein orthogonales Kommunikationsmodell.

Fenster und Ikonen

Eine Hierarchie dieser Art ist es, auf die sich heute die Fenster-Software hin entwickelt. Ein Fenster und eine Ikone stehen in gradueller Beziehung. Wenn man ein Fenster maximal verkleinert, wird es zur Ikone, bevor es völlig verschwindet. Natürlich ist heutige Fenstersoftware noch nicht dort angelangt, und ein Fenster schrumpft schon lange vorher zu einem völlig nichtssagenden Stück Bildschirm zusammen, was leider bei all' der Euphorie über diese Methodik noch nicht aufgefallen ist. Zum Beispiel hat es wenig Sinn, in einem Texteditor mehr als ein split-screen Schema einzuführen, weil kleinere Fenster nur in den wenigsten Fällen noch eine Aussagekraft haben. Betrachtet man aber den logischen Zusammenhang von Ikone und Fenster, dann wird klar, in welche Richtung die Entwicklung gehen muß und welchen Weg sie noch vor sich hat.

Ikonen und typologische Ebenen

Die Ikone enthält die Information, die auf der entsprechenden typologischen Ebene der Kontrollstruktur relevant ist. Dies rein theoretisch zu formulieren, ist sehr schwer. Aber wenn man in FORTH eine Definitionsstruktur entwirft, dann kommt man bei der Auswahl der Namen für die einzelnen FORTH-Definitionen sehr schnell darauf, auf was es ankommt. Vorausgesetzt natürlich, man ist in der Lage, 'gute' FORTH-Programme zu schreiben, eine Kunst, über deren Essenz sich die Experten dieser Sprache lange und ausgiebig streiten. Es ist eine Hierarchisierung von Informtionsebenene und eine Spezifizierung in Bezug auf das Level in der Kontroll- oder Verarbeitungsstruktur, unter der Maßgabe, daß es in einem Blick erfaßbar sein muß.

Das Problem ist also das der Informationsverdichtung, eben der hierarchischen Informationsstrukturierung. Was muß an Darstellung noch vorhanden sein, wenn man die Ebenen der Abstraktion herauf- und hinuntersteigt? – Andere Elemente der Interaktivität sind die Möglichkeiten des TRACE und des interaktiven Debugging, die hier nicht eingehender beschrieben werden. Um das Bild der interaktiven Programmierung noch abzurunden, soll hier ein Auszug aus einem Leserbrief in BYTE Magazine in der Kolumne von Jerry Pournelle gebracht werden (BYTE 5/84, S. 428, 431):

"FORTH wird compiliert wie Modula-2 oder PASCAL aber in FORTH erfolgt die Compilation direkt nach der Definition des Wortes, und das neue Wort wird einfach durch seinen Aufruf aktiviert. ...
Die Direktheit der Compilation und die Kürze der Definition, die damit ermöglicht wird, versetzt den Programmierer in die Lage, mit dem Programm zu interagieren, während es geschrieben wird. In Situationen, in denen das feedback der Aktionen im selben Augenblick erfolgt, kann man einen Bewußtseinsstand erreichen, in dem die Anstrengungen und die Ergebnisse sich gegenseitig beeinflussen und das Bewußtsein mit dem Prozeß verschmilzt, so daß man in der Arbeit 'aufgeht' - ein maximal produktiver Bewußtseinszustand. Dieser Bewußtseinszustand ist den meisten Künstlern vertraut. In ihrer Arbeit mit Pinsel, Bleistift oder Ton sind sie in direktem Kontakt mit ihrem Medium. Sie können etwas ausprobieren, sich ansehen, was herauskommt, und das Resultat,

den nächsten Schritt beeinflussen. Es ist ein vertrautes Gefühl für Schauspieler, für Sportler, für alle, die ein direktes feedback bekommen, während sie arbeiten.

Den meisten Programmierern ist das unbekannt, weil die meisten Programmiersprachen in einer Weise strukturiert und implementiert sind, daß eine Barriere für das direkte feedback existiert, eine Mauer von Zeit und Prozeduren zwischen der Idee und dem Resultat. ..."

Mit diesen Sätzen ist das Erlebnis des interaktiven Programmierens wohl angemessen beschrieben. Seine positiven Auswirkungen können zum Glück sehr einfach und empirisch bewiesen werden: Man nehme einen Home-Computer, lade das FORTH-System und fange an zu programmieren.

Interaktive Programmierung und selbstorganisierende Systeme

Dies sind also einige Ansätze zu dem Paradigma der interaktiven Programmierung. Das Besondere an diesem Prinzip ist seine leicht zu zerstörende Balance. Es ist eine Balance zwischen vielen verschiedenen und widersprüchlichen Faktoren. Diese Balance zeichnet Systeme aus, die wir allgemein als Leben bezeichnen, also die Welt der Organismen. Viele der Konzepte, die man zur Beschreibung lebendiger Systeme gefunden hat, lassen sich auch in FORTH wiederfinden. Diese Konzepte kann man auch mit dem Oberbegriff belegen: Selbstorganisierende Systeme. Sie sind sozusagen der Stein der Weisen in unser heutigen immer komplexer werdenden technologischen Welt. Sie sind das Ziel der riesigen Forschungsaufwendungen der Artificial Intelligence und der Projekte zum 5th Generation Computer.

Natürlich gibt es heute schon perfekt selbst-organisierende Systeme einer Komplexität, die 5 bis 6 Größenordnungen größer ist, als alle vom Menschen geschaffenen Maschinerien: Die Natur mit ihren Organismen. In der organischen Natur ist alles sozusagen 'natürlich' geregelt. Die Prinzipien dieser Regelung sind also der Schlüssel zur Schaffung technischer selbst-organisierender Systeme. Ein sehr wichtiger Beitrag hierzu ist Douglas Hofstadters Buch: 'Gödel, Escher, Bach', in dem er die Verbindungen der Computer und der natürlichen Systeme aufzeigt.

FORTH und das Grundprinzip der selbst-organisierenden Systeme

Ein Kernpunkt in Hofstadters Buch ist die Selbst-Referenz, ein Grundprinzip, das sich in allen selbst-organisierenden Systemen wiederfinden läßt. Es bedeutet, daß alles sich durch Rückbezug von einfachen Kopien von sich selbst aufbaut. Anders gesagt: Es gibt kleine, universelle Bausteine, die sich regelmäßig zu immer größeren Komplexen zusammenfügen. Von diesem Punkt ist es dann kein weiter Weg mehr zu FORTH. Hofstadter bescheibt die Regelmäßigkeit dieser Ordnung mit den Worten 'Schrödingersche periodische Kristalle' und bringt als Beispiele einer solchen Ordnung Fractals und die Musik von Bach. Beide Strukturen sind aber in ihrer Essenz Stack-Konstrukte. Damit die die Stack-Struktur von FORTH eine Widerspiegelung oder ein Morphismus der Organisationsprinzipien natürlicher Systeme.

Das Grundprinzip: feedback

Dieses von Hofstadter auch Selbst-Referenz genannte Prinzip ist heutzutage die Geheimformel in allen kybernetischen Patentrezepten. Es ist das Grund-Ordungsprinzip der organischen Natur. Feedback ist das Element, mit dem der Programmierer in der Interaktion mit seinem Werkzeug sein Werkzeug erst schafft, es immer neu definiert, es seinen Vorstellungen anpaßt. Dies ist das Werkzeug der Co-Evolution. Es ist ebenso das Kriterium eines neuen wissenschaftlichen Paradigmas, das nicht mehr objektivistisch, newtonisch und karthesisch ist, sondern relativistisch und synergistisch. Es ist das Paradigma des Wissenschaftlers und seines Studiums als geschlossenes System, in dem der Forscher nicht das Objekt erforscht, sondern seine Interaktion mit dem Objekt, und erlebt, wie das 'Objekt' zum Partner und Lehrer wird. Ein Wachstumsprozess.

Weiteres Prinzip: Duale Komplementarität

Ein weitereers Erfolgsrezept aller selbst-organisierenden Systeme ist die duale Komplementarität. Hochentwickelte Organismen haben immer zwei Nervensysteme, einerseits ein autonomes, lokales, das vegetative Nervensystem, und andererseits ein motorisches, hierarchisches, das Zentral-Nervensystem. – Ein Beispiel in FORTH sind die beiden Stacks. Der Datenstack und der Kontrollstack erfüllen dual komplementäre Funktionen. Der Datenstack ist lokal, dezentral und auf die Produktionsstruktur bezogen. Der Kontrollstack ist hierarchisch und mit der Kontrollstruktur verbunden. – Ein höher organisiertes System kann nicht ohne beide Komponenten funktionieren. .

Martin Albrecht, 4350 Recklinghausen

BASDIS 1.0

AIM 65 BASIC-Erweiterung

Inhalt:

1. Einleitung

Das BASIC des AIM 65 ist eine abgemagerte Version des Microsoft-BASIC, wie man es von den Commodore-Rechnern her kennt. Die vorliegende Erweiterung ergänzt die fehlenden Befehle die zur Dateibearbeitung mit Floppydisk erforderlich sind. Mit dem neuen Befehl 'CBM' sind zusätzlich CBM-Programme lad- und speicherbar. Das Diskdirectory (Inhaltsverzeichnis) kann einfach als BASIC-Programm geladen werden. Die beim AIM-BASIC nicht implementierte ATN Funktion wurde nachkodiert.

Das Programm stützt sich auf den AIM DISKMON Monitor aus MICRO MAG Nr. 35. Mit wenigen Änderungen ist es sicher auch mit den Routinen zum IEEE 488-Bus lauffähig. Zum Verständnis soll hier noch einmal kurz auf die wichtigsten DISKMON-Routinen eingegangen werden:

Alle Routinen, die Befehle an den Bus übergeben, verlangen die Gerätenummer im Accu und die Sekundäradresse im Y-Register. FOPEN eröffnet ein File. Bei gelöschtem Carrybit (C=0) wird das File mit einem Filenamen eröffnet, bei gesetztem Carry (C=1) ohne Filenamen. Die Variablen LAE, TXT und TXT+1 speichern die Länge und die Adresse des Filenamens. DOLIST und DOTALK senden unter aktivem ATN ein 'LISTEN' bzw. 'TALK' und die Sekundäradresse an das Gerät. Mit LDAIEC wird ein Byte vom Bus in den Accu eingelesen, mit STAIEC vom Accu auf den Bus geschrieben. UNLIS und UNTALK beenden die Listen- oder Talk-Sequenz. Die Variable STATUS dient der Beobachtung des Bus-Status. Sie ist nach jedem Lese- oder Schreibvorgang gesetzt und nimmt dabei folgende Werte an:

hex 40	Fileende, das letzte Byte wurde gelesen
hex 42	es wurde hinter dem Fileende gelesen
hex 02	Read-Timeout: Busfehler
hex 03	Write-Timeout: Busfehler
hex 00	alles in Ordnung

Bei IEEE 488-Bus ist zur Beurteilung des Filestatus die Leitung EOI heranzuziehen.

2. Inbetriebnahme

Das BASIC ist wie üblich mit Taste (5) zu starten. Nachdem es initialisiert ist, wird es verlassen und die BASIC-Erweiterung geladen. Mit Taste (6) geht es zurück zum BASIC. Vor Beginn der Arbeit muß ein CLEAR-Befehl gegeben werden. – Die Erweiterung beginnt ab Adresse hex 212 und beschränkt den Arbeitsspeicher des BASIC nicht nach oben. In der Seite Null werden keine Speicherplätze belegt.

3. Die neuen Befehle

3.1 ATN

SYNTAX: ATN (Argument)

ATN bildet den Arcustanges des eingegebenen Argumentes.

3.2 CLEAR

Dieser Befehl arbeitet in der gewohnten Weise, setzt aber zusätzlich die Zahl der offenen Files auf Null und den CBM-Modus (s.u.) zurück. Zu Beginn der Arbeit muß daher ein CLEAR gegeben werden.

3.3 OPEN
SYNTAX: OPEN Filenummer, Gerätenummer, Sekundäradresse oder:
OPEN Filenummer, Gerätenummer, Sekundäradresse, "Filename"
OPEN eröffnet ein File auf dem IEC-Bus. Filenummer, Gerätenummer und Sekundäradresse werden in die Liste der Files eingetragen. Es dürfen maximal 10 Files zur gleichen Zeit geöffnet sein. Bei Überschreitung der Höchstzahl erscheint ein OV-Error. Filenummern zwischen 0 und 255 sowie Gerätenummern und Sekundäradressen zwischen 0 und 15 sind gültig. Die Angabe eines Filenamens ist optional.

Beispiele: OPEN 3,8,15 eröffnet den Kommandokanal (15) der Floppy (8) als File-Nr. 3.
OPEN A,8,4,B$ eröffnet ein File mit der Nummer aus Variable A auf Gerät 8 und Kanal 4 mit dem Filenamen aus B$.

3.4 CLOSE
SYNTAX: CLOSE Filenummer
CLOSE schließt ein File auf dem IEC-Bus. Steht das File nicht in der Liste der Files, erscheint ein FC-ERROR. Das File wird aus der Liste entfernt.

Vorsicht: Wird der Kommandokanal der Floppy geschlossen, dann schließt diese selbtätig alle offenen Floppyfiles, während sie im BASIC noch als geöffnet geführt werden. Auf diese Weise lassen sich im Fehlerfall die noch offenen Files auf der Floppy schließen.

3.3 PRINT#
SYNTAX: PRINT# Filenummer, Variable oder "Text"
Dieser Befehl arbeitet genau wie das normale PRINT, jedoch auf das angegebene File. Dieses muß offen sein, sonst erscheint ein FC-ERROR. Das Zeichen '#' muß ohne Leerraum direkt hinter dem Befehl stehen. Das gilt auch für alle anderen Befehle mit diesem Zeichen. Nach der Filenummer folgt ein Komma. Die Argumente dahinter sind wie bei einem normalen PRINT zu geben.

Beispiel: PRINT# 3,A,"Text",B$ schreibt den Inhalt der Variablen A, den Text 'Text' und den Inhalt der Variablen B$ in das File 3.

3.5 INPUT#
SYNTAX: INPUT# Filenummer, Var., Var., ... oder:
INPUT# Filenummer, "Text"; Var., Var., ...
Der Befehl arbeitet wie das normale INPUT von der Tastatur. Die Eingabe erfolgt aus dem angegebenen File, welches geöffnet sein muß, sonst FC-ERROR. Sind keine Daten mehr vorhanden, erscheint ein OD-ERROR. Es darf ein Ausgabetext in Anführungszeichen gegeben werden. Er wird jedoch ebenso wie das gewohnte '?' nicht ausgegeben.

3.6 GET#
SYNTAX: GET# Filenummer, Var$, Var$, ...
Dieser Befehl liest ein Zeichen aus dem angegebenen File in eine Stringvariable ein. Es können mehrere Variable hintereinander angegeben werden. Das File muß vor dem Einlesen geöffnet sein, sonst erscheint ein FC-ERROR. Ist die Variable nicht vom Typ String, dann erscheint ein TM-ERROR. Wird beim Lesen das Ende des Files überschritten, dann wird nur ein Leerstring "" übergeben. Damit läßt sich das Fileende feststellen. Die bei CBM-Rechnern vorhandene Variable ST, die den Filestatus enthält, steht hier nicht zur Verfügung. Der Status ist aber in gleicher Weise mit PEEK(42007) abrufbar.

Beispiel: GET#5, A$,B$,C$ liest drei Zeichen von File Nr. 5 in die genannten drei Variablen ein.

3.7 LOAD#
SYNTAX: LOAD# Gerätenummer,"Filename"
Mit diesem Befehl wird das benamte Programm vom angeführten Gerät in den Speicher geladen. Der Filename muß immer angegeben werden, sonst erscheint ein SN-ERROR. Ein etwa vorhandenes altes Programm im Speicher wird beim Laden überschrieben und geht verloren. Ist das geladene Programm größer als der Speicher, erscheint ein OM-ERROR. Das Mischen zweier Programme, wie man es vom AIM her gewohnt ist, ist leider nicht möglich. Mit LOAD#8,"$" wird das Directory der Floppy (8) wie ein Programm eingelesen. Es kann mit LIST ausgegeben werden.

Beispiel: LOAD#8,A$ lädt das Programm mit dem in A$ enthaltenen Namen von Gerät Nr. 8.

3.8 SAVE#
SYNTAX: SAVE# Gerätenummer,"Filename"
Es wird das im Speicher stehende Programm auf dem bezeichneten Gerät unter dem Filenamen gespeichert. Fehlt dieser, dann erscheint ein SN-ERROR. Anwendung wie bei 3.7.

3.9 CBM
Der AIM 65 und die Commodore-Rechner arbeiten mit einem Microsoft-BASIC. Sie setzen die Befehle in Tokens von einem Byte um, wenn sie eine Programmzeile übernehmen. Die Tokens der Maschinen sind jedoch verschieden. Gibt man nun vor einem SAVE#-Befehl den Befehl CBM, dann werden die AIM-Tokens in CBM-Tokens umgewandelt und abgespeichert. Das gleiche gilt bei LOAD#, es werden beim Laden CBM-Tokens in solche des AIM 65 umgesetzt. So lassen sich Programme mit Commodore-Rechnern austauschen

In den meisten Fällen muß die fremde Software noch überarbeitet werden, weil das CBM-BASIC einen umfangreicheren Befehlssatz hat. Beim CBM werden Steuerzeichen-Codes in Anführungszeichen hinter PRINT verwandt, die vom AIM als Tokens interpretiert werden. Dadurch entstehen merkwürdige Statements. Auf die folgenden Besonderheiten ist zu achten:

a) Laden von CBM-Programmen in den AIM

CBM		AIM
CLR, CMD, SYS, OPEN, CLOSE	wird zu	REM
PRINT#		PRINT ohne #
INPUT#		INPUT, ebenso ohne #

Steuerzeichen in Anführungszeichen sind anzupassen (s.o). Die Abfrage der CBM-Variablen ST ist durch PEEK(42007) zu ersetzen.

b) Laden von AIM-Programmen in den CBM
Die Befehle CLOSE, OPEN und CBM werden beim AIM nicht als Tokens gespeichert und daher nur als Text übergeben. Das CBM entfällt natürlich für CBM-Rechner. OPEN und CLOSE müssen neu eigegeben werden. Das gilt auch für INPUT, INPUT# und PRINT#, denen jeweils noch ein # angehängt wird. Die Befehle SAVE# und LOAD# sollten in Programmen nicht vorkommen, da die Syntax hier unterschiedlich ist. Zur Abfrage des File-Endes beim Befehl GET# ist die Variable ST zu verwenden.

4. Fehlerbedingungen

Bei fehlerhafter Schreibweise der Befehle erscheint wie üblich ein SN-ERROR. Ebenso, wenn Filenummer, Gerätenummer oder Kanalnummer größer als 255 sind. Steht ein mit PRINT#, INPUT# oder GET# angesprochenes File nicht in der Liste, wird ein FC-ERROR ausgegeben. Versucht man mehr als 10 Files zu öffnen, erfolgt ein OV-ERROR. Wird bei GET# oder beim Filenamen bei OPEN, LOAD# und SAVE# der falsche Variablentyp angegeben, dann erscheint ein TM–ERROR. INPUT# und GET# dürfen nur innerhalb von Programmen benutzt werden, sonst wird ein ID-ERROR angezeigt.

Das Einlesen und Ausgeben der Bytes bei INPUT# und PRINT# geschieht über die User Input- und Outputvektoren. Der ursprüngliche Vektorinhalt wird bei der Befehlsausführung gerettet und dannach zurückgespeichert. Tritt ein Fehler beim Befehl PRINT# auf, dann wird die Fehlermeldung noch über den User-Vektor ausgegeben und erscheit nicht auf dem Display. - Bei einigen Floppy-Fehlern erfolgt ein Rücksprung in den Monitor. Schlimmstenfalls bleibt man in einer BASIC-Schleife hängen.

5. Wie es funktioniert

Bei der Initialisierung des BASIC wird die Routine, die die Zeichen aus dem BASIC-Programm list, in den RAM-Bereich hex BF bis D0 kopiert. An der Stelle C8 wird ein Sprung auf die BASIC-Erweiterung eingesetzt. Diese stellt zunächst fest, ob das gelesene Zeichen ein Token (hex 80 bis C5) ist. Findet sie ein Token für PRINT, INPUT, GET und LOAD oder SAVE, dann wird auf das mögliche Folgezeichen # abgefragt. Wenn es vorhanden ist, dann wird das entsprechende Unterprogramm angesprungen. Zur Erkennung von OPEN, CLOSE und CBM fragt die Erweiterung ab, von welcher Adresse das Programm aufgerufen wurde. Kam der Aufruf von BE09, dann hat

der Interpreter einen Variablennamen mit mehr als zwei Zeichen entdeckt. Der gefundene Name wird mit OPEN, CLOSE und CBM verglichen. Bei einer Übereinstimmung der Textkette wird in das entsprechende Unterprogramm verzweigt.

6. Erweiterungen

Weitere Befehle lassen sich leicht einbauen. Der Name eines neuen Befehls muß in die Liste 'TEXT' eingetragen werden, wobei beim letzten Zeichen Bit 7 zu 1 zu setzen ist. Die Wortlänge ist in LAENGE und die Adresse des ausführenden Programms in ADR einzutragen.

Literatur:

/1/ AIM-65 BASIC Language Programm Listing, GWK, Herzogenrath, 1979
/2/ Michael Konz: AIM-65 BASIC-ERweiterung, Funkschau, Heft 2/1981
/3/ Herwig Feichtinger: AIM liest CBM-Kassetten, mc, Heft 3/1982
/4/ Peter Treytl: Ausgabe-Umleitung beim AIM-BASIC, mc, Heft 11/1982

```
------------------------------------------------------------
BASDIS 1.0: AIM-65 Basic-Erweiterung

0000          ;(C) Martin Albrecht, 10.2.1984
0000
0000
------------------------------------------------------------
DISKMON Variablen und Unterprogramme

0000 LAE            =$A7            ;Länge des Filenamens
0000 TXT            =$A8            ;Position des Filenamens
0000 STATUS         =$A417          ;IEC-Busstatus
0000
0000 UIN            =$108           ;User Input Vektor
0000 UOUT           =$10A           ;User Output Vektor
0000 INFLG          =$A412          ;Eingabegerät
0000 OUTFLG         =$A413          ;Ausgabegerät
0000
0000 DOLIST         =$FFF1          ;zum Zuhören auffordern
0000 DOTALK         =$FEEF          ;zum Senden auffordern
0000 FCLOSE         =$F17D          ;File schließen
0000 FOPEN          =$F32C          ;File öffnen
0000 LDAIEC         =$F106          ;ein Byte vom File einlesen
0000 STAIEC         =$F0EF          ;ein Byte auf File ausgeben
0000 UNLIS          =$E597          ;Bus frei nach LISTEN
0000 UNTALK         =$E587          ;Bus frei nach TALK
0000
------------------------------------------------------------
Basic Unterprogramme

0000 NEWCH          =$BF            ;ein Zeichen lesen
0000 OLDCH          =$C5            ;altes Zeichen nocheinmal lesen
0000 GENERR         =$B259          ;Error ausgeben, Nr. in X
0000 FCERR          =$BF87          ;FC-ERROR ausgeben
0000 OVERR          =$C6BD          ;OV-ERROR ausgeben
0000 OMERR          =$B257          ;OM-ERROR ausgeben
0000 TMERR          =$BB7A          ;TM-ERROR ausgeben
0000 SNERR          =$BCD0          ;SN-ERROR ausgeben
0000 NUM255         =$C4D9          ;Zahl < 255 einlesen
0000 NUMOR          =$C4DC          ;Zahl < 255 ohne NEWCH einlesen
0000 VALEXP         =$BB7F          ;Wert eines Ausdrucks berechnen
0000 STRLAD         =$C3E4          ;Stringlänge und Position berechnen
0000 BPRINT         =$B8A9          ;Basic PRINT Funktion
0000 BINPU          =$B9BE          ;Basic INPUT Funktion
0000 BBLOOP         =$B5C8          ;Basic Befehlsschleife
0000 WSTART         =$B27F          ;Basic Warmstart
0000 DIRIND         =$C0E4          ;Direkt/Programm unterscheiden
0000 PACKEN         =$B329          ;Zeilenanfangsadressen neu setzen
0000 FINDV          =$BDE4          ;Variable suchen
0000 STRPTR         =$C1CB          ;Stringpointer setzen
0000 STOSTR         =$B849          ;String speichern
0000 BCRLF          =$B900          ;CRLF ausgeben
0000
------------------------------------------------------------
Erweiterung anbinden

0000                *=$C8
00C8          4C1202 JMP ERWEI     ;Erweiterung einschleifen
00CB
------------------------------------------------------------
```

Sicherheitsroutine anbinden

```
00CB                *=0
0000        4C9D06  JMP SAFETY
0003
```

Hauptprogramm

```
0003                *=$212
0212 ERWEI  8DB707  STA ATEMP       ;Accu retten
0215        8EB907  STX XTEMP       ;X-Reg. retten
0218        301C    BMI TOKTST      ;teste Tokens wenn neg.
021A TT2    68      PLA             ;woher kam der Aufruf ?
021B        AA      TAX
021C        C90B    CMP #$B         ;von $BE09 ?
021E        D007    BNE B1
0220        68      PLA
0221        C9BE    CMP #$BE
0223        F043    BEQ WORDS       ;ja, teste neue Befehle

0225        48      PHA             ;Stack wiederherstellen
0226        8A      TXA
0227 B1     48      PHA
0228 BR     AEB907  LDX XTEMP       ;X-Reg. und Accu wiederherstellen
022B        ADB707  LDA ATEMP
022E        C93A    CMP #':'
0230        B003    BCS B2
0232        4CCC00  JMP $CC
0235 B2     60      RTS
0236
0236        ;vergleiche gefundenen Token mit der Liste
0236 TOKTST C99A    CMP #$9A        ;teste 'CLEAR'-Token
0238        D00A    BNE TOKL        ;nein
023A        A900    LDA #0          ;setze Anzahl Files
023C        8DBC07  STA MAXFIL      ;bei 'CLEAR' auf Null
023F        8DBF07  STA CBMFLG      ;CBM-Modus zurück
0242        F0D6    BEQ TT2
0244
0244 TOKL   A204    LDX #TOKNUM     ;Anzahl Tokens
0246 NXTTOK DDEC06  CMP TOKENS,X    ;vergleiche
0249        F005    BEQ ISTOK       ;ist vorhanden
024B        CA      DEX
024C        10F8    BPL NXTTOK      ;Schleife
024E        30CA    BMI TT2
0250
0250        ;ist Token, teste ob ein '#' folgt
0250 ISTOK  A001    LDY #1
0252        B1C6    LDA ($C6),Y     ;teste auf '#'
0254        C923    CMP #'#'
0256        D0C2    BNE TT2         ;nein
0258        8A      TXA             ;hole Index
0259        0A      ASL A           ;*2
025A        AA      TAX             ;als Index in die Adreßtabelle
025B        BDF106  LDA BEFADR,X    ;Adresse Low
025E        856A    STA $6A
0260        BDF206  LDA BEFADR+1,X  ;Adresse High
0263        856B    STA $6B
0265        6C6A00  JMP ($6A)       ;Befehl ausführen
0268
0268        ;vergleiche Befehlstext mit der Liste der Befehle
0268 WORDS  48      PHA             ;rette Returnadresse
0269        8A      TXA
026A        48      PHA
026B        A5C6    LDA $C6         ;setze Pointer $6A,$6B
026D        38      SEC
026E        E902    SBC #2          ;2 Zeichen zurück
0270        856A    STA $6A
0272        A5C7    LDA $C7
0274        E900    SBC #0
0276        856B    STA $6B
0278        A200    LDX #0          ;Index in Befehlsliste
027A        A000    LDY #0          ;Index in Programmtext
027C        8CBA07  STY CNTR        ;Befehlswort-Zähler auf 0
027F        F002    BEQ COMP        ;zum Vergleich
0281 INCZEI E8      INX
0282        C8      INY
0283 COMP   B16A    LDA ($6A),Y     ;hole Zeichen
0285        38      SEC
0286        FDFB06  SBC TEXT,X      ;Befehlszeichen subtrahieren
```

```
0289          F0F6    BEQ INCZEI        ;war gleich, weiter
028B          C980    CMP #$80          ;letztes Zeichen OK?
028D          F012    BEQ MATCH         ;Befehl gefunden
028F          EEBA07  INC CNTR          ;Zähler+1
0292          A000    LDY #0            ;Programmtext-Index zurück
0294 BISMIN   E8      INX
0295          BDFA06  LDA TEXT-1,X      ;Lies bis zum nächsten Wort
0298          10FA    BPL BISMIN
029A          BDFB06  LDA TEXT,X
029D          D0E4    BNE COMP          ;ungleich 0, noch Befehlswort
029F          F087    BEQ BR            ;Ende des Vergleichs
02A1
02A1          ;Befehlswort ausführen
02A1 MATCH    68      PLA               ;Returnadressen vom Stack
02A2          68      PLA
02A3          68      PLA
02A4          68      PLA
02A5          AEBA07  LDX CNTR          ;Index
02A8          BD0807  LDA LAENGE,X      ;für die Längentabelle
02AB          18      CLC
02AC          656A    ADC $6A           ;zum Pointer addieren
02AE          85C6    STA $C6           ;in $C6, $C7
02B0          A900    LDA #0
02B2          656B    ADC $6B
02B4          85C7    STA $C7
02B6          8A      TXA               ;in Accu

02B7          0A      ASL A             ;*2
02B8          AA      TAX               ;ist Index in Befehlsadressen
02B9          BD0B07  LDA ADR,X         ;Adresse Low
02BC          856A    STA $6A
02BE          BD0C07  LDA ADR+1,X       ;Adresse High
02C1          856B    STA $6B
02C3          6C6A00  JMP ($6A)         ;Befehl ausführen
02C6
-------------------------------------------------------------
ATN (Arcustangens-) Funktion

02C6 ATN      A5AE    LDA $AE
02C8          48      PHA
02C9          1003    BPL AT1
02CB          20B8CC  JSR $CCB8
02CE AT1      A5A9    LDA $A9
02D0          48      PHA
02D1          C981    CMP #$81
02D3          9007    BCC AT2
02D5          A9FB    LDA #$FB
02D7          A0C6    LDY #$C6
02D9          204ECB  JSR $CB4E
02DC AT2      A95C    LDA #<ATNCON      ;Konstanten für ATN
02DE          A007    LDY #>ATNCON
02E0          2044CD  JSR $CD44
02E3          68      PLA
02E4          C981    CMP #$81
02E6          9007    BCC AT3
02E8          A94E    LDA #$4E
02EA          A0CE    LDY #$CE
02EC          208FC5  JSR $C58F
02EF AT3      68      PLA
02F0          1003    BPL ATRET
02F2          4CB8CC  JMP $CCB8
02F5 ATRET    60      RTS
02F6
-------------------------------------------------------------
OPEN-Funktion

02F6 OPEN     20C500  JSR OLDCH         ;altes Zeichen
02F9          D003    BNE ONR           ;kein ':' oder Zeilenende
02FB OERR     4CD0BC  JMP SNERR         ;Syntax-Error ausgeben
02FE ONR      20DCC4  JSR NUMOR         ;Filenummer einlesen
0301          C92C    CMP #','          ;Komma muß folgen
0303          D0F6    BNE OERR
0305          8A      TXA               ;Wert in Accu
0306          48      PHA               ;retten
0307          205506  JSR FILEDA        ;Filenummer schon vorhanden ?
030A          F011    BEQ NEWENT        ;nein, neues File
030C          68      PLA               ;Accu holen
030D          8EBB07  STX NOWX          ;Filenummer zwischenspeichern
0310          BDAC07  LDA KANAL,X       ;altes File schließen
```

```
0313           AB      TAY
0314           BDA207  LDA GERAET,X
0317           207DF1  JSR FCLOSE
031A           4C2F03  JMP GETGER      ;Gerätenummer lesen
031D NEWENT    AEBC07  LDX MAXFIL      ;Anzahl offener Files
0320           E00A    CPX #10         ;schon 10 ?
0322           9003    BCC ENTR        ;nein, lege es an
0324           4CBDC6  JMP OVERR       ;sonst Overflow-Error
0327 ENTR      E8      INX             ;erhöhe Anzahl
0328           8EBB07  STX NOWX        ;in NOWX für später
032B           68      PLA
032C           9D9807  STA FILENR,X    ;Filenummer eintragen
032F GETGER    20D9C4  JSR NUM255      ;Gerätenummer einlesen
0332           C92C    CMP #','        ;Komma muß folgen
0334           D0C5    BNE OERR        ;sonst Fehler
0336           8A      TXA             ;Wert in Accu
0337           AEBB07  LDX NOWX
033A           9DA207  STA GERAET,X    ;Gerätenummer eintragen
033D           20D9C4  JSR NUM255      ;Kanalnummer lesen
0340           8A      TXA
0341           AEBB07  LDX NOWX
0344           9DAC07  STA KANAL,X     ;Kanalnummer eintragen
0347           ECBC07  CPX MAXFIL      ;X < MAXFIL ?
034A           9005    BCC REPL        ;ist Replace
034C           F003    BEQ REPL        ;ebenso
034E           EEBC07  INC MAXFIL      ;Anzahl Files erhöhen
0351 REPL      20C500  JSR OLDCH       ;letztes Zeichen ',' ?
0354           D00A    BNE FNAME       ;ja, Filename folgt
0356 OPO       BDAC07  LDA KANAL,X     ;sonst ohne Namen eröffnen
0359           AB      TAY
035A           BDA207  LDA GERAET,X
035D           38      SEC             ;kein Filename
035E           B02F    BCS OOIT        ;eröffne es
0360 FNAME     C92C    CMP #','        ;Trennzeichen = Komma ?
0362           D097    BNE OERR        ;sonst Fehler
0364           20BF00  JSR NEWCH
0367           207FBB  JSR VALEXP      ;Filenamen aufbereiten
036A           240A    BIT $A          ;ist neg. bei String
036C           1024    BPL OTMERR      ;sonst Type-Mismatch-Error
036E           20E4C3  JSR STRLAD      ;Länge und Adresse holen
0371           AA      TAX             ;Länge
0372           D006    BNE SETNAM      ;Länge > 0
0374           AEBB07  LDX NOWX
0377           4C5603  JMP OPO         ;kein Filename
037A SETNAM    86A7    STX LAE         ;Länge setzen
037C           A56A    LDA $6A         ;Adresse Low
037E           A46B    LDY $6B         ;Adresse High
0380           85AB    STA TXT         ;setzen
0382           84A9    STY TXT+1
0384           AEBB07  LDX NOWX
0387           BDAC07  LDA KANAL,X
038A           AB      TAY
038B           BDA207  LDA GERAET,X
038E           18      CLC             ;mit Filenamen
038F OOIT      4C2CF3  JMP FOPEN       ;File eröffnen
0392 OTMERR    4C7ABB  JMP TMERR       ;Type-Mismatch-Error
0395
-------------------------------------------------------------
CLOSE-Funktion

0395 CLOSE     20C500  JSR OLDCH       ;altes Zeichen lesen
0398           D006    BNE FNR         ;wenn nicht Ende
039A           4CD0BC  JMP SNERR       ;Syntax-Error
039D CERR      4C87BF  JMP FCERR       ;Function-Call-Error
03A0 FNR       20DCC4  JSR NUMOR       ;Filenummer einlesen
03A3           8A      TXA             ;in Accu
03A4           205506  JSR FILEDA      ;File offen ?
03A7           F0F4    BEQ CERR        ;nein, FC-Error
03A9           BDAC07  LDA KANAL,X     ;File schließen
03AC           A8      TAY
03AD           BDA207  LDA GERAET,X
03B0           207DF1  JSR FCLOSE
03B3 UP        BDA307  LDA GERAET+1,X  ;Files aufrücken
03B6           9DA207  STA GERAET,X
03B9           BDAD07  LDA KANAL+1,X
03BC           9DAC07  STA KANAL,X
03BF           BD9907  LDA FILENR+1,X
03C2           9D9807  STA FILENR,X
```

```
03C5          E8      INX                 ;bis zum Ende der Liste
03C6          ECBC07  CPX MAXFIL
03C9          90E8    BCC UP              ;Schleife
03CB          CEBC07  DEC MAXFIL          ;ein File weniger
03CE          60      RTS
03CF
-------------------------------------------------------------
CBM-Flag setzen

03CF          ;Flag für LOAD# und SAVE# im CBM-Format
03CF CBM      A901    LDA #1
03D1          8DBF07  STA CBMFLG          ;setzen
03D4          60      RTS
03D5
-------------------------------------------------------------
PRINT#-Funktion

03D5 PRINT    20BF00  JSR NEWCH           ;neues Zeichen
03D8          20D9C4  JSR NUM255          ;Filenummer einlesen
03DB          C92C    CMP #','            ;Komma muß folgen
03DD          F006    BEQ FILOK
03DF PERR     4CD0BC  JMP SNERR           ;Syntax-Error
03E2 PFCERR   4CB7BF  JMP FCERR           ;Function-Call-Error
03E5 FILOK    8A      TXA                 ;Filenummer
03E6          205506  JSR FILEDA          ;File offen ?
03E9          F0F7    BEQ PFCERR          ;File existiert nicht
03EB          BDAC07  LDA KANAL,X         ;File zum Zuhören auffordern
03EE          A8      TAY
03EF          BDA207  LDA GERAET,X
03F2          20F1FF  JSR DOLIST
03F5          AD0A01  LDA UOUT            ;User-Output-Vektor retten
03F8          AC0B01  LDY UOUT+1
03FB          8DBD07  STA UT1
03FE          8CBE07  STY UT2
0401          A936    LDA #<BOUT          ;U-OUT auf BOUT umsetzen
0403          A004    LDY #>BOUT
0405          8D0A01  STA UOUT
0408          8C0B01  STY UOUT+1
040B          AD13A4  LDA OUTFLG          ;OUTFLG retten
040E          8DB807  STA FT

0411          A955    LDA #'U'            ;auf 'U' setzen
0413          8D13A4  STA OUTFLG
0416          20BF00  JSR NEWCH           ;neues Zeichen
0419          20A9B8  JSR BPRINT          ;Basic-PRINT ausführen
041C          ADB807  LDA FT              ;OUTFLG und U-OUT-Vektor restaurieren
041F          8D13A4  STA OUTFLG
0422          ADBD07  LDA UT1
0425          ACBE07  LDY UT2
0428          8D0A01  STA UOUT
042B          8C0B01  STY UOUT+1
042E          2097E5  JSR UNLIS           ;Bus freigeben
0431          68      PLA                 ;Return-Adresse vom Stack
0432          68      PLA
0433          4CC8B5  JMP BBLOOP          ;zur Basic-Befehlsschleife
0436
0436 BOUT     68      PLA                 ;Zeichen vom Stack
0437          4CEFF0  JMP STAIEC          ;in File schreiben
043A
-------------------------------------------------------------
INPUT#-Funktion

043A INPUT    20E4C0  JSR DIRIND          ;kein direkter Befehl erlaubt
043D          20BF00  JSR NEWCH           ;neues Zeichen
0440          20D9C4  JSR NUM255          ;Filenummer einlesen
0443          C92C    CMP #','            ;Komma muß folgen
0445          F006    BEQ FILOKI
0447          4CD0BC  JMP SNERR           ;Syntax-Error
044A IERR     4C87BF  JMP FCERR           ;Function-Call-Error
044D FILOKI   8A      TXA                 ;Filenummer
044E          205506  JSR FILEDA          ;File offen ?
0451          F0F7    BEQ IERR            ;nein
0453          BDAC07  LDA KANAL,X         ;File zum Senden auffordern
0456          A8      TAY
0457          BDA207  LDA GERAET,X
045A          20EFFE  JSR DOTALK
045D          AD0801  LDA UIN             ;User Input Vektor retten
0460          AC0901  LDY UIN+1
0463          8DBD07  STA UT1
```

```
0466          8CBE07 STY UT2
0469          A9A3   LDA #<BIN        ;U-IN auf BIN umsetzen
046B          A004   LDY #>BIN
046D          8D0B01 STA UIN
0470          8C0901 STY UIN+1
0473          AD12A4 LDA INFLG        ;INFLG retten
0476          8DB807 STA FT
0479          A955   LDA #'U'         ;INFLG auf 'U' setzen
047B          8D12A4 STA INFLG
047E          38     SEC
047F          6610   ROR $10          ;Ausgabe unterdrücken
0481          20BF00 JSR NEWCH        ;nächstes Zeichen
0484          20BEB9 JSR BINPU        ;Basic-Input-Funktion
0487          4610   LSR $10          ;Ausgabe wieder zulassen
0489          ADB807 LDA FT           ;INFLG und UIN-Vektor restaurieren
048C          8D12A4 STA INFLG
048F          ADBD07 LDA UT1
0492          ACBE07 LDY UT2
0495          8D0B01 STA UIN
0498          8C0901 STY UIN+1
049B          2087E5 JSR UNTALK       ;Bus freigeben
049E          68     PLA              ;Returnadresse vom Stack
049F          68     PLA
04A0          4CC8B5 JMP BBLOOP       ;zur Basic Befehlsschleife
04A3
04A3 BIN      2006F1 JSR LDAIEC       ;ein Zeichen vom File lesen
04A6          48     PHA              ;retten
04A7          AD17A4 LDA STATUS
04AA          C942   CMP #$42         ;hinter Fileende ?
04AC          D005   BNE BINOK        ;nein
04AE          A206   LDX #6           ;Out-Of-Data-Error
04B0          4C59B2 JMP GENERR       ;erzeugen
04B3 BINOK    68     PLA              ;Zeichen holen
04B4          60     RTS
04B5
-------------------------------------------------------------
GET#-Funktion

04B5 GET      20E4C0 JSR DIRIND       ;kein direkter Befehl erlaubt
04B8          20BF00 JSR NEWCH
04BB          20D9C4 JSR NUM255       ;Filenummer lesen
04BE          C92C   CMP #','         ;Komma muß folgen
04C0          F009   BEQ GETVAR
04C2 GETERR   4CD0BC JMP SNERR        ;Syntax-Error
04C5 GFCERR   4C87BF JMP FCERR        ;Function-Call-Error
04C8 GTMERR   4C7ABB JMP TMERR        ;Type-Mismatch-Error
04CB GETVAR   8A     TXA              ;Filenummer
04CC          205506 JSR FILEDA       ;File offen ?
04CF          F0F4   BEQ GFCERR       ;nein
04D1          8EBB07 STX NOWX         ;Index in Filetabelle
04D4 RNCH     AEBB07 LDX NOWX         ;Index holen
04D7          BDAC07 LDA KANAL,X      ;File zum Senden auffordern
04DA          A8     TAY
04DB          BDA207 LDA GERAET,X
04DE          20EFFE JSR DOTALK
04E1          2006F1 JSR LDAIEC       ;Zeichen vom File einlesen
04E4          A000   LDY #0           ;Index in Eingabepuffer
04E6          AE17A4 LDX STATUS       ;Status lesen
04E9          F004   BEQ NB           ;noch im File
04EB          E040   CPX #$40         ;Fileende-Marke ?
04ED          D004   BNE LEERST       ;schon hinter Fileende, Leerstring!
04EF NB       991600 STA $16,Y        ;Zeichen in Eingabepuffer
04F2          C8     INY              ;Index + 1
04F3 LEERST   A900   LDA #0           ;Begrenzer
04F5          991600 STA $16,Y        ;in Puffer
04F8          8506   STA $6           ;0 als Begrenzer
04FA          2087E5 JSR UNTALK       ;Bus freigeben
04FD          20BF00 JSR NEWCH        ;nächstes Zeichen
0500          20E4BD JSR FINDV        ;Variable suchen bzw. einrichten
0503          240A   BIT $A           ;neg. wenn Stringvariable
0505          10C1   BPL GTMERR       ;sonst Type-Mismatch
0507          8591   STA $91          ;Variablenadresse in $91,$92
0509          8492   STY $92
050B          A916   LDA #$16         ;Zeiger auf eingegebenes Zeichen ($0016)
050D          A000   LDY #0
050F          20CBC1 JSR STRPTR       ;Stringpointer setzen
0512          2049B8 JSR STOSTR       ;String in Variable speichern
0515          20C500 JSR OLDCH        ;folgt Komma auf Variable ?
```

```
0518          F006    BEQ GETEND      ;nein, Ende
051A          C92C    CMP #','        ;Komma ?
051C          F0B6    BEQ RNCH        ;ja, nächstes Zeichen
051E          D0A2    BNE GETERR      ;anderes Zeichen, SN-Error
0520 GETEND   68      PLA             ;Returnadresse vom Stack
0521          68      PLA
0522          4CC8B5  JMP BBLOOP      ;zur Basic-Befehlsschleife
0525
-----------------------------------------------------------
SAVE@-Funktion

0525 SAVE     206406  JSR LS          ;Gerätenummer und Filenamen lesen
0528          A001    LDY #1          ;Programm vorhanden ?
052A          B1C6    LDA ($C6),Y
052C          D003    BNE SOP         ;ja, wenn nicht 0
052E          4C7FB2  JMP WSTART      ;zum Basic-Warmstart
0531 SOP      ADBB07  LDA NOWX        ;Gerätenummer
0534          18      CLC             ;mit Filenamen
0535          202CF3  JSR FOPEN
0538          ADBB07  LDA NOWX        ;Gerätenummer
053B          A001    LDY #1          ;Kanalnummer
053D          20F1FF  JSR DOLIST      ;zum Zuhören auffordern
0540          A573    LDA $73         ;Programmstart in File
0542          20EFF0  JSR STAIEC
0545          A574    LDA $74
0547          20EFF0  JSR STAIEC
054A          A204    LDX #4          ;4 Bytes Header
054C GC       A000    LDY #0
054E          B1C6    LDA ($C6),Y     ;ein Byte lesen
0550          E000    CPX #0          ;schon Programmtext ?
0552          F003    BEQ SX0         ;ja
0554          CA      DEX
0555          101E    BPL STORE       ;auf File speichern
0557 SX0      C900    CMP #0          ;Ende der Zeile ?
0559          D004    BNE SNEOL       ;nein
055B          A202    LDX #2          ;2 Bytes (Rest des Headers)
055D          D01F    BNE NULL        ;Zeilenende bearbeiten
055F SNEOL    A8      TAY             ;Programmbyte
0560          1013    BPL STORE       ;pos. wenn kein Token
0562          ACBF07  LDY CBMFLG      ;CBM-Modus ?
0565          F00E    BEQ STORE       ;nein
0567          A000    LDY #0          ;Zeiger in Tokenliste
0569 NAT      D91107  CMP AIMTOK,Y    ;Vergleich mit AIM-Tokens
056C          F003    BEQ CALTOK      ;Token berechnen
056E          C8      INY             ;Index + 1
056F          D0F8    BNE NAT         ;nächster Token
0571 CALTOK   98      TYA             ;Index in Tabelle + $80
0572          18      CLC
0573          6980    ADC #$80        ;ergibt CBM-Token
0575 STORE    20EFF0  JSR STAIEC      ;in File speichern
0578 GCN      204E06  JSR INCPTR      ;Zeichenpointer erhöhen
057B          4C4C05  JMP GC          ;nächstes Zeichen
057E NULL     20EFF0  JSR STAIEC      ;Zeilenende behandeln
0581          204E06  JSR INCPTR      ;nächstes Zeichen

0584          B1C6    LDA ($C6),Y
0586          20EFF0  JSR STAIEC      ;auf File speichern
0589          204E06  JSR INCPTR      ;nächstes Zeichen
058C          B1C6    LDA ($C6),Y     ;holen
058E          48      PHA             ;retten
058F          20EFF0  JSR STAIEC      ;auf File speichern
0592          68      PLA             ;holen
0593          D0E3    BNE GCN         ;wenn 0 dann Programmende
0595          ADBB07  LDA NOWX        ;Programmende, File schließen
0598          A001    LDY #1          ;Kanal 1
059A          207DF1  JSR FCLOSE
059D          A900    LDA #0
059F          8DBF07  STA CBMFLG      ;CBM-Modus zurücksetzen
05A2          4C7FB2  JMP WSTART      ;zum Basic-Warmstart
05A5
-----------------------------------------------------------
LOAD@-Funktion

05A5 LOAD     206406  JSR LS          ;Gerätenummer und Filenamen lesen
05A8          ADBB07  LDA NOWX        ;Gerätenummer
05AB          A000    LDY #0          ;Kanalnummer
05AD          18      CLC             ;mit Filenamen
05AE          202CF3  JSR FOPEN       ;File eröffnen
05B1          ADBB07  LDA NOWX        ;zum Senden auffordern
```

```
05B4          A000    LDY #0
05B6          20EFFE  JSR DOTALK
05B9          2006F1  JSR LDAIEC        ;Programmstartadresse überlesen
05BC          2006F1  JSR LDAIEC
05BF          A204    LDX #4            ;4 Bytes Header
05C1 LLOOP    A000    LDY #0
05C3          2006F1  JSR LDAIEC        ;Byte vom File lesen
05C6          E000    CPX #0            ;noch Header ?
05C8          F003    BEQ XO            ;nein
05CA          CA      DEX
05CB          101A    BPL NOREM         ;abspeichern
05CD XO       C900    CMP #0            ;Zeilenende ?
05CF          D004    BNE NEOL          ;nein
05D1          A204    LDX #4            ;4 Bytes Header
05D3          D012    BNE NOREM
05D5 NEOL     A8      TAY               ;Zeichen testen
05D6          100F    BPL NOREM         ;pos. wenn kein Token
05D8          ACBF07  LDY CBMFLG        ;CBM-Modus ?
05DB          F00A    BEQ NOREM         ;nein
05DD          297F    AND #%01111111    ;Bit 7 entfernen
05DF          A8      TAY               ;als Index in Y-Reg.
05E0          B91107  LDA AIMTOK,Y      ;AIM-Token aus der Liste holen
05E3          D002    BNE NOREM         ;>0, dann gültig
05E5          A98E    LDA #$8E          ;sonst auf REM setzen
05E7 NOREM    A000    LDY #0
05E9          91C6    STA ($C6),Y       ;Zeichen speichern
05EB          204E06  JSR INCPTR        ;Zeiger erhöhen
05EE          AD17A4  LDA STATUS        ;teste Fileende
05F1          C940    CMP #$40          ;EOF ?
05F3          F029    BEQ LEND          ;Fileende
05F5          C942    CMP #$42          ;Fehler ?
05F7          F00F    BEQ NULLE         ;ja
05F9          A5C7    LDA $C7           ;teste, ob noch Platz da ist
05FB          C580    CMP $80           ;Top of Memory High
05FD          90C2    BCC LLOOP         ;ja
05FF          A57F    LDA $7F
0601          38      SEC
0602          E5C6    SBC $C6           ;noch 3 Zeichen Platz ?
0604          C904    CMP #4
0606          B0B9    BCS LLOOP         ;>= 4 Zeichen Platz
0608 NULLE    08      PHP               ;Flags retten
0609          A900    LDA #0            ;sonst Programmende setzen
060B          A202    LDX #2
060D NULLS    91C6    STA ($C6),Y       ;3 Nullen als Ende-Marke
060F          204E06  JSR INCPTR        ;Pointer erhöhen
0612          CA      DEX
0613          10F8    BPL NULLS         ;Schleife
0615          28      PLP               ;Flags holen
0616          B006    BCS LEND          ;bei Fehler
0618          202406  JSR SETVA         ;File schließen, Pointer setzen
061B          4C57B2  JMP OMERR         ;OM-Error ausgeben
061E LEND     202406  JSR SETVA         ;File schließen, Pointer setzen
0621          4C7FB2  JMP WSTART        ;zum Basic-Warmstart
0624
-------------------------------------------------------------
LOAD-File schließen, Pointer setzen

0624 SETVA    ADBB07  LDA NOWX          ;Gerätenummer
0627          A000    LDY #0            ;Kanalnummer
0629          8CBC07  STY MAXFIL        ;Anzahl Files auf 0
062C          8CBF07  STY CBMFLG        ;CBM-Modus zurücksetzen

062F          207DF1  JSR FCLOSE        ;File schließen
0632          A5C6    LDA $C6
0634          8575    STA $75           ;Variablenstart setzen
0636          8577    STA $77           ;Arraystart setzen
0638          8579    STA $79           ;Top of Used Memory setzen
063A          A5C7    LDA $C7
063C          8576    STA $76
063E          8578    STA $78
0640          857A    STA $7A
0642          A57F    LDA $7F           ;Top of free space setzen
0644          857B    STA $7B
0646          A580    LDA $80
0648          857C    STA $7C
064A          2029B3  JSR PACKEN        ;Zeilenstartadressen neu setzen
064D          60      RTS
064E
-------------------------------------------------------------
```

```
Programmzeiger erhöhen

064E INCPTR E6C6    INC $C6          ;Pointer $C6,$C7
0650         D002   BNE IR
0652         E6C7   INC $C7
0654 IR      60     RTS
0655
-------------------------------------------------------------
auf offenes File testen

0655 FILEDA AEBC07 LDX MAXFIL        ;Anzahl Files
0658        F008   BEQ FILRET        ;0, dann kein File offen
065A FLOOP  DD9807 CMP FILENR,X      ;vergleiche mit Filenummern
065D        F003   BEQ FILRET        ;File gefunden
065F        CA     DEX
0660        D0F8   BNE FLOOP         ;Schleife
0662 FILRET 8A     TXA               ;Index in Accu
0663        60     RTS
0664
-------------------------------------------------------------
Gerätenummer und Filenamen lesen

0664 LS     20BF00 JSR NEWCH         ;neues Zeichen
0667        20D9C4 JSR NUM255        ;Gerätenummer lesen
066A        C92C   CMP #','          ;Komma muß folgen
066C        F009   BEQ FDNAM         ;Filenamen holen
066E LSERR  4CD0BC JMP SNERR         ;Syntax-Error
0671 LTMERR 4C7ABB JMP TMERR         ;Type-Mismatch-Error
0674 LFCERR 4C87BF JMP FCERR         ;Function-Call-Error
0677 FDNAM  8EBB07 STX NOWX          ;Gerätenummer retten
067A        20BF00 JSR NEWCH         ;neues Zeichen
067D        207FBB JSR VALEXP        ;Filenamen lesen
0680        240A   BIT $A            ;neg. wenn String
0682        10ED   BPL LTMERR        ;kein Filename
0684        20E4C3 JSR STRLAD        ;Länge und Adresse setzen
0687        AA     TAX               ;Länge
0688        F0EA   BEQ LFCERR        ;kein Filename
068A        86A7   STX LAE           ;Länge setzen
068C        A56A   LDA $6A           ;Adresse setzen
068E        A46B   LDY $6B
0690        85AB   STA TXT
0692        84A9   STY TXT+1
0694        A573   LDA $73           ;Pointer $C6 auf Programmstart
0696        A474   LDY $74
0698        85C6   STA $C6
069A        84C7   STY $C7
069C        60     RTS
069D
-------------------------------------------------------------
Sicherheitsroutine

069D               ;teste, ob in der Eingabeschleife INFLG=U oder OUTFLG=U
069D               ;wenn ja, teste ob UIN auf BIN oder UOUT auf BOUT steht
069D               ;und restauriere in diesem Fall UIN bzw. UOUT.
069D SAFETY AD12A4 LDA INFLG         ;teste INFLG
06A0        C955   CMP #'U'
06A2        D01F   BNE TOUT          ;nicht User
06A4        AD0B01 LDA UIN           ;teste ob UIN auf BIN steht
06A7        C9A3   CMP #<BIN
06A9        D018   BNE TOUT
06AB        AD0901 LDA UIN+1
06AE        C904   CMP #>BIN
06B0        D011   BNE TOUT
06B2        A90D   LDA #13           ;steht auf BIN
06B4        8D12A4 STA INFLG         ;INFLG zurück
06B7        ADBD07 LDA UT1           ;UIN restaurieren
06BA        8D0B01 STA UIN
06BD        ADBE07 LDA UT2
06C0        8D0901 STA UIN+1
06C3 TOUT   AD13A4 LDA OUTFLG        ;teste OUTFLG
06C6        C955   CMP #'U'
06C8        D01F   BNE DOCRLF        ;nein
06CA        AD0A01 LDA UOUT          ;teste, ob UOUT auf BOUT steht
06CD        C936   CMP #<BOUT
06CF        D018   BNE DOCRLF
06D1        AD0B01 LDA UOUT+1
06D4        C904   CMP #>BOUT
06D6        D011   BNE DOCRLF
```

```
06D8          A90D    LDA #13          ;steht auf BOUT
06DA          8D13A4  STA OUTFLG       ;OUTFLG zurück
06DD          ADBD07  LDA UT1          ;UOUT restaurieren
06E0          8D0A01  STA UOUT
06E3          ADBE07  LDA UT2
06E6          8D0B01  STA UOUT+1
06E9 DOCRLF   4C00B9  JMP BCRLF        ;Basic CRLF
06EC
-------------------------------------------------------------
Tabellen

06EC TOKNUM           =4               ;Anzahl Tokens
06EC TOKENS 97        .BYT $97,$84,$94,$93,$9B
06ED        84
06EE        94
06EF        93
06F0        9B
06F1 BEFADR D503      .WOR PRINT,INPUT,SAVE,LOAD,GET
06F3        3A04
06F5        2505
06F7        A505
06F9        B504
06FB
06FB TEXT   4F5045    .BYT 'OPE',$4E+$80
06FE        CE
06FF        434C      .BYT 'CLOS',$45+$80
0703        C5
0704        4342      .BYT 'CB',$4D+$80
0706        CD
0707        00        .BYT 0
0708 LAENGE 04        .BYT 4,5,3
0709        05
070A        03
070B ADR    F602      .WOR OPEN,CLOSE,CBM
070D        9503
070F        CF03
0711
-------------------------------------------------------------
AIM-Tokens

0711          ;CBM-Token ist Index in Liste der AIM-Tokens
0711 AIMTOK
0711          80      .BYT $80,$81,$82,$83,$84,$84,$85,$86,$87,$88
0712          81
0713          82
0714          83
0715          84
0716          84
0717          85
0718          86
0719          87
071A          88
071B          89      .BYT $89,$8A,$8B,$8C,$8D,$8E,$8F,$90,$92,$93
071C          8A
071D          8B
071E          8C
071F          8D
0720          8E
0721          8F
0722          90
0723          92
0724          93
0725          94      .BYT $94,$00,$95,$96,$97,$97,$98,$99,$00,$00
0726          00
0727          95
0728          96
0729          97
072A          97
072B          98
072C          99
072D          00
072E          00
072F          00      .BYT $00,$00,$00,$9B,$9C,$9D,$9E,$9F,$A0,$A1
0730          00
0731          00
```

```
0732          9B
0733          9C
0734          9D
0735          9E
0736          9F
0737          A0
0738          A1
0739          A2      .BYT $A2,$A3,$A4,$A5,$A6,$A7,$A8,$A9,$AA,$AB
073A          A3
073B          A4
073C          A5
073D          A6
073E          A7
073F          A8
0740          A9
0741          AA
0742          AB
0743          AC      .BYT $AC,$AD,$AE,$AF,$B0,$B1,$B2,$B3,$B4,$B5
0744          AD
0745          AE
0746          AF
0747          B0
0748          B1
0749          B2
074A          B3
074B          B4
074C          B5
074D          B6      .BYT $B6,$B7,$B8,$B9,$BA,$BB,$BC,$BD,$BE,$BF
074E          B7
074F          B8
0750          B9
0751          BA
0752          BB
0753          BC
0754          BD
0755          BE
0756          BF
0757          C0      .BYT $C0,$C1,$C2,$C3,$C4
0758          C1
0759          C2
075A          C3
075B          C4
075C
-------------------------------------------------------------
Konstanten für Arcustangens-Funktion

075C ATNCON   0B      .BYT $0B,$76,$B3,$83,$BD,$D3,$79,$1E,$F4,$A6
075D          76
075E          B3
075F          83
0760          BD
0761          D3
0762          79
0763          1E
0764          F4
0765          A6
0766          F5      .BYT $F5,$7B,$83,$FC,$B0,$10,$7C,$0C,$1F,$67
0767          7B
0768          83
0769          FC
076A          B0
076B          10
076C          7C
076D          0C
076E          1F
076F          67
0770          CA      .BYT $CA,$7C,$DE,$53,$CB,$C1,$7D,$14,$64,$70
0771          7C
0772          DE
0773          53
0774          CB
0775          C1
0776          7D
0777          14
0778          64
0779          70
```

```
077A        4C      .BYT $4C,$7D,$B7,$EA,$51,$7A,$7D,$63,$30,$88
077B        7D
077C        B7
077D        EA
077E        51
077F        7A
0780        7D
0781        63
0782        30

0783        88
0784        7E      .BYT $7E,$7E,$92,$44,$99,$3A,$7E,$4C,$CC,$91
0785        7E
0786        92
0787        44
0788        99
0789        3A
078A        7E
078B        4C
078C        CC
078D        91
078E        C7      .BYT $C7,$7F,$AA,$AA,$AA,$13,$B1,$00,$00,$00
078F        7F
0790        AA
0791        AA
0792        AA
0793        13
0794        81
0795        00
0796        00
0797        00
0798        00      .BYT $00
0799
------------------------------------------------------------
Arbeitsvariablen

0799 FI             *=*+10          ;Filenummern
07A3 GE             *=*+10          ;Gerätenummern
07AD SE             *=*+10          ;Sekundäradressen
07B7 FILENR         =FI-1           ;Index läuft ab 1
07B7 GERAET         =GE-1           ;Index läuft ab 1
07B7 KANAL          =SE-1           ;Index läuft ab 1
07B7
07B7 ATEMP          *=*+1           ;Zwischenspeicher für Accu
07B8 FT             *=*+1           ;Zwischenspeicher für INFLG/OUTFLG
07B9 XTEMP          *=*+1           ;Zwischenspeicher für X-Reg.
07BA CNTR           *=*+1           ;Zähler und Zwischenspeicher
07BB NOWX           *=*+1
07BC MAXFIL         *=*+1           ;Anzahl offener Files
07BD UT1            *=*+1           ;Zwischenspeicher für User-Vektor
07BE UT2            *=*+1           ;dito
07BF CBMFLG         *=*+1           ;Flag für CBM-Modus
07C0
07C0 ENDERW                         ;Ende der Erweiterung = Programmstart
07C0        00      .BYT 0,0,0      ;kein Programm nach dem Initialisieren
07C1        00
07C2        00
07C3
07C3                *=$BB
00BB        4CC602  JMP ATN         ;Einsprung für ATN-Funktion
00BE
00BE                *=$73
0073        ;Programmstart, Var.-Start, Array-Start, Top of Used Memory
0073        ;setzen
0073        C007    .WOR ENDERW,ENDERW,ENDERW,ENDERW
0075        C007
0077        C007
0079        C007
007B
007B                .END
ERRORS=0000
```

Karl-Anton Dichtel, 7800 Freiburg

EPSON FX-80 am AIM mit Remon

Das folgende Programm ermöglicht die Ansteuerung eines FX-80 mit Centronics-Interface ohne weitere Hardware durch die User-VIA des AIM 65. Programmstart über *=9C00 mit Taste F1 oder Änderung des Betriebssystems in ECE5 in JSR IRSO in JSR 9C00. Bei dieser Umschaltung des ON/OFF bzw. Programmstart über PRINT-Taste muß der Befehl in Zelle 9C13 in ein RTS geändert werden. – Der erstmalige Durchlauf nach dem Einschalten oder nach dem Betätigen der Reset-Taste ergibt:

1. Initialisierung der User-VIA
2. Umsetzen des DILINK-Vektors für Zeichenausgabe an den Drucker
3. Ausgabe von Kontrollzeichen für den Drucker
4. Text 'ON' auf dem Display
5. Einschalten des Druckers durch Setzen von Bit 1 zu '1' in Zelle A408.

Ein Programmdurchlauf, wenn die VIA bereits initialisiert ist, ergibt:

1. Ausschalten Druckerausgabe
2. Text 'OFF' (nur auf dem Display)

Weitere Programmstarts ändern nur den ON/OFF-Zustand des Druckers.

Eine Betätigung der DEL-Taste löscht keine Zeichen im Printerbuffer. Dies müßte analog zur Videoausgabe in REMON noch implementiert werden. Falls nur gespeicherte Daten bzw. Texte gedruckt werden, stellt dies keinen Nachteil dar.

Die Hardware wird wie folgt beschaltet, wobei alle Anschlüsse des Centronics-Interfaces mit der VIA verbunden sind, aber nicht alle werden vom Programm bedient. Anschlüsse von Port A, die als Eingänge programmiert sind, erzeugen über die internen pull up-Widerstände logisch 1.

USER-VIA	FX-80	Bedeutung
CB2	1	Strobe
PB0	2	Data 1
PB1	3	Data 2
PB2	4	Data 3
PB3	5	Data 4
PB4	6	Data 5
PB5	7	Data 6
PB6	8	Data 7
PB7	9	Data 8
CB1	10	Acknlg
PA7	11	Busy
PA6	12	Paper empty
PA5	14	Auto Feed XT
GND	16,17	Masse
GND	19-30	Masse
PA4	31	Init
PA3	32	Error
GND	33	Masse
PA2	36	Slct In

```
0000 ON          =$E6FA      ;TEXTAUSGABE ON
0000 OFF         =$E6F1      ;TEXTAUSGABE OFF
0000 VIA         =$A000      ;GRUNDADRESSE VIA
0000 OUTREB      =VIA        ;AUSGABE DATENBYTE
0000 OUTREA      =VIA+1      ;E/A KONTROLLEI-TUNGEN
0000 DDRB        =VIA+2      ;8 AUSGAENGE
```

```
0000 DDRA          =VIA+3          ;BIT5:OUT,REST:IN
0000 PCR           =VIA+$C
0000 DIRA          =$20            ;NUR BIT 5 AUSGANG
0000 DIRB          =$FF            ;AUSGANG DATEN
0000 STROBE        =$A0            ;AUTOMATISCHER STROBE
0000 FLAG          =$A408          ;BIT 1
0000 VIOUT         =$F255
*************************************************************

0000          ;PROGRAMMMSTART
0000                *=$9C00        ;STARTADRESSE INIT BZW. ON/OFF
9C00 PRINT  AD03A0 LDA DDRA
9C03        C920   CMP #DIRA       ;IST VIA INITIALISIERT ?
9C05        F009   BEQ PRIN1
9C07        20429C JSR INIT        ;INITIALISIERUNG
9C0A        20699C JSR DILINK      ;DILINK-VEKTOR SETZEN
9C0D        205B9C JSR KOUT        ;DRUCKERMODE
9C10 PRIN1  20169C JSR ALTER       ;DRUCKER EIN
9C13        4C82E1 JMP $E182       ;ZURUECK
*************************************************************

9C16          ;DRUCKER EIN/AUS
9C16 ALTER  AD08A4 LDA FLAG        ;FLAG=0:PRINTER OFF
9C19        2902   AND #02         ;BIT 1
9C1B        F00A   BEQ ALTE1
9C1D        CE08A4 DEC FLAG
9C20        CE08A4 DEC FLAG
9C23        20F1E6 JSR OFF         ;TEXT:OFF(NUR DISPLAY)
9C26        60     RTS
9C27 ALTE1  20FAE6 JSR ON          ;TEXT:ON(NUR DISPLAY)
9C2A        EE08A4 INC FLAG
9C2D        EE08A4 INC FLAG
9C30        60     RTS
*************************************************************

9C31          ;ZEICHENAUSGABE AN DRUCKER
9C31 DROUT  48     PHA
9C32        AD08A4 LDA FLAG
9C35        2902   AND #02
9C37        F005   BEQ NOTPR       ;DRUCKER AUS
9C39        68     PLA
9C3A        48     PHA
9C3B        20529C JSR BOUT        ;BYT AUSGEBEN
9C3E NOTPR  68     PLA
9C3F        4C55F2 JMP VIOUT       ;VIDEO AUSGABE

9C42          ;INITIALISIERUNG
9C42 INIT   A9FF   LDA #DIRB
9C44        8D02A0 STA DDRB
9C47        A920   LDA #DIRA
9C49        8D03A0 STA DDRA
9C4C        A9A0   LDA #STROBE
9C4E        8D0CA0 STA PCR
9C51        60     RTS
*************************************************************

9C52          ;AUSGABE:1 BYT AN DRUCKER
9C52 BOUT   2C01A0 BIT OUTREA
9C55        30FB   BMI BOUT        ;READY?
```

```
9C57          8D00A0 STA OUTREB       ;JA!
9C5A          60     RTS
*************************************************************

9C5B          ;AUSGABE KONTROLLZEICEN AN DRUCKER
9C5B KOUT     A200   LDX #00
9C5D KOU1     BD749C LDA TAB,X        ;TEXTTABELLE
9C60          D001   BNE KOU2         ;ENDZEICHEN:00
9C62          60     RTS
9C63 KOU2     20529C JSR BOUT
9C66          E8     INX
9C67          D0F4   BNE KOU1
*************************************************************

9C69          ;DILINK-VEKTOR SETZEN
9C69 DILINK   A931   LDA #<DROUT      ;BYTE LOW
9C6B          8D06A4 STA $A406
9C6E          A99C   LDA #>DROUT      ;BYTE HIGH
9C70          8D07A4 STA $A407
9C73          60     RTS
9C74 TAB      1B     .BYT $1B,$43,$47 ;300 MM
9C75          43
9C76          47
9C77          1B     .BYT $1B,$4E,$07 ;64 ZEILEN
9C78          4E
9C79          07
9C7A          1B     .BYT $1B,$6C,$0A,$00 ;10 LEERSPALTEN LINKS
9C7B          6C
9C7C          0A
9C7D          00
9C7E                 .END
 ERRORS= 0000
```

Wolfgang Zweygart, 7030 Böblingen

65C02 Assembler - Disassembler

unter FORTH

1. Der Disassembler

Der Disassembler basiert auf dem Vorschlag von Dr. H. Mörtl aus MICRO MAG Nr. 31. Dabei wurden folgende wesentliche Änderungen eingearbeitet:

- Keine Verwendung von Screens; es wird eine Opcode-Tabelle vom Programm aufgebaut.
- Erweiterung auf den Befehlssatz des 65C02.

Die neuen Befehle wurden in die Opcode- und in die Adressierungsartentabelle eingefügt. Der Disassembler 'paßt' in einen beliebigen 4K-Block, hier ist es der Block ab $D000, und er ist EPROM-fähig geschrieben. Natürlich kann er noch immer mit dem FORTH-Disassembler aus Nr. 31 bzw. Nr. 35 zusammenarbeiten. Es gibt folgende Besonderheiten: Bei der Adressierungsart 'immediate' wird der Operand zuerst in hexadezimal, dann dezimal, dann binär und, falls es sich um ein druckbares Zeichen handelt, als ASCII-Zeichen ausgegeben. Bei den neuen Bit-Branch-befehlen wird hinter einem %-Zeichen das angesprochene Bit, dann die fragliche Zeropageadresse und zuletzt das Sprungziel ausgegeben. Der Disassembler enthält außerdem die CASE-Struktur und das korrigierte Wort U< (unsigned compare).

2. Der Assembler

Das Ziel bei der Entwicklung dieses Programmes war es, dem alten FORTH-Assembler die neuen Befehle des C02-Prozessors beizubringen, ohne dabei die Mängel dieser Assemblerart zu beseitigen und ohne den Aufwand, einen völlig eigenständigen neuen Assembler aufzubauen. In sein Vokabular werden die neuen Befehle und Adressierungsarten durch Neudefinierung des alten Wortes eingefügt. Dabei ist die Variable MODE von großem Nutzen, da sie die vorher gewählte Adressierungsart enthält. Diese Variable wird abgefragt und das Ergebnis mit den möglichen Adressierungsarten verglichen. Dadurch wird das problemlose Einfügen der neuen Adressierungsarten in die alten Befehlsworte erst möglich.

Die Bit-Branch-Befehle sind sowohl direkt als auch durch die größeren Strukturen wie IF '' THEN, BEGIN .. UNTIL usw. erreichbar. Dazu wurde ein neuer 'Conditional Specifier' eingeführt. BS ergibt dann die Bit-Set-Bedingung und BS NOT entsprechend die Bit-Reset-Bedingung. Zusätzlich müssen vorher das angesprochene Bit und die Zeropage-Adresse auf den Stack geschoben werden. Ebenso wird durch den Assembler der unbedingte absolute Sprung in den Kontrollstrukturen durch den neuen Branch-Always-Befehl (BRA) ersetzt. Daher bringt der neue Assembler bereits dann, wenn vorhandene alte FORTH-Assembler-Programme neu compiliert werden, einen Zeit- und Speicherplatzvorteil.

```
FORGET TASK
HEX D000 DP ! ( Compiliert ab $D000 )
E0A0 DUP UFIRST ! ULIMIT !
( routine fuer start mit <N> )
ASSEMBLER C28F JSR, C2CE JSR,
0 # LDA, 305 STA,
0 # LDA, 306 STA, ( TASK bei $ 309 )
6 # LDY, UP )Y LDA, TAX, B535 JMP,
FORTH

DECIMAL
( CASE - Struktur  )
: CASE ?COMP CSP @ !CSP 6 ; IMMEDIATE
: OF 6 ?PAIRS COMPILE OVER COMPILE =
COMPILE 0BRANCH HERE 0 , COMPILE DROP 7 ; IMMEDIATE
: ENDOF 7 ?PAIRS COMPILE BRANCH HERE 0 ,
SWAP 2 [COMPILE] ENDIF 6 ; IMMEDIATE
: ENDCASE 6 ?PAIRS COMPILE DROP
BEGIN SP@ CSP @ = 0= WHILE 2 [COMPILE] ENDIF REPEAT
CSP ! ; IMMEDIATE
: U< ( n1 n2 --- f )
( unsigned compare for address )
OVER OVER 0< SWAP 0< XOR
IF SP@ 3 + C@ SP@ 3 + C@ < ROT ROT 2DROP
ELSE - 0< THEN ;

: TABELLE ( defining word for opcodetabell )
<BUILDS 30 * ALLOT ( opcodetabelle als neuer datentyp )
DOES> SWAP 30 * + 30 - ;
HEX
2B TABELLE OPTAB ( tabelle aufbauen )
: % ( wort zum einlesen der tabelle )
20 WORD HERE DUP 1+ SWAP C@ DUP 1+ ALLOT
ROT SWAP CMOVE ; ( liest nachfolgenden string bis zum )
( naechsten leerzeichen )
```

```
( Opcods als mnemnonic, adressierungsart, bytanzahl )
 1 OPTAB % BRK11ORA32---11---11TSB62ORA62
 2 OPTAB % ASL62RMBH2PHP11ORA52ASL01---11
 3 OPTAB % TSB93ORA93ASL93BBRF3BPL42ORA22
 4 OPTAB % ORAE2---11TRB62ORA82ASL82RMBH2
 5 OPTAB % CLC11ORAC3INC01---11TRB93ORAB3
 6 OPTAB % ASLB3BBRF3JSR93AND32---11---11
 7 OPTAB % BIT62AND62ROL62RMBH2PLP11AND52
 8 OPTAB % ROL01---11BIT93AND93ROL93BBRF3
 9 OPTAB % BMI42AND22ANDE2---11BIT82AND82
 A OPTAB % ROL82RMBH2SEC11ANDC3DEC01---11
 B OPTAB % BITB3ANDB3ROLB3BBRF3RTI11EOR32
 C OPTAB % ---11---11---11EOR62LSR62RMBH2
 D OPTAB % PHA11EOR52LSR01---11JMP93EOR93
 E OPTAB % LSR93BBRF3BVC42EOR22EORE2---11
 F OPTAB % ---11EOR82LSR82RMBH2CLI11EORC3
10 OPTAB % PHY11---11---11EORB3LSRB3BBRF3
11 OPTAB % RTS11ADC32---11---11STZ62ADC62
12 OPTAB % ROR62RMBH2PLA11ADC52ROR01---11
13 OPTAB % JMPA3ADC93ROR93BBRF3BVS42ADC22
14 OPTAB % ADCE2---11STZ82ADC82ROR82RMBH2
15 OPTAB % SEI11ADCC3PLY11---11JMPB3ADCB3
16 OPTAB % RORB3BBRF3BRA42STA32---11---11
17 OPTAB % STY62STA62STX62SMBH2DEY11BIT52
18 OPTAB % TXA11---11STY93STA93STX93BBSF3
19 OPTAB % BCC42STA22STAE2---11STY82STA82

1A OPTAB % STX72SMBH2TYA11STAC3TXS11---11
1B OPTAB % STZ93STAB3STZB3BBSF3LDY52LDA32
1C OPTAB % LDX52---11LDY62LDA62LDX62SMBH2
1D OPTAB % TAY11LDA52TAX11---11LDY93LDA93
1E OPTAB % LDX93BBSF3BCS42LDA22LDAE2---11
1F OPTAB % LDY82LDA82LDX72SMBH2CLV11LDAC3
20 OPTAB % TSX11---11LDYB3LDAB3LDXC3BBSF3
21 OPTAB % CPY52CMP32---11---11CPY62CMP62
22 OPTAB % DEC62SMBH2INY11CMP52DEX11---11
23 OPTAB % CPY93CMP93DEC93BBSF3BNE42CMP22
24 OPTAB % CMPE2---11---11CMP82DEC82SMBH2
25 OPTAB % CLD11CMPC3PHX11---11---11CMPB3
26 OPTAB % DECB3BBSF3CPX52SBC32---11---11
27 OPTAB % CPX62SBC62INC62SMBH2INX11SBC52
28 OPTAB % NOP11---11CPX93SBC93INC93BBSF3
29 OPTAB % BEQ42SBC22SBCE2---11---11SBC82
2A OPTAB % INC82SMBH2SED11SBCC3PLX11---11
2B OPTAB % ---11SBCB3INCB3BBSF3---11---11
60 USER CODE.S SMUDGE ( variable )
( hier mit USER damit objectcode eprom-faehig ist )
: .BEFEHL DUP 3 TYPE ;
: CODE-ADR ( sucht CODE in OPTAB )
DUP CODE.S ! 6 ( 6 codes pro zeile )
/MOD SWAP 5 * ( 5 zeichen pro code )
SWAP 1+ OPTAB SWAP + ;
: ?BYTE ( sucht anzahl der bytes )
DUP 4 + C@ F AND 1 MAX 3 MIN ;
: ?MODE ( sucht addressierungsart )
3 + C@ 12 DIGIT DROP ; ( hier neue zahlenbasis )
: .2BYT S->D <# # # # # #> TYPE SPACE ;
: $2BYT ." $" .2BYT ;
: .1BYT S->D <# # # #> TYPE SPACE ;
```

```
: $1BYT ." $" .1BYT ;
: RELA ( branch-rumpf )
DUP 7F > IF FF XOR 1+ NEGATE THEN
3 PICK + ;
: BRBS ( branch fuer BBR und BBS )
RELA 3 + ( correctur der relativen addresse )
$2BYT ;
: BRAN ( branch fuer normale branchbefehle )
RELA 2+ $2BYT ;
ASSEMBLER
CODE NIB ( schiebt um ein nibble nach rechts )
TOP LDA,
.A LSR, .A LSR, .A LSR, .A LSR,
TOP STA,
NEXT JMP, END-CODE
: BITS ( nummer des bits und zero-page adresse ausgeben )
." %" CODE.S @ NIB DUP
8 < IF . ELSE 8 - . ENDIF $1BYT ;
CODE MSB ( n1 n2 --- >b <b )
( trennung von zp-adresse und relativem sprung )
0 # LDA,
TOP 1+ STA,
SEC 1+ LDA,
SEC STA,
0 # LDA,
SEC 1+ STA,
NEXT JMP,
END-CODE
12 BASE ! ( wieder neue zahlenbasis )

: .MODE ( bestimmung der adressierungsart )
OVER ?MODE 4 SPACES
CASE ( es gibt 17 moeglichkeiten )
0 OF ." .A" DROP ENDOF ( accu )
1 OF DROP ENDOF ( implied )
2 OF ." (" $1BYT ." ),Y" ENDOF ( <zp>,y )
3 OF ." (" $1BYT ." ,X)" ENDOF ( <zp,x> )
4 OF BRAN ENDOF ( branch )
5 OF ." #" DUP 2DUP $1BYT ( immidiate )
."  #" DECIMAL 0 3 D.R 2 BASE ! ( hex, dec, bin, string )
."  #%" 0 8 D.R HEX DUP 20 > IF
."  #'" EMIT ELSE DROP ENDIF ENDOF
6 OF $1BYT ENDOF ( zero page )
7 OF $1BYT ." ,Y" ENDOF ( zp,y )
8 OF $1BYT ." ,X" ENDOF ( zp,x )
9 OF $2BYT ENDOF          ( extended )
A OF ." (" $2BYT ." )" ENDOF ( indirect )
B OF $2BYT ." ,X" ENDOF ( ext,x )
C OF $2BYT ." ,Y" ENDOF ( ext,y )
E OF ." (" $1BYT ." )" ENDOF ( <zp> indirect )
F OF DUP MSB BITS BRBS ENDOF ( bbr,bbs )
G OF ." (" $2BYT ." ),X" ENDOF ( <ind>,x )
H OF BITS ENDOF ( rmb,smb )
ENDCASE ;
HEX
: PRINT ( adr --- adr+ )
CR DUP .2BYT SPACE ( gibt einen befehl aus )
DUP C@ CODE-ADR ?BYTE 3 PICK C@ SPACE .1BYT
CASE
```

```
1 OF 6 SPACES OVER C@ ENDOF ( 1-byte befehle )
2 OF OVER 1+ C@ DUP .1BYT 3 SPACES ENDOF ( 2-byte ..)
3 OF OVER 1+ DUP DUP @ SWAP C@ .1BYT SWAP 1+ C@ .1BYT ENDOF
ENDCASE
5 SPACES SWAP ( abstand zum memnonic-feld )
.BEFEHL SPACE SWAP .MODE ( decodieren und ausgeben )
?BYTE SWAP DROP + ;
: DISASS ( from to --- )
SWAP BEGIN PRINT ( disassembliert von - bis )
DUP 3 PICK U<
0= UNTIL DROP DROP ;
: DISA ( from --- next )
16 0 DO PRINT LOOP ; ( schreibt einen Bildschrim voll)

HEX
300 DP ! ( wieder ab $0300 compilieren )
5000 DUP UFIRST ! ULIMIT !
: TASK ;
305 C@ D007 C!
306 C@ D00C C!
FINIS ( ende des disassemblers )

FORGET TASK HEX : TASK ; ( beginn ab $0300 )
ASSEMBLER DEFINITIONS
HEX
: NJ NEXT JMP, ;
( 65C02 kommandos )
: MODE-CL 2 MODE C! ;
: BRA, 80 C, ; ( 'branch always' direkt erreichbar )
: PHY, 5A C, ; ( neue push-pull befehle )
: PLY, 7A C, ;
: PHX, DA C, ;

: PLX, FA C, ;
: INC, MODE C@ 0 = IF 1A C, ELSE INC, ENDIF MODE-CL ;
( adressierungsart 'accumulator' hinzufuegen )
: DEC, MODE C@ 0 = IF 3A C, ELSE DEC, ENDIF MODE-CL ;
: JMP, MODE C@ 3 = IF 7C C, , ELSE JMP, ENDIF MODE-CL ;
( mit x indizierter indirekter sprung hinzufuegen )
: ZPTEST ( kontrolliert ob zeropage-adressierung moeglich )
SWAP DUP FF > ROT SWAP ;
: ZPTE DUP FF > ;
: STZ, MODE C@ DUP ( neuer befehl store-zero )
2 = IF ZPTEST IF 9C C, DROP , ELSE 64 C, DROP C,
THEN ELSE DUP
3 = IF ZPTEST IF 9E C, DROP , ELSE 74 C, DROP C,
THEN ELSE DUP
ENDIF ENDIF MODE-CL ;
: BIT, MODE C@ DUP ( mit neuen adressierungsarten )
1 = IF 89 C, DROP C, ELSE
3 = IF ZPTE IF 3C C, , ELSE 34 C, C, THEN ELSE
BIT, ENDIF ENDIF MODE-CL ;
: MODE@ MODE C@ F = ;
( adresierungsart 'indirect' hinzufuegen )
: ORA, MODE@ IF 12 C, C, ELSE ORA, ENDIF MODE-CL ;
: AND, MODE@ IF 32 C, C, ELSE AND, ENDIF MODE-CL ;
: EOR, MODE@ IF 52 C, C, ELSE EOR, ENDIF MODE-CL ;
: ADC, MODE@ IF 72 C, C, ELSE ADC, ENDIF MODE-CL ;
: STA, MODE@ IF 92 C, C, ELSE STA, ENDIF MODE-CL ;
```

```
: LDA, MODE@ IF B2 C, C, ELSE LDA, ENDIF MODE-CL ;
: CMP, MODE@ IF D2 C, C, ELSE CMP, ENDIF MODE-CL ;
: SBC, MODE@ IF F2 C, C, ELSE SBC, ENDIF MODE-CL ;
: TSB, ZPIE IF C C, , ELSE 4 C, C, ENDIF ;
( zwei neue befehle fuer bits )
: TRB, ZPTE IF 1C C, , ELSE 14 C, C, ENDIF ;
: RMB, 10 * 7 + C, C, ; ( reset memory bit )
( zp-add bit --- )
: SMB, 10 * 87 + C, C, ; ( set memory bit )
: BBR, 10 * F + C, C, C, ; ( direkt erreichbar )
: BBS, 10 * 8F + C, C, C, ; ( reljmp zp-add bit --- )

( bei control words JMP, durch BRA, ersetzen )
( und relativen sprung berechnen )
: ELSE, ( addr1 2 --- addr2 2  ; at assembly )
2 ?PAIRS HERE 1+ BRA, 0 C, SWAP
HERE OVER 1+ - SWAP C! 2 ;
: ENDIF, ( addr 2 ---  ; at assembly )
2 ?PAIRS HERE OVER 1+ - SWAP C! ;
: THEN, ENDIF, ;
: AGAIN, ( addr 1 ---  ; at assembly )
1 ?PAIRS HERE 3 + - FF - BRA, C, ;
: REPEAT, ( addr1 1 addr3 3 ---  ; at assembly )
3 ?PAIRS >R AGAIN, R> HERE OVER 1+ - SWAP C! ;

CODE NIB2 ( subr. fuer NOT )
TOP LDA, ( hoeherwertiges nibble loeschen )
.A ASL, .A ASL, .A ASL, .A ASL,
.A LSR, .A LSR, .A LSR, .A LSR,
TOP STA,
NJ END-CODE
: BR.CHK DUP NIB2 F = ; ( unterscheidung zwischen normal- )
( und bit-branch befehlen )
: NOT ( cc1 --- cc2 )  ( negation der bedingung )
BR.CHK IF 80 XOR ELSE NOT ENDIF ;

: BR.INS ( branch einfuegen )
BR.CHK IF C, C, ELSE C, ENDIF ;

( bei control words bit-branch befehle ermoeglichen )
: IF, ( --- addr 2 ; at assembly )
BR.INS HERE 0 C, 2 ;
: WHILE, ( --- addr 3 ; at assembly )
BR.INS HERE 0 C, 3 ;
: UNTIL, ( addr 1 cc ---  ; at assembly )
BR.CHK IF >R >R 1 ?PAIRS R> R> C, C, ELSE
>R 1 ?PAIRS R> C, ENDIF HERE 1+ - C, ;
: BS ( add bit --- add cc )
10 * F + ; ( as a conditional specifier  )


FORTH DEFINITIONS ( umschalten auf forth )
ASSEMBLER ( und assembler aufrufen )
CODE TEST ( beispiele fuer die neuen moeglichkeiten )
PHX,
PHY,
,X 2000 JMP,
TOP LDA,
.A INC,
```

```
20 ) ORA,
20 ) STA,
88 # LDA,
20 TSB,
0= IF,
TOP STZ,
ENDIF,
# 11 BIT,
CS IF,
20 6 RMB,
ELSE,
20 1 SMB,
ENDIF,
```

```
20 0 BS IF,
55 # LDA,
ELSE,
54 # LDA,
ENDIF,
E97A JSR,
BEGIN,
TOP STZ,
22 4 BS
UNTIL,
50 # LDA,
E97A JSR,
PLY,
PLX,
NJ
END-CODE
HEX
FINIS
```

Hier das funktionslose Beispielprogramm.
So sieht ein Ausdruck des Disassemblers aus

```
ON  OK
080F 0845 DISASS
080F   DA            PHX
0810   5A            PHY
0811   7C 00 20      JMP     ($2000 ),X
0814   B5 00         LDA     $00 ,X
0816   1A            INC     .A
0817   12 20         ORA     ($20 )
0819   92 20         STA     ($20 )
081B   A9 88         LDA     #$88  #136 #%10001000
081D   04 20         TSB     $20
081F   D0 02         BNE     $0823
0821   74 00         STZ     $00 ,X
0823   89 11         BIT     #$11  # 17 #%   10001
0825   90 04         BCC     $082B
0827   67 20         RMB     %6 $20
0829   80 02         BRA     $082D
082B   97 20         SMB     %1 $20
082D   0F 20 04      BBR     %0 $20 $0834
0830   A9 55         LDA     #$55  # 85 #% 1010101 #'U
0832   80 02         BRA     $0836
0834   A9 54         LDA     #$54  # 84 #% 1010100 #'T
0836   20 7A E9      JSR     $E97A
0839   74 00         STZ     $00 ,X
083B   4F 22 FB      BBR     %4 $22 $0839
083E   A9 50         LDA     #$50  # 80 #% 1010000 #'P
0840   20 7A E9      JSR     $E97A
0843   7A            PLY
0844   FA            PLX     Ok
 OK
```

Roland Löhr

FORTH: Entwicklungs und Zielsystem

Rockwell bietet u.a. die Einchipper R65F11 (40 Pin) und R65F12 (64 Pin Quip) an, die im maskenprogrammierten ROM von 3 KB bereits einen Standard-iBefehlssatz der Sprache FORTH enthalten. Hinzu kommen 2 Timer, 192 Byte RAM on Chip und 2 bzw. 5 Ports von 8 Bit Breite. Die Pins PA6 und PA7 können dabei als serielle Schnittstelle voll duplex für verschiedene Datenformate und Baudraten betrieben werden. Zusammen mit wenigen zusätzlichen Bausteinen läßt sich damit ein stand alone Computer aufbauen, der in FORTH programmiert ist und der auch schon einen Floppy Disk-Handler enthält.

Solche FORTH-Computer sind nicht nur für den bequemen Aufbau von Steuerungen interessant, sondern auch als Lehr- und Entwicklungssysteme. In diesem Fall benötigt man ein Entwicklungs-ROM (Typenbezeichnungen: siehe am Schluß), das Programme zu erproben und zu kompilieren gestattet. Interessant in dieser Kombination sind die Möglichkeiten, schon bei der Entwicklung mit Disketten zu arbeiten, einen Target-Compiler benutzen und den erzeugten Code direkt in ein EEROM oder EPROM übertragen zu können. Die Steuerung des Entwicklungssystems erfolgt über den seriellen Kanal des Einchippers von einem Terminal in Vollduplex-Betrieb (hier mit 7 Daten- und 2 Stopbits, ohne parity bei 1200 Baud). Nach der Entfernung des Entwicklungs-ROMs kann man das System dann stand alone arbeiten lassen.

Eine Entwicklungsplatine

In der Rockwell Applikationsschrift mit Bestell-Nr. 2162 ist ein vollständiges Platinen-Layout nebst Schalt- und Bestückungsplan für den Aufbau eines FORTH-Systems enthalten, und zwar für den Typ R65F11 und auch für R65F12, wenn man die abgebildete Adapterplatine aufsetzt. Die Hauptplatine hat einen Steckplatz für das 28-polige Entwicklungs-ROM, einen weiteren 28-poligen und zwei 24-polige Steckplätze für Speicher (EPROMs, RAMs oder EEROMs). Hinzu kommt der serielle Duplexkanal und der Platz für einen FD-Controller WD 2793 (Western Digital), neben Takterzeugung, Dekodierung, Reset usw.. Der FD-Controller kann bis zu 4 Laufwerke mit insgesamt 2,4 MB ansprechen (360 Blöcke quad density pro Seite, Blöcke zu 1 KB). Diese unbestückte Platine kann zu etwa 100 DM (ohne Gewähr) bei den Rockwell-Distributoren bezogen werden. Sie hat keine Bestellnummer, so daß man sich wohl auf die vorerwähnte Applikationsschrift beziehen muß.

Der Autor konnte eine fertig bestückte Platine dieser Art betreiben, die Rockwell vorübergehend leihweise zur Verfügung stellte. Der Anschluß an das Terminal erfolgte in Minutenschnelle. Dabei trat vorübergehend die Überraschung auf, daß am Terminal CBM 710 die Leiterbahnen 2 und 3 für TX und RX Data am 25-poligen Stecker nicht gekreuzt sein dürfen. - Im seriellen Betrieb echot der Computer der Entwicklungsplatine alle Eingaben auf das Terminal..

RSC-FORTH: Eigenschaften

Zum Betrieb und zur Programmierung eines FORTH-Computers dieser Art sollte man unbedingt das RSC-FORTH User's Manual heranziehen: ca. 350 Seiten, Bestell-Nr. 2148, Preis 15 $). Zwar läuft es auf Strecken parallel mit dem AIM 65 FORTH User's Manual, es enthält jedoch die entscheidenden Hinweise für den Betrieb dieses FORTH, die VLISTs, neue Befehle, Betrieb der Massenspeicherung und vor allem die Target-Compilierung (kopfloser/headerless Code). Die nachfolgenden Angaben sind vor allem dieser Quelle entnommen.

Zunächst nebenstehend die VLIST (Befehlsliste) des 3 K Kernel-ROMs auf dem Chip. Die dort genannten Adressen sind Parameterfeldadressen.

Wie beim AIM 65, so handelt es sich auch hier um ein FIG-FORTH. Wir finden im Kernel alle notwendigen arithmetischen, logischen und Vergleichsbefehle, solche für die Ein- und Ausgabe, auch zur Formatierung. Neben einigen wichtigen neuen Befehlen, auf die später hingewiesen wird, sind auch solche zum Holen und Abspeichern von Bytes oder Worten vom/zum Memory enthalten. Alle Befehlsworte sind nur runtime executives (ausführende Routinen), die normalerweise im

Wörterbuch (Dictionary) als verkettete Information gehalten würden. Dieses Dictionary wird stattdessen getrennt im Entwicklungs-ROM gehalten.

Die Trennung hat folgende Vorteile: Ein Anwendersystem mit fertig compiliertem Programm braucht keine Suche im Dictionary und keinen Assembler mehr. Es braucht platzsparend nur noch ausführbaren Code in der Form vorwiegend von Wortadressen für den inneren Interpreter und in der Form ggfs. von Maschinenbefehlen. Daher werden die nicht mehr benötigten Dienstleistungen extern im entfernbaren Entwicklungs-ROM untergebracht. Und diese strenge Trennung kann der Programmierer auch durchhalten, wenn er eine Target-Compilierung veranlaßt. In diesem Falle wird der erzeugte EPROM-fähige Code von den 'Headern' freigehalten, und die Verwaltungsinformation landet in einem anderen RAM-Bereich nach Wahl des Benutzers.

Das RSC-FORTH erspart damit Kopfschmerzen anderer Art in der Target-Compilierung. Hier könnte der Ehrgeiz aufkommen, nur die 'wirklich benötigten' FORTH-Worte einzubinden. Das erfordert zeitaufwendige mehrfache Durchgänge. Hier kann man hingegen sagen: Ein arbeitsfähiger (hier und da vielleicht etwas zu großer) Kernel ist unveränderbar im ROM on chip. Alles was hinzukommt, kann als dichter Code in das EPROM des Anwenders übernommen werden, ohne daß man Mehrfachverdichtungen und Einbindungen laufen lassen muß. - Der Ausführungszeit schadet der ggfs. zu große Kernel nicht, denn in jedem Falle werden ja vom FORTH-System nur Adreßworte ausgeführt, egal wie sie über den Speicher streuen.

Besonderheiten der Speicherverwaltung

Die genannten Einchipper haben ein RAM on chip, und zwar im Adressenbereich von hex 40-FF. Hier liegt - im Gegensatz zur 6502 - von FF herunterkommend der Hardwarestack. FORTH

!	F858	2DUP	F7EF	D+-	FC42	MOD	FCD4
#	FE7D	3	F8BB	D.	FECE	NEGATE	F7A5
#>	FE5C	4	F8BF	D.R	FEB0	OR	F6CE
#S	FEA0	;S	F717	DABS	FC56	OVER	F7C5
(+LOOP)	F4D0	<	F954	DECIMAL	F9BC	PAD	F8F0
(.")	FA31	<#	FE52	DIGIT	F50D	PICK	F98F
(DO)	F4F8	=	F938	DISK	FD08	QUERY	FAAC
(FIND)	F535	>	F96C	DNEGATE	F7B5	R	F73F
(LOOP)	F4AF	>R	F734	DPL	F8E2	R>	F73F
<NUMBER)	FB06	?	FEEC	DREAD	F056	RP!	F6FD
*	FCB6	?TERMINAL	F60A	DROP	F7CF	RP@	F80C
*/	FCE8	@	F83B	DUP	F7E5	R0	F8CD
*/MOD	FCDC	ABS	FC4E	DWRITE	FDBE	ROT	F974
+	F778	AND	F6BE	EEC!	FEF4	S->D	FC2C
+!	F818	BANKC!	FF42	EMIT	F5D4	S0	F8CA
+-	FC36	BANKC@	FF4A	ENCLOSE	F591	SEEK	FE11
-	F96C	BANKEEC!	FF52	ERASE	FAE4	SELECT	FD43
-DUP	F9A5	BANKEXECUTE	FF5A	EXECUTE	F471	SIGN	FE6C
-TRAILING	FA0B	BASE	F8D9	EXPECT	FA45	SP!	F6F3
.	FEE4	BL	F8C3	FILL	FABE	SP@	F6EA
.R	FED8	BLANKS	FAEC	HEX	F9B1	SPACE	F99D
/	FCCA	BOUNDS	F803	HLD	F8E5	SPACES	FE3A
/MOD	FCBE	BRANCH	F480	HOLD	FAF4	SWAP	F7D3
0	F8AF	C!	F868	IN	F8DF	TIB	F8C6
0<	F76B	C/L	F8E8	INIT	FDF1	TOGGLE	F830
0=	F7A5	C@	F84B	KEY	F5F6	TYPE	F93F
0BRANCH	F497	CLD/WRM	F8DC	LEAVE	F722	U*	F646
1	F8B3	CLIT	F458	LIT	F40E	U/	F67B
1+	F8F8	CMOVE	F626	M*	FC7E	U<	F940
1-	F913	COLD	FB48	M/	FC94	UC/L	F8D0
2	F8B7	COUNT	F9E3	M/MOD	FCF2	UPAD	F8D3
2+	F820	CR	F613	MAX	FC6E	UR/W	F8D6
2-	F920	D+	F787	MIN	FC5E	XOR	F6DC
2DROP	F7D1						

VLIST des RSC-FORTH mit Parameterfeldadressen

erfordert daneben einen Datenstack, der zu C2 initialisiert wird. Damit ist für viele Anwendungen eine Verschachtelungstiefe bis zu etwa 30 gegeben. Am unteren Ende, oberhalb der Adresse 40 liegen verschiedene Pointer des Systems. Die I/O-Ports, Timer und Kontrollregister liegen im Adressenbereich 00-1F.

Der Bereich 0100-013F ist für die Decodierung eines FDC-Controllers vorgesehen. Ab 0200 kann auf der genannten Entwicklungsplatine RAM decodiert werden. Ab 0300 findet man User-Variable, ab 0380 den Terminal Input Buffer und ab 0400 RAM oder EPROM nach den Bedürfnissen des Benutzers. In Systemen mit minimaler Ausstattung muß mindestens der Bereich 0300-03FF als RAM gehalten werden.

Beim Reset des Systems führen die Routinen im Kernel an jeder Speichergrenze von 1 K eine Prüfung auf ein bestimmtes Bitmuster durch. Treffen sie es an, so ist es das Zeichen für ein erkanntes Autostart-ROM. Sie entnehmen dann aus den folgenden zwei Bytes die Parameterfeldadresse des ersten Befehls und gelangen somit in die Ausführung des Anwender Forth-Systems. Wir haben dafür das Wort AUTOSTART, das dieses Muster hinterlegt. Diese Eigenschaft Autostart kann bereits während der Programmentwicklung vorteilhaft benutzt werden, solange das Programm noch im RAM ist und nicht in einen Festwertspeicher kopiert wurde (trotzdem: vorher sichern!). Legt man mit Autostart das fragliche Bitmuster ins RAM und drückt die Reset-Taste, so erfolgt bereits ein Test unter Bedingungen wie beim stand alone.

Die Hilfen des Development-ROM (8 K)

Für die Target-Compilierung wird, wie erwähnt, der ausführende Code von der Verwaltungsinformation des Wörterbuches freigehalten. Um trotzdem in der Entwicklungsphase vom Wörterbuch her die Bezüge zum erzeugten Code finden zu können, wird im Wörterbuchteil gegenüber dem normalen FORTH ein zusätzlicher Parameterfeld-Pointer geführt. Bei der Compilierung wird dessen Inhalt statt des benutzten Wortes in den Zielcode eingebunden. Es ist aber auch möglich, ein Wort nur für die Testphase zu entwickeln, dessen Code nicht in den Zielcode eingehen soll. Das Wort HWORD entfernt dafür den Code der letzten Definition aus dem Zielcode (unter Berichtigung der Verwaltungsinformation) und stellt ihn stattdessen in das Dictionary ein. Es gibt weitere neue Kontrollworte für das Dictionary, z.B. ALLOT/. Sie enden typisch mit dem Schrägstrich.

Der Kernel enthält bereits Befehlsworte und beschreibende Variable für Diskettenbetrieb, so z.B. DISK, DREAD, DWRITE, SELECT, SEEK u.a.m. Das Entwicklungs-ROM enthält daneben den Befehl FORMAT, der beide Seiten einer Disk formatiert. Auch Befehle zur Bearbeitung von Screens sind enthalten.

Für den Memory-DUMP auf einen externen Prommer gibt es Befehle, die das vom AIM u.a. her bekannte 'Semikolon-Format' bei der Ausgabe von Blöcken erzeugen. Auf dem Board kann aber auch mit EEC! ein elektrisch löschbares PROM programmiert werden oder, wenn man die Spannungen anlegt, auch ein EPROM. — Wenn man Port B mit den höheren Adreßbusleitungen belegt, um zusätzliche Speicherbänke zu dekodieren, dann kann man fast 4 MB Speicher erreichen. Zur Benutzung steht dann das WORT BANKEEC! zur Speicherung und das BANKEXECUTE für die Ausführung zur Verfügung.

Andere Leistungen des Entwicklungs-ROMs betreffen mnemonische Konstanten für die Interfaceadressen, das CASE-Statement und natürlich einen FORTH-Assembler, der den erweiterten Befehlssatz der CPU abdeckt. Das erwähnte Handbuch zum RSC-FORTH geht ausführlich auf alle alten und neuen Merkmale ein, erklärt auch die Interfaceprogrammierung und die Behandlung von Interrupts.

Zusammenfassung

Dieser Bericht umfaßt für die Leser, die sich schon etwas mit FORTH auskennen, nur die wichtigsten Merkmale der Einchipper mit FORTH und der Entwicklungsplatine. Rückschauend läßt sich feststellen, daß beide Komponenten weitsichtig angelegt sind und es erlauben, sowohl ein Entwicklungssystem als auch Zielsysteme bequem und preiswert aufzubauen. Das zur Hardware. Bekanntlich kostet das Schreiben von Software viel mehr als die Hardware. Hier bietet FORTH

als höhere strukturierte Programmiersprache erhebliche Vorteile gegenüber der reinen Assemblerprogrammierung, sobald man sich in die Anfangsgründe eingearbeitet hat. Insofern ist zu hoffen, daß die hier gegebenen zeitgemäßen Möglichkeiten mit FORTH zunehmend genutzt werden.
Hinweise

Hinweise

Als notwendige Literatur wurden im Text bereits erwähnt: a) RSC-FORTH User's Manual (Best.-Nr. 2148), b) Die Application Note ' A Low Cost Development Module for the R65F11 FORTH Microcomputer' (2162). Empfehlenswert sind weiterhin c) RSC FORTH R65FRX Development and R65FKX Kernel ROMs (2177). Diese Schrift enthält Vorschläge für System-Konfigurationen und die dazu benötigten Kernel- und Entwicklungs-ROMs, denn FORTH-Systeme können auch mit dem ROM-freien R6511Q aufgebaut werden, wenn man den Kernel als weiteren Baustein auf die Platine bringt. Insgesamt gibt es fünf verschiedene Systemoptionen.

```
3FDC END-CODE   3FCB 0<      3FC0 0 =     3FB5 VS
3FAA CS         3F9B NOT     3F73 ELSE,   3F63 THEN,
3F37 ENDIF,     3F1E IF,     3EFC REPEAT, 3EE6 AGAIN,
3ECD WHILE,     3EAC UNTIL,  3E99 BEGIN,  3E79 BITCLR
3E4B BITSET     3E29 RMB,    3DF1 SMB,    3DE1 BIT,
3DD1 JMP,       3DC1 JSR,    3DB1 STY,    3DA1 LDY,
3D91 LDX,       3DB1 CPY,    3D71 CPX,    3D61 STX,
3D51 ROR,       3D41 ROL,    3D31 LSR,    3D21 INC,
3D11 DEC,       3D01 ASL,    3CF1 STA,    3CE1 SBC,
3CD1 ORA,       3CC1 LDA,    3DB1 EOR,    3CA1 CMP,
3C91 AND,       3C81 ADC,    3C73 TXS,    3C65 TYA,
3C57 TXA,       3C49 TSX,    3C3B TAY,    3C2D TAX,
3C1F SEI,       3C11 SED,    3C03 SEC,    3BF5 RTS,
3BE7 RTI,       3BD9 PLP,    3BCB PLA,    3BBD PHP,
3BAF PHA,       3BA1 NOP,    3B93 INY,    3B85 INX,
3B77 DEY,       3B69 DEX,    3B5B CLV,    3B4D CLI,
3B3F CLD,       3B31 CLC,    3B23 BRK,    3A54 RP)
3A44 SEC        3A34 TOP     3A27 )       3A1C )Y
3A10 X)         3A04 ,Y      39F8 ,X      39EC MEM
39DF #          39D4 ,A      39A1 SETUP   3993 BINARY
3984 PUTOA      3976 PUSHOA  3967 POPTWO  3958 POP
394C PUT        3940 PUSH    3933 NEXT    3926 XSAVE
3918 UP         390D W       3903 IP      38F8 N
```

VLIST des RSC-FORTH Assemblers

Bücher

Norton, H. n. (Hrsg.): SSG Sensor Selection Guide. Elsevier Sequoia S.A., Lausanne 1984, 162 S., SFrs. 30, ISBN 0-444-75024-X. Die Sensortechnik wird immer weiter vorangetrieben. Das Buch enthält etwa 150 Einträge unter gleicher Gliederung für Konfiguration, Montage, el. Diagramm, stat. u. dyn. Verhalten, Kalibrierung, Anwendungshinweise, Zusatzeinrichtungen. Behandelte Sachgebiete sind Beschleunigung, Vibration, mechanischer Ort, Durchfluß, Feuchtigkeit, Verdrehung, Füllstand, Druck, Dichte, Geschwindigkeit, Temperatur, Viskosität, Vakuum, Kraft und Masse, Höhe.Die Blätter sind mit Schemazeichnungen und Diagrammen unterlegt.

Computer Publications Survey 1984. Der Verlag North Holland, PO. Box 1991, NL-1000 NZ Amsterdam legte seinen neuen Buchkatalog (100 S.) vor, in dem sicher mehr als 200 wissenschaftliche Titel zu den zahlreichen mit dem Computer verbundenen Sachgebieten mit der Beschreibung des Inhalts vorgestellt werden. Dieser Verlag veröffentlicht auch einen SOFTWARE CATALOG.

Rockwell International: R6500 Programming Manual. Der vom AIM 65 her bekannte Programmierleitfaden ist nunmehr auf den erweiterten Befehlssatz der neuen CPU's angepaßt worden. Das Buch hat die Bestell-Nr. 202 bei den Distributoren.

Hans-Georg Lange, 1000 Berlin 30

CBM 6x0 und 7x0 als Terminal

Die Commodore-Rechner der o.a. Serien verfügen über eine hardwaremäßig realisierte RS-232-Schnittstelle, wobei der Interfacebaustein 6551 benutzt wird. Damit entfallen die vielen Probleme der softwaremäßigen Realisierung, die denjenigen wohl leidlich bekannt sind, die eine asynchrone Schnittstelle auf diesem Wege realisieren wollten. Das Betriebssystem 'FIFOt' zudem alle ankommenden Bytes in einen 256 Byte großen Pufferspeicher. Die Verwendung der CBM als 'DCU' bietet sich somit geradezu an.

Zur Hardware:

1. Da der CBM sich selbst als Host betrachtet, der über die RS-232 Daten an Peripheriegeräte ausgeben will, müssen im Anschlußstecker die Leitungen RxD und TxD (2 und 3) für diese Anwendung vertauscht werden.

2. Die vorgeschriebenen Signalpegel (HIGH größer/gleich 3 V, LOW kleiner/gleich 3 V) müssen eingehalten werden. Eine per TTL simulierte Schnittstelle funktioniert nicht, es müssen entsprechende Treiber, z.B. MC 1488, zwischengeschaltet werden. Die benötigten Spannungen können dem RS-232-Stecker direkt entnommen werden.

Das wiedergegebene Programm erklärt sich (hoffentlich) selbst. Wie man mit wenig Aufwand der ganzen Geschichte einen Hauch von 'Intelligenz' verleiht, sollen die beiden realisierten Sonderfunktionen zeigen. Hier ist eine Reihe von Erweiterungen denkbar, z.B. die Einbindung des Screen-Editors oder der Floppy-Betrieb. - Nebenbei: Der beim Programmtest benutzte 'Host' war ein AIM 65.

```
LINE # LOC   CODE      LINE

0004 0000             ;
0005 0000             ;
0006 0000             ; DIESES PROGRAMM DIENT ZUM BETRIEB DES CBM 610/20 710/20
0007 0000             ; ALS TERMINAL AN EINEM HOST-RECHNER UEBER RS232.
0008 0000             ; DATEN WERDEN VOM KEYBOARD GELESEN
0009 0000             ; UND UEBERTRAGEN. ANKOMMENDE BYTES WERDEN AUF
0010 0000             ; DEN BILDSCHIRM GESCHRIEBEN. DIE
0011 0000             ; BETRIEBSART IST 'FULL DUPLEX'
0012 0000             ; 'DEL' ZEICHEN (ASCII 127) UND 'LF' (ASCII 10)
0013 0000             ; WERDEN AUSGEBLENDET
0014 0000             ;
0015 0000             ; DA DIE INPUT-ROUTINE $FFCF NICHT BENUTZT WERDEN
0016 0000             ; KANN, WIRD MIT 'GET' GEARBEITET UND EIN UNDERLINE-
0017 0000             ; KURSOR SIMULIERT.
0018 0000             ;
0019 0000             ; SONDERFUNKTIONEN:
0020 0000             ;'\': (POUND-SIGN BEI CBM 7X0)
0021 0000             ;     ALLE ANKOMMENDEN ZEICHEN WERDEN AB $1000
0022 0000             ;     IM SEGMENT ISEG ABGESPEICHERT BIS ZUM AUFTRETEN
0023 0000             ;     DES NAECHSTEN PROMPTERZEICHENS 'PROMPT' VOM HOST
0024 0000             ;'_': AUSGABE DES SPEICHERINHALTS, AUCH WIEDERHOLT,
0025 0000             ;     AN DEN BILDSCHIRM
0026 0000             ;
0027 0000             ;-----------------------------------------
0028 0000             ;
0029 0000             ; CBM ROM-ROUTINEN:
0030 0000             ;
```

```
0031  0000                BSOUT=$FFD2             ;AUSGABE EINES ZEICHENS
0032  0000                BASIN=$FFE4             ;BASIN=GETIN: EINGBE EINES ZEICHENS
0033  0000                CHKIN=$FFC6             ;TALK
0034  0000                CHKOUT=$FFC9            ;LISTEN
0035  0000                CLRCH=$FFCC             ;RESTORE I/O
0036  0000                OPEN   =$FFC0           ;OEFFNEN LOG.FILE
0037  0000                ;
0038  0000                I6509  =1               ;6509 INDIRECTION BANK
0039  0000                ;
0040  0000                RS232=14                ;LOG FILE RS232
0041  0000                LA     = $9E            ;LOG. GERAETEADRESSE
0042  0000                FA     = $9F            ;PRIMAERADRESSE
0043  0000                SA     = $A0            ;SEKUNDAERADRESSE
0044  0000                FNADR  = $90            ;STARTADRESSE DES FILENAMENS:LOW,HIGH,SEGMENTNR
0045  0000                FNLEN  = $9D            ;LAENGE DES FILENAMENS
0046  0000                ;
0047  0000                ISEG   = 2              ;USER SEGMENT
0048  0000                PROMPT =60              ;'<': HOST-PROMPTER
0049  0000                ;
0050  0000                ;  ZERO PAGE
0051  0000                ;
0052  0000                      * = $2
0053  0002                ;
0054  0002                MFLAG  *=*+1            ;<>0 = ABSPEICHERN
0055  0003                PTR1   *=*+2            ;ADRESSZAEHLER
0056  0005                PTR2   *=*+2            ;ZEIGER AUF LETZTES GESP. BYTE
0057  0007                ;
0058  0007                ;
0059  0007                ;
0060  0007                      *=$700            ;FREIER RAM-BEREICH IM SEGMENT 15
0061  0700                ;
0062  0700                INIT                    ;OEFFNEN DER RS232 SCHNITTSTELLE
0063  0700  A9 0E               LDA #RS232
0064  0702  85 9E               STA LA            ;LOG .FILE 14
0065  0704  A9 02               LDA #2            ;PRIMAERADR. 2=RS232
0066  0706  85 9F               STA FA
0067  0708  A9 03               LDA #3            ;BIDIREKTIONAL
0068  070A  85 A0               STA SA
0069  070C  A9 AC               LDA #<FNAME       ;LOW-BYTE ADR. FILENAME
0070  070E  85 90               STA FNADR
0071  0710  A9 07               LDA #>FNAME
0072  0712  85 91               STA FNADR+1
0073  0714  A9 0F               LDA #15           ;LAEUFT IN UND HAT FILENAME IN SEG. 15
0074  0716  85 92               STA FNADR+2
0075  0718  A9 04               LDA #FNEND-FNAME  ;SOLLTE 4 SEIN
0076  071A  85 9D               STA FNLEN
0077  071C  18                  CLC
0078  071D  20 C0 FF            JSR OPEN          ;OEFFNEN RS232
0079  0720  A9 00               LDA #0            ;SPEICHERFLAG ZURUECK
0080  0722  85 02               STA MFLAG
0081  0724                ;
0082  0724                START
0083  0724  A2 0E               LDX #RS232
0084  0726  20 C6 FF            JSR CHKIN         ;HOLE ZEICHEN VON RS232C
0085  0729  20 E4 FF            JSR BASIN
0086  072C  48                  PHA
0087  072D  20 CC FF            JSR CLRCH         ;RESTORE I/O
0088  0730  68                  PLA
```

```
0091  0731  A6 02        LDX MFLAG     ;SPEICHERFLAG GESETZT?
0092  0733  F0 10        BEQ STRT11    ;NEIN
0093  0735  C9 3C        CMP #PROMPT   ; PROMPTER?
0094  0737  D0 0C        BNE STRT11    ;NEIN
0095  0739  A5 03        LDA PTR1      ;ZEIGER FUER SPEICHERENDE UMSETZEN
0096  073B  85 05        STA PTR2
0097  073D  A5 04        LDA PTR1+1
0098  073F  85 06        STA PTR2+1
0099  0741  A9 00        LDA #0        ;MEMORY-FLAG ZURUECK
0100  0743  85 02        STA MFLAG
0101  0745  29 7F  STRT11 AND #$7F
0102  0747  C9 00        CMP #$00      ;WAR EIN ZEICHEN VON RS ?
0103  0749  F0 29        BEQ START2    ;NEIN
0104  074B  C9 7F        CMP #$7F      ;DELETE ?
0105  074D  F0 25        BEQ START2    ;WAR DELETE
0106  074F  C9 0A        CMP #$0A      ;LINEFEED ?
0107  0751  F0 21        BEQ START2    ;WAR LF
0108  0753  A6 02        LDX MFLAG
0109  0755  F0 03        BEQ STRT01    ;NICHT ABSPEICHERN
0110  0757  20 B7 07     JSR PUTMEM    ;ABSPEICHERN
0111  075A  C9 0D  STRT01 CMP #$0D     ;NEUE ZEILE?
0112  075C  D0 07        BNE START1
0113  075E  48           PHA
0114  075F  A9 14        LDA #20       ;DELETE KURSOR
0115  0761  20 D2 FF     JSR BSOUT     ;IN LETZTER ZEILE
0116  0764  68           PLA
0117  0765  48     START1 PHA           ;ZEICHEN RETTEN
0118  0766  A9 9D        LDA #157      ;KURSOR LINKS
0119  0768  20 D2 FF     JSR BSOUT
0120  076B  68           PLA
0121  076C  20 D2 FF     JSR BSOUT     ; ZEICHEN AUSGEBEN
0122  076F  A9 EF        LDA #239      ;KURSOR AUSGEBEN
0123  0771  20 D2 FF     JSR BSOUT

0125  0774           ;
0126  0774  A9 00  START2 LDA #0
0127  0776  20 E4 FF     JSR BASIN     ;EIN ZEICHEN VON KBD HOLEN
0128  0779  F0 29        BEQ START3    ;WAR KEINE TASTE
0129  077B  C9 5C        CMP #'\       ; BACKSLASH: ALLES IN BANK ISEG AB $1000 SPEICHERN
0130  077D  D0 0C        BNE STRT21    ;WAR NICHT '\
0131  077F  A9 00        LDA #0        ;ADRESSZEIGER ZURUECKSETZEN
0132  0781  85 03        STA PTR1
0133  0783  A9 10        LDA #$10
0134  0785  85 04        STA PTR1+1
0135  0787  85 02        STA MFLAG     ;MEMORY-FLAG SETZEN
0136  0789  A9 0D        LDA #$0D      ;STATT '\ EIN CR AUSGEBEN
0137  078B  C9 5F  STRT21 CMP #'_'      ;VIEW-BEFEHL?
0138  078D  D0 06        BNE STRT22
0139  078F  20 C9 07     JSR VIEW      ;AUSGABE SPEICHERINHALT
0140  0792  4C 45 07     JMP STRT11    ; PROMPTER
0141  0795  48     STRT22 PHA
0142  0796  A2 0E        LDX #RS232
0143  0798  20 C9 FF     JSR CHKOUT    ;LISTEN TO RS232
0144  079B  68           PLA
0145  079C  20 D2 FF     JSR BSOUT     ;ZEICHEN AUSGEBEN
0146  079F  48           PHA
0147  07A0  20 CC FF     JSR CLRCH     ;RESTORE I/O
0148  07A3  68           PLA
0149  07A4  C9 40  START3 CMP #'@       ;ENDE?
```

```
0150  07A6  F0 03           BEQ ENDE
0151  07A8  4C 24 07        JMP START      ;MEIN, LOOP!
0152  07AB  60             ENDE  RTS        ;RUECKKEHR ZU BASIC
0153  07AC                 ;
0154  07AC                 FNAME            ;'FILENAME' DER RS 232
0155  07AC  1A                   .BYT 26+8+0    ;2400 BAUD, 8 DATENBITS,KEINE PARITAET
0156  07AD  10                   .BYT 16        ;ECHO MODE AN
0157  07AE  00                   .BYT 0         ;2 BYTE MUESSEN NOCH KOMMEN
0158  07AF  00                   .BYT 0         ;OHNE BEDEUTUNG
0159  07B0                 FNEND
0160  07B0                 ;
0161  07B0  E6 03          INCPT  INC PTR1       ;ERHOEHEN ADRESSZEIGER
0162  07B2  D0 02                 BNE INCRTS
0163  07B4  E6 04                 INC PTR1+1
0164  07B6  60             INCRTS RTS
0165  07B7                 ;
0166  07B7                 PUTMEM                ;ABSPEICHERN EMPF. ZEICHEN
0167  07B7  A0 00                 LDY #0
0168  07B9  A6 01                 LDX I6509      ;SEGMENT SICHERN
0169  07BB  48                   PHA            ;ZEICHEN SICHERN
0170  07BC  A9 02                 LDA #ISEG      ;ES WIRD ALLES IN BANK ISEG ABGELEGT
0171  07BE  85 01                 STA I6509
0172  07C0  68                   PLA
0173  07C1  91 03                 STA (PTR1),Y   ;ZEICHEN ABLEGEN
0174  07C3  20 B0 07              JSR INCPT      ;ZEIGER ERHOEHEN
0175  07C6  86 01                 STX I6509
0176  07C8  60                   RTS
0177  07C9                 ;
0178  07C9                 VIEW                  ;AUSGABE DES GESPEICHERTEN DIALOGS
0179  07C9  A6 01                 LDX I6509      ;IND. BANK RETTEN
0180  07CB  A9 02                 LDA #ISEG      ;USER SEGMENT
0181  07CD  85 01                 STA I6509
0182  07CF  A9 00                 LDA #0
0183  07D1  85 03                 STA PTR1       ;ZEIGER AUF TEXTANFANG
0184  07D3  A8                   TAY
0185  07D4  A9 10                 LDA #$10

0186  07D6  85 04                 STA PTR1+1
0187  07D8                 ;
0188  07D8  B1 03          VLOOP  LDA (PTR1),Y   ;EINE ZEICHEN HOLEN
0189  07DA  20 D2 FF              JSR BSOUT      ;AUSGEBEN
0190  07DD  20 B0 07              JSR INCPT      ;ZEIGER ERHOEHEN
0191  07E0  A5 06                 LDA PTR2+1     ;TEST AUF ENDE
0192  07E2  C5 04                 CMP PTR1+1
0193  07E4  D0 F2                 BNE VLOOP      ;HIGH-BYTE UNGLEICH?
0194  07E6  A5 03                 LDA PTR1       ;VERGLEICH LOW-BYTE
0195  07E8  C5 05                 CMP PTR2
0196  07EA  90 EC                 BCC VLOOP
0197  07EC.                                      ;
0198  07EC  A9 00                 LDA #0
0199  07EE  85 03                 STA PTR1       ;ZEIGER ZURUECK
0200  07F0  A9 10                 LDA #$10
0201  07F2  85 04                 STA PTR1+1
0202  07F4  86 01                 STX I6509      ;IND.BANK ZURUECK
0203  07F6  A9 3C                 LDA #PROMPT    ; PROMPTERZEICHEN LADEN
0204  07F8  60                   RTS
0205  07F9                 ;
0206  07F9                       .END
```

```
SYMBOL TABLE

SYMBOL  VALUE

BASIN   FFE4  BSOUT   FFD2  CHKIN   FFC6  CHKOUT  FFC9
CLRCH   FFCC  ENDE    07AB  FA      009F  FNADR   0090
FNAME   07AC  FNEND   07B0  FNLEN   009D  I6509   0001
INCPT   07B0  INCRTS  07B6  INIT    0700  ISEG    0002
LA      009E  MFLAG   0002  OPEN    FFC0  PROMPT  003C
PTR1    0003  PTR2    0005  PUTMEM  07B7  RS232   000E
SA      00A0  START   0724  START1  0765  START2  0774
START3  07A4  STRT01  075A  STRT11  0745  STRT21  07BB
STRT22  0795  VIEW    07C9  VLOOP   07D8
END OF ASSEMBLY
```

Roland Löhr

Terminalbetrieb mit CBM 710

Im Artikel 'BASIC 4: Die neuen Befehle' in Heft 32 dieser Zeitschrift wurden die neuen Möglichkeiten des CBM 710 vorgestellt, insbesondere auch auf Seite 9 die Kommandos, die die Bedienung der RS-232-Schnittstelle betreffen. Diese wird von einem Interfacebaustein vom Typ ACIA 6551 zur Entlastung der CPU bedient. Die Schnittstelle muß als Gerätenummer 2 angesprochen werden, gefolgt von einem Byte für die Datenrichtung und einem OPEN-String von 4 Bytes, von denen die ersten beiden Nutzbytes sind, und der Rest ist leeres Stroh ('HI' in Zeile 250 des nachfolgenden Programms).

Die Schnittstelle wurde inzwischen mit 1200 Baud im Duplexverkehr zufriedenstellend erprobt. Höhere Geschwindigkeiten unter diesen Verkehrsverhältnissen konnten hier noch nicht ausprobiert werden. – Bei der Programmentwicklung war es das Ziel, eine Lösung nicht nur für einen speziellen Fall zu implementieren, sondern im interaktiven Dialog mit dem Benutzer eine Anpassung an alle geläufigen Fälle zu erlauben. Wir finden daher in der Reihenfolge die nachstehende Abfrage der Betriebsparameter:

1. Wahl der Baudrate zwischen 50 und 19200 bzw. 16x externe Clock
2. Zahl der Datenbits (5-8)
3. Zahl der Stop-Bits
4. Betriebsart: Ausgabe/Eingabe/Duplex
5. Codewandlung CBM/ASCII?
6. Echo-Modus: Normal oder Echo (dann Senden nicht möglich)
7. Paritätskontrolle und Mark und Space: disabled/odd/even/mark bit ohne parity/ space bit ohne parity

Mit den erfragten Parametern erfolgt in Zeile 250 das OPEN. Zeile 260 holt Zeichen von der Tastatur und sendet sie über RS 232 an den Computer. 270 holt das Echo im Duplexverkehr vom Host zurück und schreibt es auf den Bildschirm. - Die Routine ließe sich beschleunigen, wenn dieser Teil des aktiven Datenverkehrs an den Programmanfang verlegt würde.

```
10 REM RS232 INTERFACE FÜR VOLL DUPLEX DATUM 24.4.84
20 PRINT"INITIALISIERUNG DER RS232-SCHNITTSTELLE":PRINT:PRINTCHR$(15)
22 PRINT"EINGABE + WAHL DER BAUDRATE"
25 FORI=0TO15:READA$:PRINTI,A$:NEXT
30 INPUT"EINGABE";A$:BR=VAL(A$):IFBR>15THEN30:REM BAUDRATE
32 IFBR<>0THENBR=BR+16:REM INTERNE CLOCK EIN
35 PRINT"EINGABE ZAHL DER DATENBITS":FORI=8TO5STEP-1:PRINTI;"BIT":NEXT
40 INPUT"EINGABE";A$:DB=VAL(A$):IFDB>8ORDB<5THEN40:REM DATENBITS
45 IFDB=8THENDB=0:ELSEIFDB=7THENDB=32:ELSEIFDB=6THENDB=64:ELSEDB=96
50 PRINT"EINGABE ZAHL DER STOPBITS":PRINT
60 PRINT0,"1 STOP-BIT":PRINT
```

```
 70 PRINT1,"1 STOP-BIT BEI WORTLÄNGE 8 MIT PARITY"
 75 PRINT,"1.5 STOP-BITS B. WORTLÄNGE 5 OHNE PARITY"
 80 PRINT,"2 STOP-BITS IN ALLEN ANDEREN FÄLLEN"
 85 INPUT"EINGABE";A$:SB=VAL(A$):IFSB<0ORSB>1THEN85
 90 IFSB=1THENSB=128:REM BIT 7 SETZEN
 100 PRINT"EINGABE BETRIEBSART":PRINT
 110 PRINT1,"AUSGABEKANAL":PRINT2,"EINGABEKANAL":PRINT3,"DUPLEXKANAL"
 120 INPUT"EINGABE";A$:BA=VAL(A$):IFBA<0ORBA>3THEN120
 130 PRINT"CODEWANDLUNG CBM/ASCII  J/N?":INPUTA$:IFA$="J"THENBA=BA+128:REM MIT W
ANDLUNG
 140 PRINT"EINGABE ECHO-MODUS":PRINT
 150 PRINT0,"NORMALE ÜBERMITTLUNG":PRINT4,"ECHO-MODUS; SENDEN NICHT MöGLICH"
 160 INPUT"EINGABE";A$:EM=VAL(A$):IFEM=0OREM=4THEN:ELSE160
 170 PRINT:PRINT"PARITÄTSKONTROLLE, MARK & SPACE":PRINT
 180 PRINT0,"PARITY DISABLED"
 190 PRINT32,"ODD PARITY"
 200 PRINT96,"EVEN PARITY"
 210 PRINT160,"MARK BIT OHNE PARITY"
 220 PRINT224,"SPACE BIT OHNE PARITY"
 230 INPUT"EINGABE";A$:PB=VAL(A$):IF(PB=0ORPB=32ORPB=96ORPB=160ORPB=224)THEN:ELS
E230
 250 OPEN2,2,BA,CHR$(SB+DB+BR)+CHR$(EM+PB)+"HI":REM LOG. FILE 2
 255 PRINT""
 257 PRINT#2,CHR$(3)
 260 GETA$:IFA$<>""THENPRINT#2,A$;
 270 GET#2,A$:IFA$=""THEN260:ELSEPRINTA$; :GOTO260
 1000 DATA16X EXTERNE CLOCK,50,75,110,135,150,300,600,1200,1800,2400,3600,4800,7
200,9600,19200
READY
```

Roland Löhr

Textsystem PROTEXT

Persönliche Computer werden verstärkt in der Textbearbeitung des Anwenders eingesetzt. Im allgemeinen muß man für diesen Zweck ein besonderes Textbearbeitungsprogramm hinzukaufen. Bekannte Namen sind da Wordpro für Systeme von Commodore oder Wordstar für CP/M-Systeme. Mit den Rechnergenerationen und mit dem Fortschritt gibt es die unterschiedlichsten Programmversionen. Und man stellt (trotz Vorführung) häufig fest, daß die zusammenarbeitenden Geräte (Computer, Drucker, Floppy) ein fröhliches Eigenleben führen. - In unserem nicht kleinen Lande benutzt man zur Texteingabe auf Schreibmaschinen meistens die geläufige DIN-Tastatur, bei der das 'Z' neben den 'T' sitzt und die deutschen Umlaute rechts vom 'L' und das 'Ü' etwas oberhalb.
Computer werden leider noch zu oft mit amerikanischen ASCII-Tastaturen ausgeliefert. Anbieter von Textsystemen versuchen daher im allgemeinen, die Tastatur auf deutsche Verhältnisse anzupassen und liefern dafür für den Bereich der Festwertspeicher, in dem die Tastatur dekodiert wird, ein besonderes EPROM. Damit ist es aber nicht allein getan: Auch das Character-ROM, das die Bildmatrix auf dem Bildschirm erzeugt, muß mit der Tastatur konform gehen. Es ist dem Anwender also nicht zuzumuten, daß er statt deutscher Umlaute z.B. eckige Klammern auf dem Bildschirm sieht oder statt des 'ß' einen Hochpfeil.

Die kleinen Probleme von dieser Art sind nach dem Stand der Technik in Hard- und Software heute praktisch mühelos zu bewältigen. Oft fehlen natürlich die Austauschtastenkappen für die deutsche Tastaturbelegung, man muß dann mit Auklebern für die Tasten weiterarbeiten, was nicht immer angenehm ist, vor allem, wenn man das Textsystem in die Hand eines Laien legt, z.B. einer Schreibdame für die Texterfassung.

Nachdem man durch Austausch der EPROMs für Tastatur und die Zeichenerzeugung auf dem Bildschirm nun endlich sieht, was man in deutscher Sprache eintippt, beginnen weitere Probleme: Schreibt der Drucker auch alle Buchstaben, so wie sie am Bildschirm stehen? Oder wenn man einen Matrixdrucker hat und eine Schreibmaschine als Schöndrucker: Verhalten sie sich beide gleich? Und dann ist noch die Frage, welche Schnittstellen vom Textsystem bedient werden, Centronics?IEE 488-Bus? Serielle Schnittstelle mit wieviel Baud? Kan man solche Schnittstellen wahlweise bedienen? - Es kann gelegentlich längere Zeit dauern, bis man die Fetzen des Textverarbeitungssystems zusammengeklittert hat, bis man auf allen angeschlossenen Einheiten eine gleichmäßige deutsche Ausgabe nach dem Willen des Anwenders erreicht.

Wenn die Hardware sich nun endlich konformistisch zeigt, tritt die Frage auf,wie freundlich das Textsystem nun eigentlich für den Anwender ist. Beim Wordpro in verschiedenen Versionen hat den Autor eigentlich immer gestört, daß er sich zu seiner Bedienung eigentlich immer ein dickes Handbuch auf die Knie legen mußte, um richtige Bedienungen zu erreichen. Der Wordstar war hier schon hilfreicher, indem er die HELP-Funktion zur Verfügung stellte, mit der man je nach Einarbeitungsgrad feststellen konnte, welche Bearbeitungsmöglichkeiten auf der jeweiligen Befehlsebene zur Verfügung stehen.

Es sind Überlegungen und Beobachtungen aus langer Zeit, die in den vorgehenden Absätzen mitgeteilt wurden. Vor etwa zwei Wochen kam nun auf den Autor die Aufgabe zu, zwei Bücher für ein anderes Unternehmen im Texte zu erfassen und mit weiteren Aufgaben in der Organisation als fertige Exemplare herstellen zu lassen, wobei die möglichst modernen Methoden des computergesteuerten Fotosatzes zum Einsatz kommen sollten. Es war dieses Anlaß, unter Geschäftsfreunden einen möglichst aktuellen Überblick über leistungsfähige Texterfassungssysteme zu gewinnen. Die Wahl fiel nach einiger Beratung auf eine Kombination von Commodore 8296 mit Floppy 1001 (Single Drive) und auf das Textsystem PROTEXT. Nachdem die erste größere Texteingabe mit der notwendigen Korrekturlesung erledigt ist, darf aus den Erfahrungen wie folgt berichtet werden:

1. Erfahrungen

Ein Texterfassungsystem, das man als 'Hausarbeitplatz' in andere Hände legt, muß möglichst unkompliziert sein. Das trifft auf PROTEXT zu. Das Handbuch ist gut gegliedert, hat nur 43 aussagefähige Seiten und bietet Such-, Veränderungs- und Edtiermöglichkeiten, die mühelos zu beherrschen sind, auch nach kurzer Einweisung von einer Dame, die bisher nur die elektromechnischen Kugelköpfe der IBM hat rollen lassen. Eine solche Weitergabe der Texterfassung für Buchdruck sollte aber auf den reinen Text beschränkt werden, wobei der Auftraggeber dann noch aus eigener Sorgfalt die Formatierungsbefehle einfügen sollte, die den Schriftgrad, die Schriftart, die Schrifthöhe (Punkte) und den Durchschuß zwischen den Zeilen betreffen. Die Terminalcomputer der Lichtsatzsysteme haben eine eigene Formatierungssprache, deren Gebrauch viel Aufmerksamkeit erfordert. PROTEXT hat zwei Stufen der HILFE-Funktion: In der Kurzform sieht man (ohne Beschädigung des bisher eingegebenen Textes) alle Befehle als Kombination von Tastendrücken dokumentiert. In der Langform findet man zu jedem Tastendruck eine ausführliche Dokumentation auf dem Bildschirm, womit ein Lernsystem zur Einarbeitung gegeben ist. Die prinzipiell guten interaktiven Fähigkeiten unserer derzeitigen Computer sind damit benutzerfreundlich in deutscher Sprache realisiert, ähnlich wie bei der HELP-Funktion des Wordstar, die sich ja auf den Einarbeitungsgrad des Benutzers einstellen läßt.

PROTEXT wird in zwei EPROMs für deutsche Tastaturbelegung und entsprechenden Zeichensatz für den Bildschirm geliefert. Das Programm wird von Disketten für CBM 8050 oder 8250 geladen, auf die die erfaßten Texte dann auch abgespeichert und wieder geladen werden können. Es zeigte sich jedoch, daß man offensichtlich nicht mit einer single drive Floppy auskommt, in die man abwechselnd die Programm- und dann die Datendiskette einlegt. Keine Schwierigkeiten treten jedoch dann ein, wenn man auch die Texte auf die Programmdiskette abspeichert (was allerdings nicht immer so ganz erwünscht ist). Zu bemerken ist weiter, daß die Programmversion 3.00 auf den CBM 8032 zugeschnitten ist. Sie läuft auch auf dem 8296, macht jedoch von dessen großen Speicher noch keinen Gebrauch. Hier soll die Version 3.02, die jetzt in Umlauf kommt, mit erweiterten Leistungsmerkmalen die auf 8296 angepaßte sein.

2. Leistungsmerkmale
Die für den Bildschirm üblichen und bequemen Editierbefehle sollen hier nicht vertieft werden. Es gilt wieder die Regel: Das was auf dem Bildschirm ist, das wird auch so abgespeichert. Im übrigen gibt es viele Parallelen zum Wordstar. Text wird absatzweise als Endlostext eingetippt. Wenn ein längers Wort nicht mehr in eine Zeile paßt, dann wird es ohne Trennung automatisch ganz in die Folgezeile gezogen. Die Zeilembreite für die Bildschirmeingabe läßt sich von normal 80 Zeichen auf 120 umstellen. Alle Such- und Tabulierfunktionen (die auch repetiert werden können) überstreichen dann die volle Textbreite. Man kann das Textfenster für die Anzeige auf den interessierenden Bereich legen.

Die Formatierungen für Drucker sind weitgehend unabhängig von denen für den Bildschirm. Ihre Leistungsfähigkeit wird im nächsten Abschnitt beschrieben. Es besteht jedoch die Möglichkeit, das, was auf dem Drucker erscheinen soll, sich im Vorwege auf dem Bildschirm anzeigen zu lassen, und zwar eben auch formatiert. Dabei kann man dem System mitteilen, wieviele Schreibstellen je Zeile benutzt werden sollen. Man definiert ferner die Trenntiefe, d.h. die Stelle am Zeilenende, ab der die automatische Silbentrennung einsetzen soll. Diese Dienste stehen sowohl für Block- als auch für Flattersatz zur Verfügung.

Weiterhin sind folgende Eigenschaften bemerkenswert: Texte können zu einem globalen File verkettet werden. Andererseits ist es möglich, eine sog. Jobdatei zu schaffen, die dann die Namen und die Reihenfolge der Files zu einer globalen Datei für den Ausdruck und für Textänderungen zusammenstellt. Bei der vorliegenden Version 3.00 von Protext klappten beim Autor allerdings diese globalen und Jobdienste nicht. Das soll 3.02 bringen. Hilfreiche Funktionen sind auch: Wort löschen, Springe zu Zeile Nr., Datum einziehen, Alpha-Lock, Einfügepunkt setzen/suchen, Absatz erzeugen, zwei Absätze zusammenfügen und die Archivfunktion, mit der man Textfiles lesen kann, ohne den gegenwärtig im Computer enthaltenen Text zu stören.

3. Ausgabeformatierung
Die Formatierung für den Bildschirm wurde schon erwähnt. Nun diejenige für Drucker. Das Seitenbild kann gestaltet werden durch die Schreibbreite, die Zeilenzahl/Seite, Kopf- und Fußleisten, Setzen des linken und rechten Randes, Silbentrennung ein/aus, Trennungstiefe, Druck aus, Druckpause mit Ausgabe einer Nachricht an den Bediener, Leerzeilen usw..

Für die Ausgabe stehen einige Druckertreiberprogramme zur Verfügung, und zwar für die Geräte CBM 8510/12 und 8515, juki 10/12, Epson, CBM 8028-Einzug und Olympia. Diese Treiber enthalten ganze Tabellenwerke, die auf die besonderen Bedürfnisse des Benutzers abgeändert werden können, und zwar für Gerätenummer, Sekundäradresse, IEEE-Steuerzeichen usw.. Innerhalb des

Textprogrammes können Sonderzeichen definiert werden, so daß eine Codeumsetzung erfolgt. Auch kann die Belegung der Tastatur mit einer Tabelle geändert werden. Die Anpassung der Ausgabe ist damit wirklich großzügig gelöst und dürfte den meisten vorkommenden Fällen gerecht werden können.

4. Zusammenfassung
PROTEXT ist ein leistungsfähiges Textverarbeitungsprogramm mit einer guten Bedienerführung, die eine sichere und schnelle Einarbeitung ermöglicht. Bei der vorliegenden Version wurde vermißt, daß es keinen zusätzlichen Textspeicher im Computer gibt, aus dem man mit Codeworten Standardbegriffe oder -texte schnell hereinziehen kann. Das soll erst die Version 3.02 bringen, ebenso die zuverlässige globale Änderungsfunktion bzw. Jobfunktion beim Ändern.

Der vorstehende Text wurde von PROTEXT versuchsweise auf eine Schreibmaschine Olivetti ET-221 mit Interface für den Bus IEEE 488 ausgegeben.

□□

Roland Löhr

Assemblerbeispiele für 6805/68705

In Heft 35 dieser Zeitschrift wurde ein Cross-Assembler unter FORTH für 6805/68705 vorgestellt. Um dessen Gebrauch transparenter zu machen, nachfolgend einige Formulierungsbeispiele. Darin sind alle Adressierungsarten berücksichtigt, ferner der Gebrauch der Kontrollstrukturen und der Assembleranweisungen, zur Ablage von Bytes, Wörtern, Strings, zur Zuweisung von Speicherraum (.RMB) und für die Bildung externer Symbole und interner Label.

```
        ( FORMULIERUNGSBEISPIELE FÜR FORTH-ASSEMBLER 6805/68705)

FORGET TASK  : TASK ;
0 ORG            ( SETZE VIRTUELLEN PROGRAMMZÄHLER)
2000 ASSIGN      ( ABLAGEBEREICH FÜR DEN ERZEUGTEN CODE)
                 ( KOMMENTARE BEGINNEN MIT KLAMMER AUF UND SPACE)
                 ( SIE ENDEN MIT KLAMMER ZU: )
( INHERENT-BEFEHLE)
NEGA,
NEGX,
TXA,

% 101 LDA,        ( BINARES ARGUMENT, LADE A AUS ADRESSE 5)
DDRA STA,        ( STORE A INS DATENRICHTUNGSREGISTER A)
4 # LDX,         ( LADE X, DIREKTOPERAND)

( INDIZIERTE ADRESSIERUNG)
 66 ,X LDA,      ( IX1 INDIZIERT)
678 ,X LDA,      ( IX2 INDIZIERT)
(X)    LDA,      ( IX ADRESSIERT, INDIREKT MIT X FÜR SEITE 0)
555    LDA,      ( EXTENDED ADRESSIERT, ABSOLUTE ADRESSE)

( KONTROLLSTRUKTUREN)
.GT              ( CONDITIONAL, ABFRAGE VOR IF,)
IF,   ROLA,      ( IF .. ELSE .. THEN)
ELSE, RORA,
THEN,
```

```
BEGIN,              ( BEGIN .. WHILE .. REPEAT)
 ROLA,
.CS WHILE,
 INCX,
REPEAT,

( ASSEMBLER-ANWEISUNGEN)
100 ORG             ( SETZE KUNSTLICHEN PROGRAMMZAHLER AUF 100)
  55 .FCB           ( LEGE KONSTANTE AB)
3456 .FDB           ( LEGE WORT AB)
   5 .RMB           ( RESERVIERE 5 BYTE SPEICHERPLATZ)
" STRING"           ( ABLAGE EINES STRINGS)
100 LABEL L1        ( BILDUNG EINES EXTERNEN SYMBOLS)
L1 JMP,             ( SYMBOLISCHES SPRUNGZIEL)
BHI,                ( CONDITIONAL, DAS NUR SEINEN OPCODE ABLEGT)
4 WR1               ( OFFSET 4 DAZU INS RAM SCHREIBEN)
VLOC LABEL L2       ( BILDUNG EINES LABELS MIT WERT DER STERNADRESSE)
L2 JMP,             ( SPRUNG AUF DIESES LABEL, HIER AUF SICH SELBST)

( BRANCH ON BIT SET OR CLEAR)
BEGIN,
 PORTB .BC5 WHILE,  ( SOLANGE BIT 5 IN PORT B NOCH CLEAR)
REPEAT,             ( WARTESCHLEIFE)

FINIS               ( ENDE DES QUELLPROGRAMMES)
```

Aus der Branche

Fifth Generation and Super Computers. Internationales Symposium in Rotterdam, 11.-13. Dez. 1984. Ausskunft: c/o Rotterdam Tourist Office, Stadthuisplein 19, NL-3012 AR Rotterdam.

Euromicro-Symposium und europ. Mikromaus-Wettbewerb 28.-30. Aug. 1984 in Kopenhagen mit Lehrveranstaltungen, Hauptvorträgen, techn.-wiss. Sitzungen. Weitere Informationen: Dr. Harald Schumny, PTB, Lab. 7.41, Bundesallee 100, 3300 Braunschweig.

FORCE Computers (68000 VMEbus) jetzt auch im Vertrieb der SYNELEC Datensysteme GmbH, Postfach 15 17 27, 8000 München 15, Tel.: 089 - 725 30 81.

VALVO GmbH, Hamburg: Auf der Hannover-Messe wurde das EUROM für BTX vorgeführt. Es enthält alle für den Bildschirmtext (CEPT) notwendigen Attribute. Das 40-polige IC erlaubt einen sehr kompakten Aufbau eines Dekoders und ist in seiner Busstruktur bereits für das Zusammenwirken mit 16 bit-Prozessoren ausgelegt. Einbau in Serienprodukte wohl ab Herbst 1984.
PDV-bus-Controller MEE 3000 für Prozeßdatenverarbeitung und serielle Datenübertragung zwischen max. 255 Stationen mit 1 Mbis/sec.. Der Baustein wurde zur Hannover-Messe angekündigt.
Infos: VALVO GmbH, UB Bauelemente, Burchardstr. 19, 2000 Hamburg 1.

Pufferspeicher für Computer/Drucker mit IEEE 488-Schnittstelle werden von der Fa. Wiesemann in 5600 Wuppertal 2, Postfach 201 605 angeboten mit 32, 64, 96 und 120 KB. Tel.: 0202-50 50 77.

Literaturrecherchen für EDV und Elektronik führt die Firma M+C Micro-Computer GmbH in 4018 Langenfeld, Karlstr. 17 durch. T. 021 73-7 65 77.

6502 Controllerkarten, auch mit BASIC bietet die Fa. Michael Nowak in 1000 Berlin an, G.-Müller-Str. 15, T. 030-784 91 93. Europakarten mit 6502, VIA 6522, ACIA 6551, bis 16 K Ram, bis 32 K EPROM.

1/2 Europakarte mit CPU R6511Q, bis 32 KB EPROM und 16 KB RAM, serieller Port mit VG-64-Leiste für die Portleitungen und Buserweiterungsstecker und mit KIM-1-ähnlichem Betriebsystem von der Fa. Brühl Elektronik, Hegelstr. 10, 8500 Nürnberg 10, T. 0911 - 35 90 88. Bei Redaktionsschluß ging dem Herausgeber ein Exemplar der sauber verarbeiteten Karte zu. Die CPU hat bekanntlich 32 Portpins und eignet besonders für größere Steueraufgaben. Die Karte wird über die serielle Schnittstelle angesprochen, wenn sie zur Entwicklung benutzt wird. Im Verlaufe soll auch das RSC-FORTH für sie erhältlich sein. Wir berichten weiter.

Karl-Heinz Dose, 2060 Bad Oldesloe

Berechnung der ATN-Funktion

auf AIM 65/PC 100

Obwohl die ATN-Funktion bei vielen technischen Berechnungen benötigt wird und Grundlage für die Berechnung aller weiteren Arcus-Funktionen ist, wird sie im BASIC des AIM 65 recht stiefmütterlich behandelt. Zum einen, weil sie nicht vollständig implementiert ist, zum anderen, weil das von Rockwell angegebene Programm auf einem sehr schlechten Näherungspolynom basiert. Besonders störend ist, daß auch markante Werte dieser Funktion nicht richtig approximitert werden. Beispiel:

arctan(1)=π/4 =0,785 ≙ 45 Grad

Der AIM liefert demgegenüber für ATN(1)= 0,794, entsprechend 45,5 Grad. Es entsteht also eine Abweichung bereits in der 2. Stelle hinter dem Komma.

Dieser Fehler kann behoben werden durch geeignete Veränderung der Koeffizienten des Näherungspolynoms, ohne den Grad des Polynoms (hier: N=23) zu vergrößern. Dazu müssen im ATN-Programm von Rockwell einige DATA-Zeilen verändert werden. Der besseren Übersicht wegen habe ich das Programm noch einmal vollständig aufgelistet. Die Zeilen 1200 bis 1310 enthalten die neuen Koeffizienten, die im BASIC als normalisierte Dualzahl mit fünf Bytes darzustellen sind.

Die Funktion kann natürlich auch unmittelbar mit einem Maschinenprogramm realisiert werden, das bei entsprechender Änderung der Startadresse auch in ein EPROM abgespeichert werden könnte. Im angefügten Assemblerausdruck ist zusätzlich in der Kommentarspalte der Wert der Koeffizienten noch einmal als Dezimalzahl angegeben.

Nach der BASIC-Initialisierung müssen beide Programme nur einmal durchlaufen werden. Im übrigen gelten die gleichen Bemerkungen wie für das Originalprogramm von Rockwell. - Der maximale Fehler des veränderten Polynoms liegt bei 10 hoch -8 und hat damit wohl für alle technischen Berechnungen einen akzeptablen Wert.

```
1000 REM ************************************
1010 REM ATN-FUNKTION MIT NEUEN KOEFFIZIENTEN
1020 REM ************************************
1030 REM INIT ATN PC 100
1040 POKE 188,189
1050 POKE 189,15
1060 POKE 127,128
1070 POKE 128,15
1080 RESTORE
1090 READ A
1100 READ B
1110 IF B<0 THEN RETURN
1120 POKE A,B
1130 A=A+1
1140 GOTO 1100
1150 REM ************************************
1160 REM KOEFFIZIENTEN-TABELLE
1170 DATA 3968 : REM STARTADRESSE HEX (0F80)
1180 DATA 11 : REM ANZAHL-1
1190 REM 12 NEUE KOEFFIZIENTEN
1200 DATA 118,150,50,109,227
1210 DATA 121,9,234,134,63
1220 DATA 122,237,184,58,91
1230 DATA 124,2,201,144,122
```

```
1240 DATA 124,213,254,24,234
1250 DATA 125,17,247,158,34
1260 DATA 125,182,253,153,252
1270 DATA 125,98,246,152,13
1280 DATA 126,146,64,68,242
1290 DATA 126,76,204,53,222
1300 DATA 127,170,170,168,86
1310 DATA 129,0,0,0,0
1320 REM TABELLENENDE
1330 REM ****************************
1340 REM ANFANG MASCHINENPROGRAMM
1350 REM STARTADRESSE HEX(0FBD)
1360 DATA 165,174,72,16,3
1370 DATA 32,184,204,165,169
1380 DATA 72,201,129,144,7
1390 DATA 169,251,160,198,32
1400 DATA 78,200,169,128,160
1410 DATA 15,32,68,205,104
1420 DATA 201,129,144,7,169
1430 DATA 78,160,206,32,143
1440 DATA 197,104,16,3,76
1450 DATA 184,204,96,-1
1460 END
```

```
0000            ;ATN-FUNKTION MIT VERBESSERTEN KOEFFIZIENTEN
0000            ;*******************************************
0000                    * = $0F80
0F80
0F80  A98F      LDA #<ATN       ; ATN-VEKTOR IN ZERO-PAGE
0F82  85BC      STA $BC         ;     UND
0F84  857F      STA $7F         ; BASIC-OBERGRENZE AENDERN
0F86  A90F      LDA #>ATN
0F88  85BD      STA $BD
0F8A  8580      STA $80
0F8C  4C82E1    JMP $E182       ; RUECKSPRUNG MONITOR
0F8F
0F8F ATN        A5AE    LDA $AE         ; START MASCHINENPROGRAMM
0F91            48      PHA
0F92            1003    BPL ATN1
0F94            20B8CC  JSR $CCB8
0F97 ATN1       A5A9    LDA $A9
0F99            48      PHA
0F9A            C981    CMP #$81
0F9C            9007    BCC ATN2
0F9E            A9FB    LDA #$FB
0FA0            A0C6    LDY #$C6
0FA2            204EC8  JSR $C84E
0FA5 ATN2       A9BF    LDA #<ATNTAB
0FA7            A00F    LDY #>ATNTAB
0FA9            2044CD  JSR $CD44
0FAC            68      PLA
0FAD            C981    CMP #$81
0FAF            9007    BCC ATN3
0FB1            A94E    LDA #$4E
0FB3            A0CE    LDY #$CE
0FB5            208FC5  JSR $C58F
0FB8 ATN3       68      PLA
```

```
0FB9            1003    BPL ATN4
0FBB            4CB8CC  JMP $CCB8
0FBE ATN4       60      RTS
0FBF
0FBF            ;KOEFFIZIENTEN-TABELLE
0FBF ATNTAB     0B      .BYT 11          ; ANZAHL-1
0FC0            76      .BYT 118,150,50,109,227  ; -5.72956044E-4
0FC1            96
0FC2            32
0FC3            6D
0FC4            E3
0FC5            79      .BYT 121,9,234,134,63  ; 4.20886569E-3
0FC6            09
0FC7            EA
0FC8            86
0FC9            3F
0FCA            7A      .BYT 122,237,184,58,91  ; -.0145092554
0FCB            ED
0FCC            B8
0FCD            3A
0FCE            5B
0FCF            7C      .BYT 124,2,201,144,122  ; .031930508
0FD0            02
0FD1            C9
0FD2            90
0FD3            7A
0FD4            7C      .BYT 124,213,254,24,234  ; -.0522442792
0FD5            D5
0FD6            FE
0FD7            18
0FD8            EA
0FD9            7D      .BYT 125,17,247,158,34  ; .0712730746
0FDA            11
0FDB            F7
0FDC            9E
0FDD            22
0FDE            7D      .BYT 125,182,253,153,252  ; -.089350894
0FDF            B6
0FE0            FD
0FE1            99
0FE2            FC
0FE3            7D      .BYT 125,98,246,152,13  ; .110821907
0FE4            62
0FE5            F6
0FE6            98
0FE7            0D
0FE8            7E      .BYT 126,146,64,68,242  ; -.142823293
0FE9            92
0FEA            40
0FEB            44
0FEC            F2
0FED            7E      .BYT 126,76,204,53,222  ; .199997751
0FEE            4C
0FEF            CC
0FF0            35
0FF1            DE
0FF2            7F      .BYT 127,170,170,168,86  ; -.333333264
0FF3            AA
0FF4            AA
```

```
0FF5        A8
0FF6        56
0FF7        81      .BYT 129,0,0,0,0  ; 1
0FF8        00
0FF9        00
0FFA        00
0FFB        00
0FFC                .END ATN
```

Bücher

Hilf, W.; Nausch, A.: M68000 Familie, Teil 1, Grundlagen und Architektur. te-wi Verlag, München 1984, ca. 576 S., DM 79,-, ISBN 3-921 803 16-0. Die Autoren sind Mitarbeiter der Fa. Motorola, W. Hilf als langjähriger Schulungleiter und A. Nausch als Applikationsingenieur. Man darf ihnen bestätigen, daß sie mit ihren Erfahrungen, 'aus erster Quelle', ein informatives und sorgfältig aufgemachtes Buch herausgegeben haben. Die grobe Gliederung ist: Allgemeines, Aufbau des M68000, Adressierungsarten (61 S.), Befehlssatz (372 S.), Die verschiedenen Versionen des M68000 und Anhänge. An der Stoffeinteilung sieht man, daß vor allem in Hinsicht auch auf den bald erwarteten zweiten Band (Schwerpunkt Programmierung und Interfacing) Schwerpunkte bei den Adressierungsarten und bei der ausführlichen Besprechung der Befehle und ihrer Wirkungsweise gesetzt wurden. Hier gibt es einen gut lesbaren verständlichen Text, der in seinem Inhalt weit über das hinausgeht, was z.B. in den Firmenschriften zum 68000 und in entsprechenden Büchern der Firma Valvo enthalten war. So ist der Hauptteil für die Befehle jeweils pro Befehl gegliedert in die Abschnitte: Name und Wirkung in Kurzform, Assembler-Syntax, Operation, Wirkung auf Flags, Befehlsbeschreibung ausführlich, Opcode und Erklärung, zugelassene Adressierungsarten für Quelle und Ziel, Ausführungszeiten des Befehls für verschiedene Adressierungsarten und Zahl der benötigten Bytes für den Befehl, Beispiel, typische Anwendung. Vor allem die letzteren Abschnitte sind jeweils hilfreich, zumal wir Darstellungen in umrahmten Tabellen finden. Oft genug sind zudem Schemazeichnungen für die Abläufe enthalten. – Unter den zahlreichen Büchern, die von Osborne/McGraw-Hill in englisch und von te-wi in deutsch zu den einzelnen Prozessortypen herausgegeben wurden, darf dieses als das wohl gelungendste und in den einzelnen Abschnitten als das ausführlichste bezeichnet werden. Es gehört auf jeden Fall in die Hand der Systemplaner und der Assemblerprogrammierer. Es ist aber auch interessant, wenn man sich nur für die Architektur der leistungsfähigen CPU-Familie interessiert.

Thies, K-D.: Die 8085/8086 Interfaces - Funktion und Programmierung intelligenter Mikroprozessor-Peripherie. te-wi Verlag, München 1983, 633 S., DM 89,--, ISBN 3-921 803-23-3. Auch dieser Autor hat langjährige Schulungserfahrung. Sein Stil ist vorbildlich für die Behandlung komplexer technischer Zusammenhänge, er verbindet die Verläßlichkeit und Vollständigkeit in Datenblättern mit einer bildlichen und sofort erfaßbaren Darstellungseise. Das Buch beschreibt die wichtigsten Peripheriebausteine zu o.a. Mikroprozessoren. Wir finden folgende Hauptabschnitte: Interfaces im System, 8205: 1-aus-8 Binärdekoder, 8282 Oktal-Latch, 8085 Interruptsteuerung on chip, Hypothetischer A/D-Wandler, 8255A Parallel-E/A, 8253 Zähler und Zeitgeber, 8251 Serielle E/A, 8237 DMA-Controller, 8259 Interruptstuerung. Die einzelnen Kapitel sind für die verschiedenen Anwendungen z.T. noch weit gegliedert. Wir finden immer aufschlußreiche Schemazeichnungen, die die Anschlußbelegungen an den Bausteinen erkennen lassen und die ihre Einbettung in das System und die hervorgehobenen Signalwege schnell erkennen lassen, dazu den Begleittext und auch Programmabschnitte in Assemblersprache. Das Buch macht damit innerhalb eines mit den Bausteinen aufgebauten Systems das Zusammenspiel der Signale und der Programmierung transparent und dürfte sich nicht nur für Betreiber von 8xxx-Systmen anbieten, sondern auch für solche, die Bausteine dieser Familie mit bestimmten Eigenschaften in anderen Systemen einsetzen möchten.

Osborne, A.; Eubanks, G. jr.; McNiff, M.: C BASIC Anwenderhandbuch. Bei McGraw-Hill, Hamburg 1984, 224 S. DM 39,80, ISBN 3-89028-006-4. Eubanks wird als der Erfinder des C BASIC bezeichnet, das auf CP/M-Systemen sehr beliebt ist und das auch kompiliert. Das vorliegende Buch ist eine gelungene Übersetzung aus dem Amerikanischen. Wir finden klare deutsche Formulierungen und damit einen gut lesbaren Text. Das Standardwerk beleuchtet die Sprache von allen Seiten her, insbesondere auch die Hantierung des Computers unter C BASIC. Zahlreiche Skizzen, Programmausschnitte und Protokolle von Dialogen unterlegen den Inhalt, so daß man deutlich sieht, worauf es ankommt. Wichtige Abschnitte betreffen die Datentypen und Operationen, die Ein- und Ausgabe, die Programmorganisation und modulares Programmieren sowie die Dateibehandlung. Übersichten runden den Stoff ab. Obwohl der Rezensent kein Kenner des Sprachdialektes ist, hat er den Eindruck, daß er sich mit diesem Buch zuverlässig einarbeiten kann.

Ing. Louis Rutten, B-3980 Tessenderlo

Step by Step and Walk for CBM

(Anm. des Hrsg.: Das vorliegende Programm wurde vor allem für Ausbildungszwecke entwickelt. Es erlaubt es, Maschinenprogramme -M.L.- und BASIC-Programme im Modus Step oder Walk mit Registeranzeige zu durchfahren. Der Disassembler wurde vom Autor aus anderer Quelle übernommen, um die mnemonischen Befehle ausschreiben zu können. Wenn man letztere nicht möchte, erhält man nur Hexcode angezeigt. In diesem Falle ist der JSR-Befehl ab Adresse 3984 in 3x'EA'=NOP abzuändern. Die vorliegende Version gilt für CBM 3032. Ein Umschreiben auf 4032 ist möglich, wenn man die Adreßreferenzen im Text berücksichtigt.)

This program gives the opportunity to execute an other program step by step or to walk through it until certain chosen breakpoints are reached.

Written to examine how the microprocessor works, it can also efficiently be used to test and debug M.L. programs or to follow BASIC and system routines.

Four modes can be chosen: step by step BASIC (BS), walk BASIC(BW), step by step M.L. (MS) and walk M.L. (MW).

The program has following advantages.

- A clearly display in the mode STEP, as shown on figure 1. Even print statements are executed on the bottom-side of the screen and the contents of absolute addresses will be displayed in binary form (which is useful to examine VIA and PIA).
- False opcodes are detected, so that de microprocessor doesn't crash; in that case the execution will be stopped and the address that contains the false code will be displayed.
- The execution can always be stopped (even in an endless itteration).
- On a breakpoint one can change the mode from walk to step. Changing from step to walk can always be done.
- The program doesn't destroy BASIC. Nearly all BASIC statements are possible in BW mode. One can stay in that mode until the STOP button is pressed (try RUN and GOTO).

For them who like to imply this program into a monitor (only the beginning and end have to be changed), they must know that $BA contains a flag to indicate the mode. For BASIC bit 6=0, for M.L. bit 6=1; for step bit 7=0, for walk bit 7=1. Also $10 contains a flag; this flag indicates whether the operand is an absolute addres (0) or not (1).

Lit. - BCM, Banneux, Belgique: Programmes internes PET CBM.
- MC Juli 83: 6502-Programmiermodell (Rudiger Hambsch)

FORMATS

VECTOR:
SYS14336 <CR>
◆V XXXX YYYY <CR>
Sets two breakpoints for walk. Always two addresses have to be given, this being the case it can be a dummy ($FFFF).

M.L. STEP:
SYS14336 <CR>
◆S XXXX <CR>
XXXX is the startaddress of a program that will be executed step by step. Some buttons have a function:

"<": each time this button is pressed the next instruction will be executed.

"SP": holding de spacebar down, the instructions have been successively executed.
".": changes mode to walk.
"STOP": stops the execution.

M.L. WALK
SYS 14336 <CR>
◆SHIFT S XXXX <CR>
Starts a program in walk mode. The function of the buttons is the same. "." changes the mode in step.

BASIC STEP
As exemple: 10 PRINT "K"
20 SYS 14380
30 FOR K=1 TO....
Line 20 starts the step mode for the rest of the program (unless one changes the mode).

BASIC WALK
SYS 14385:RUN or SYS 14385 <CR>
Unless the "STOP" key is pressed, all BASIC lines and direct statements are executed in the walk mode.
($FFD2 is a certain breakpoint and then one can change the mode to step if wished-for.)
The display in walk mode is as shown below.

21 3E FF 00 F6 FFD2 4C 32 F2 JMP F232

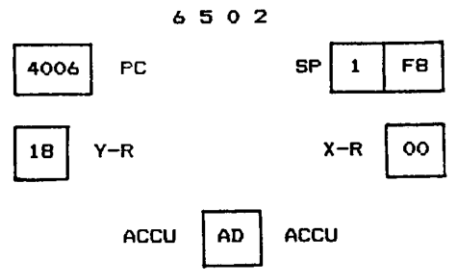


Fig. 1

1	0	1	0	0	0	0	0
N	V	-	B	D	I	Z	C

AD 02 40 LDA 4002 10101101 NA
8D 10 40 STA 4010 10101010 VOOR
AB

```
CRSR MOVEMENTS
36FB D8          CLD
36FC A9 13       LDA #$13      HOME
36FE 20 D2 FF    JSR $FFD2
3701 60          RTS
3702 A9 11       LDA #$11      DOWN
3704 D0 08       BNE $370E
3706 A9 1D       LDA #$1D      RIGHT
3708 D0 04       BNE $370E
370A A2 04       LDX #$04      CR
370C A9 0D       LDA #$0D
370E 20 D2 FF    JSR $FFD2
3711 CA          DEX
3712 D0 FA       BNE $370E
3714 60          RTS

PRINT REGISTERS
3715 B9 42 03    LDA $0342,Y
3718 20 75 E7    JSR $E775
371B 60          RTS

PRINT BYTE BIN.
371C EA          NOP
371D A0 01       LDY #$01      OPERAND (ADDRESS)
371F B1 FB       LDA ($FB),Y   IN $05/06
3721 85 05       STA $05
3723 C8          INY
3724 B1 FB       LDA ($FB),Y
3726 85 06       STA $06
3728 A0 00       LDY #$00
372A B1 05       LDA ($05),Y   GET BYTE
372C 48          PHA           SAVE BYTE
372D A0 08       LDY #$08      FOR 8 BIT
372F E8          INX
3730 68          PLA           REST TO ACCU
3731 0A          ASL
3732 48          PHA           SAVE REST
3733 A9 18       LDA #$18      1 OR 0
3735 2A          ROL
3736 9D 20 83    STA $8320,X   ON SCREEN
3739 88          DEY
373A D0 F3       BNE $372F
373C 68          PLA           0 TO ACCU
373D 85 10       STA $10       FLAG ABS. ADDR.=0
373F 60          RTS

LENGTH OF INSTR.
3740 E8          INX           X=0
3741 86 B6       STX $B6       LENGTH=0
3743 AA          TAX           SAVE OPCODE
3744 C9 20       CMP #$20      "JSR"=3 BYTES
3746 F0 26       BEQ $376E
3748 29 9F       AND #$9F
374A F0 24       BEQ $3770     1 BYTE
374C 29 0F       AND #$0F
374E C9 08       CMP #$08
3750 F0 1E       BEQ $3770     1 BYTE
3752 C9 0A       CMP #$0A
3754 F0 1A       BEQ $3770     1 BYTE
3756 8A          TXA
3757 29 10       AND #$10
3759 F0 09       BEQ $3764
375B 8A          TXA
375C 29 0F       AND #$0F
375E C9 09       CMP #$09
3760 B0 0C       BCS $376E     3 BYTES
3762 90 0B       BCC $376F     2 BYTES
3764 8A          TXA
3765 29 09       AND #$09
3767 F0 06       BEQ $376F     2 BYTES
3769 B0 02       BCS $376D     3 BYTES ABS. ADDR.
376B 90 02       BCC $376F     2 BYTES
376D C8          INY           Y=4 ABS. ADDR.
376E C8          INY           NUMB. BYTES
376F C8          INY
3770 C8          INY
3771 8A          TXA           OPCODE TO ACCU
3772 60          RTS
```

```
HEX CODE INSTR.
3773 B1 FB      LDA ($FB),Y GET BYTE
3775 20 75 E7   JSR $E775   PRINT BYTE
3778 20 CD FD   JSR $FDCD   SPACE
377B C8         INY
377C C4 B6      CPY $B6     ALL BYTES?
377E D0 F3      BNE $3773
3780 A0 00      LDY #$00
3782 60         RTS

PROC.STAT.BIN.
3783 AD 42 03   LDA $0342   GET P
3786 A2 08      LDX #$08    FOR 8 BIT
3788 48         PHA         SAVE BYTE
3789 68         PLA         REST TO ACCU
378A 0A         ASL         GET NEXT BIT
378B 48         PHA         SAVE REST
378C A9 18      LDA #$18    0 OR 1
378E 2A         ROL
378F 20 D2 FF   JSR $FFD2   PRINT
3792 A9 1D      LDA #$1D    CRSR RIGHT
3794 20 D2 FF   JSR $FFD2
3797 20 D2 FF   JSR $FFD2
379A CA         DEX
379B D0 EC      BNE $3789
379D 68         PLA         CORRECTION SP
379E 60         RTS

PC IN $05/06
379F AD 40 03   LDA $0340
37A2 85 06      STA $06
37A4 AD 41 03   LDA $0341
37A7 85 05      STA $05
37A9 A0 00      LDY #$00
37AB 60         RTS

EXIST OPCODE?
37AC 20 E3 37   JSR $37E3   PC IN $FB/FC
37AF B1 FB      LDA ($FB),Y GET OPCODE
37B1 C9 A2      CMP #$A2    "BIT" EXIST
37B3 F0 1A      BEQ $37CF
37B5 48         PHA         SAVE OPCODE
37B6 29 0F      AND #$0F
37B8 C9 02      CMP #$02    ......10 NO CODE
37BA F0 14      BEQ $37D0
37BC 4A         LSR
37BD 90 03      BCC $37C2
37BF 4A         LSR
37C0 B0 0E      BCS $37D0   ......11 NO CODE
37C2 68         PLA
37C3 48         PHA
37C4 A2 1A      LDX #$1A
37C6 DD 69 3B   CMP $3B69,X COMP. WITH FALSE
37C9 F0 05      BEQ $37D0   CODES
37CB CA         DEX
37CC 10 F8      BPL $37C6
37CE 68         PLA         OPCODE TO ACCU
37CF 60         RTS

END
37D0 EA         NOP
37D1 EA         NOP
37D2 EA         NOP
37D3 20 D0 FD   JSR $FDD0   CR
37D6 20 E3 37   JSR $37E3   PC IN $FB/FC
37D9 20 6A E7   JSR $E76A   PRINT PC
37DC AE 46 03   LDX $0346   RESTORE SP AND
37DF 9A         TXS         TO B. READY
37E0 4C 89 C3   JMP $C389

PC IN $FB/FC
37E3 AD 41 03   LDA $0341
37E6 85 FB      STA $FB
37E8 AD 40 03   LDA $0340
37EB 85 FC      STA $FC
37ED A0 00      LDY #$00
37EF 60         RTS
```

```
GET 2 ADDRESSES
37F0 20 EB E7    JSR $E7EB
37F3 20 A7 E7    JSR $E7A7
37F6 20 97 E7    JSR $E797
37F9 20 EB E7    JSR $E7EB
37FC 20 A7 E7    JSR $E7A7
37FF 60          RTS

M.L. START
3800 A9 DA       LDA #$DA     PRINT◆
3802 20 D2 FF    JSR $FFD2
3805 20 CF FF    JSR $FFCF    GET CHAR.
3808 C9 DA       CMP #$DA     DROP◆
380A F0 F9       BEQ $3805
380C 08          PHP          CORRECTION SP
380D C9 56       CMP #$56     V?
380F F0 0A       BEQ $381B    THEN VECTOR
3811 C9 53       CMP #$53     S?
3813 F0 1F       BEQ $3834    THEN MS
3815 C9 D3       CMP #$D3     SHIFT S?
3817 F0 1B       BEQ $3834    THEN MW
3819 68          PLA          TO BASIC
381A 60          RTS

VECTOR
381B 20 F0 37    JSR $37F0    2 ADDRESSES
381E A2 04       LDX #$04     IN BREAKVECTORS
3820 B5 FA       LDA $FA,X
3822 9D 50 03    STA $0350,X
3825 CA          DEX
3826 D0 F8       BNE $3820
3828 68          PLA          CR AND TO BASIC
3829 4C D0 FD    JMP $FDD0

BASIC STEP
382C 08          PHP          SAVE P
382D A9 38       LDA #$38     FLAG BS
382F D0 03       BNE $3834

BASIC WALK
3831 08          PHP          SAVE P
3832 A9 B8       LDA #$B8     FLAG BW
3834 85 BA       STA $BA
3836 68          PLA          P TO TEMP.MEM.
3837 8D 42 03    STA $0342
383A D8          CLD
383B 18          CLC          PC+1 (FOR BS)
383C 68          PLA          IN TEMP.MEM.
383D 69 01       ADC #$01
383F 8D 41 03    STA $0341
3842 68          PLA
3843 69 00       ADC #$00
3845 8D 40 03    STA $0340
3848 8E 44 03    STX $0344    REG. TO TEMP.MEM.
384B 8C 45 03    STY $0345
384E BA          TSX
384F 8E 46 03    STX $0346
3852 24 BA       BIT $BA      MODE?
3854 50 0D       BVC $3863    IF BASIC NO INPUT
3856 20 F9 37    JSR $37F9    M.L. GET ADDRESS
3859 A5 FB       LDA $FB      FOR START IN
385B 8D 41 03    STA $0341    TEMP.MEM.
385E A5 FC       LDA $FC
3860 8D 40 03    STA $0340
3863 24 BA       BIT $BA      MODE?
3865 30 2A       BMI $3891    IF WALK NO LAY OUT
3867 A0 01       LDY #$01     FLAG ABS. ADDR.=1
3869 84 10       STY $10
386B 88          DEY
386C A9 93       LDA #$93     CLR
386E 20 D2 FF    JSR $FFD2
3871 A9 0F       LDA #$0F     PRINT LAY OUT
3873 85 C7       STA $C7
3875 A9 3A       LDA #$3A
3877 85 C8       STA $C8
3879 A9 69       LDA #$69
387B 85 C9       STA $C9
387D A9 3B       LDA #$3B
```

```
387F 85 CA       STA $CA
3881 B1 C7       LDA ($C7),Y
3883 20 D2 FF    JSR $FFD2
3886 E6 C7       INC $C7
3888 D0 02       BNE $388C
388A E6 C8       INC $C8
388C 20 C6 FC    JSR $FCC6
388F D0 F0       BNE $3881
3891 20 9F 37    JSR $379F    GET OPCODE
3894 B1 05       LDA ($05),Y
3896 20 B1 37    JSR $37B1    EXIST OPCODE?
3899 78          SEI
389A A9 C0       LDA #$C0     ENABLE INT. T1
389C 8D 4E E8    STA $E84E
389F A9 CC       LDA #$CC     INT. VECTOR IS
38A1 85 90       STA $90      $38CC
38A3 A9 38       LDA #$38
38A5 85 91       STA $91
38A7 CE 13 E8    DEC $E813    NO SYST. INT.
38AA A9 00       LDA #$00
38AC 8D 4B E8    STA $E84B    T1 ONE SHOT
38AF A2 70       LDX #$70     TIME TO REACH
38B1 8E 44 E8    STX $E844    NEXT INSTR.
38B4 8D 45 E8    STA $E845    IN T1
38B7 AE 46 03    LDX $0346    REG. AND PC
38BA 9A          TXS          ON STACK
38BB A2 00       LDX #$00
38BD BD 40 03    LDA $0340,X
38C0 48          PHA
38C1 E8          INX
38C2 E0 06       CPX #$06
38C4 D0 F7       BNE $38BD
38C6 68          PLA
38C7 A8          TAY
38C8 68          PLA
38C9 AA          TAX
38CA 68          PLA
38CB 40          RTI          JUMP TO PROGR.
38CC 20 7B FC    JSR $FC7B    REST. SYST. INT.
38CF A2 06       LDX #$06     PC AND REG.
38D1 68          PLA          TO TEMP.MEM.
38D2 9D 3F 03    STA $033F,X
38D5 CA          DEX
38D6 D0 F9       BNE $38D1
38D8 BA          TSX
38D9 8E 46 03    STX $0346
38DC 58          CLI
38DD A5 D9       LDA $D9      SAVE CHAR. TO
38DF 48          PHA          PRINT
38E0 24 BA       BIT $BA      MODE?
38E2 10 03       BPL $38E7    STEP
38E4 4C BF 39    JMP $39BF    WALK
38E7 A2 20       LDX #$20
38E9 A4 10       LDY $10      ABS. ADDRESS?
38EB D0 07       BNE $38F4    NO
38ED A2 16       LDX #$16     PRINT BIN.
38EF 20 1D 37    JSR $371D
38F2 A2 14       LDX #$14     COPY LINE
38F4 BD 47 83    LDA $8347,X
38F7 9D 1F 83    STA $831F,X
38FA CA          DEX
38FB D0 F7       BNE $38F4
38FD E8          INX          FLAG ABS.ADDR.=1
38FE 86 10       STX $10
3900 20 E3 37    JSR $37E3    PC IN $FB/FC
3903 20 FB 36    JSR $36FB    HOME
3906 A0 04       LDY #$04
3908 A2 03       LDX #$03
390A 20 02 37    JSR $3702
390D E8          INX
390E 20 06 37    JSR $3706
3911 20 6A E7    JSR $E76A    PRINT PC
3914 A2 10       LDX #$10
3916 20 06 37    JSR $3706
3919 20 15 37    JSR $3715    PRINT SP
391C 20 0A 37    JSR $370A
391F 88          DEY
3920 E8          INX
```

```
3921 20 06 37    JSR $3706
3924 20 15 37    JSR $3715    PRINT Y
3927 A2 12       LDX #$12
3929 20 06 37    JSR $3706
392C 88          DEY
392D 20 15 37    JSR $3715    PRINT X
3930 20 0A 37    JSR $370A
3933 88          DEY
3934 A2 0B       LDX #$0B
3936 20 06 37    JSR $3706
3939 20 15 37    JSR $3715    PRINT ACCU
393C 20 0A 37    JSR $370A
393F E8          INX
3940 20 06 37    JSR $3706
3943 20 83 37    JSR $3783    PRINT P BIN.
3946 A2 06       LDX #$06
3948 20 0C 37    JSR $370C
394B 88          DEY
394C A9 20       LDA #$20     EFFACE LINE
394E AA          TAX
394F 9D 47 83    STA $8347,X
3952 CA          DEX
3953 D0 FA       BNE $394F
3955 20 AF 37    JSR $37AF    EXIST OPCODE?
3958 20 40 37    JSR $3740    LENGTH INSTR.?
395B A2 3E       LDX #$3E
395D 84 B6       STY $B6
395F C0 04       CPY #$04     ABS. ADDRESS?
3961 D0 12       BNE $3975
3963 88          DEY
3964 84 B6       STY $B6
3966 C9 4C       CMP #$4C     IGNORE "JMP"
3968 F0 0B       BEQ $3975
396A C5 6C       CMP $6C
396C F0 07       BEQ $3975
396E 84 B6       STY $B6
3970 20 1D 37    JSR $371D    PRINT BIN.
3973 84 10       STY $10      FLAG ABS.ADDR.=0
3975 A0 00       LDY #$00
3977 20 73 37    JSR $3773    PRINT HEXCODE
397A A9 14       LDA #$14     INSTRUCTION
397C 24 BA       BIT $BA
397E 30 02       BMI $3982
3980 A9 0C       LDA #$0C
3982 85 C6       STA $C6
3984 20 84 3B    JSR $3B84    DISASSEMBLY
3987 20 D0 FD    JSR $FDD0
398A A4 C6       LDY $C6      CRSR READY FOR
398C B1 C4       LDA ($C4),Y  PRINT STATEMENT
398E C9 20       CMP #$20
3990 F0 05       BEQ $3997
3992 E6 C6       INC $C6
3994 4C 8A 39    JMP $398A
3997 68          PLA          RESTORE CHAR.
3998 85 D9       STA $D9      TO PRINT
399A 20 01 F3    JSR $F301    STOP?
399D C9 FB       CMP #$FB     IF "SPACE"
399F D0 03       BNE $39A4
39A1 4C 99 38    JMP $3899    NEXT
39A4 C9 FF       CMP #$FF     RELEASE BUTTON
39A6 D0 F2       BNE $399A
39A8 20 01 F3    JSR $F301    "STOP"?
39AB F0 0F       BEQ $39BC    YES
39AD C9 FF       CMP #$FF
39AF F0 F7       BEQ $39A8
39B1 C9 BF       CMP #$BF     IF "." CHANGE MODE
39B3 D0 EC       BNE $39A1
39B5 45 BA       EOR $BA
39B7 85 BA       STA $BA
39B9 4C 63 38    JMP $3863    NEXT
39BC 4C D3 37    JMP $37D3    END
39BF 20 01 F3    JSR $F301    "STOP"?
39C2 F0 F8       BEQ $39BC    YES
39C4 A2 06       LDX #$06     BREAKPOINT?
39C6 CA          DEX
39C7 CA          DEX
```

```
39C8 F0 3C      BEQ $3A06    NO
39CA 30 3A      BMI $3A06
39CC AD 40 03   LDA $0340
39CF DD 50 03   CMP $0350,X
39D2 D0 F2      BNE $39C6
39D4 CA         DEX
39D5 AD 41 03   LDA $0341
39D8 DD 50 03   CMP $0350,X
39DB D0 EA      BNE $39C7
39DD 20 D0 FD   JSR $FDD0    CR
39E0 20 E3 37   JSR $37E3    PC IN $FB/FC
39E3 20 6A E7   JSR $E76A    PRINT PC
39E6 20 CD FD   JSR $FDCD    SPACE
39E9 A0 04      LDY #$04     PRINT REG.
39EB B9 42 03   LDA $0342,Y
39EE 20 75 E7   JSR $E775
39F1 20 CD FD   JSR $FDCD
39F4 88         DEY
39F5 10 F4      BPL $39EB
39F7 20 CD FD   JSR $FDCD
39FA A9 1C      LDA #$1C     PRINT INSTR.
39FC 85 C6      STA $C6
39FE A0 00      LDY #$00
3A00 20 CD FD   JSR $FDCD
3A03 4C 55 39   JMP $3955    AND GO ON
3A06 20 AC 37   JSR $37AC    EXIST OPCODE?
3A09 68         PLA
3A0A 85 D9      STA $D9
3A0C 4C 99 38   JMP $3899    THEN NEXT
```

```
M 3A0F,3B83
3A0F 20 20 20 20 20 20 20 20 36 20 35 20 30 20 32 0D
3A1F 20 0D CF A3 A3 A3 A3 D0 20 20 20 20 20 20 20 20
3A2F 20 20 20 CF A3 D0 A3 A3 A3 D0 0D A5 20 20 20 20
3A3F A7 20 50 43 20 20 20 20 20 53 50 20 A5 31 A7 20
3A4F 20 20 A7 0D CC A4 A4 A4 A4 BA 20 20 20 20 20 20
3A5F 20 20 20 20 20 CC A4 BA A4 A4 A4 BA 0D 20 0D CF
3A6F A3 A3 D0 20 20 20 20 20 20 20 20 20 20 20 20 20
3A7F 20 20 20 CF A3 A3 D0 0D A5 20 20 A7 20 59 2D 52
3A8F 20 20 20 20 20 20 20 20 58 2D 52 20 A5 20 20 A7
3A9F 0D CC A4 A4 BA 20 20 20 20 20 20 20 20 20 20 20
3AAF 20 20 20 20 20 CC A4 A4 BA 0D 20 0D 20 20 20 20
3ABF 20 20 20 20 20 20 CF A3 A3 D0 0D 20 20 20 20 20
3ACF 41 43 43 55 20 A5 20 20 A7 20 41 43 43 55 0D 20
3ADF 20 20 20 20 20 20 20 20 20 CC A4 A4 BA 0D 20 0D
3AEF CF A3 D0 A3 A3 D0 A3 A3 D0 A3 A3 D0 A3 A3 D0 A3
3AFF A3 D0 A3 A3 D0 A3 A3 D0 0D A5 20 A7 A0 20 A7 A0
3B0F 31 A7 A0 20 A7 A0 20 A7 A0 20 A7 A0 20 A7 A0 20
3B1F A7 0D CC A4 BA A4 A4 BA A4 A4 BA A4 A4 BA A4 A4
3B2F BA A4 A4 BA A4 A4 BA A4 A4 BA 0D 20 0D 20 4E 20
3B3F 20 56 20 20 2D 20 20 42 20 20 44 20 20 49 20 20
3B4F 5A 20 20 43 11 11 20 20 20 20 20 20 20 20 20 20
3B5F 4E 41 11 9D 9D 56 4F 4F 52 0D FC 04 0C FA 14 1A
3B6F 1C F4 DC 34 3A 3C 44 54 5A 5C 74 7A 64 80 7C 89
3B7F DA 9C 9E D4 DB A2 00 A1 FB 20 DF 3B A2 03 20 25
```

ADDRESSES CBM3000 — CBM4000

$10 - $BB	$E797 - $D744	$FC7B - $FCC0
$C389 - $B3FF	$E7A7 - $D754	$FCC6 - $FD0B
$E76A - $D717	$E7EB - $D798	$FDCD - $D531
$E775 - $D722	$F301 - $F335	$FDD0 - $D534

DISASSEMBLER

```
3B84 A2 00      LDX #$00
3B86 A1 FB      LDA ($FB,X)
3B88 20 DF 3B   JSR $3BDF
3B8B A2 03      LDX #$03
3B8D 20 25 3C   JSR $3C25
3B90 A2 06      LDX #$06
3B92 E0 03      CPX #$03
```

```
3B94 D0 12       BNE $3BA8
3B96 A4 B6       LDY $B6
3B98 F0 0E       BEQ $3BA8
3B9A A5 FF       LDA $FF
3B9C C9 E8       CMP #$E8
3B9E B1 FB       LDA ($FB),Y
3BA0 B0 1C       BCS $3BBE
3BA2 20 CB 3B    JSR $3BCB
3BA5 88          DEY
3BA6 D0 F2       BNE $3B9A
3BA8 06 FF       ASL $FF
3BAA 90 0E       BCC $3BBA
3BAC BD A4 3C    LDA $3CA4,X
3BAF 20 4B 3C    JSR $3C4B
3BB2 BD AA 3C    LDA $3CAA,X
3BB5 F0 03       BEQ $3BBA
3BB7 20 4B 3C    JSR $3C4B
3BBA CA          DEX
3BBB D0 D5       BNE $3B92
3BBD 60          RTS
3BBE 20 D3 3B    JSR $3BD3
3BC1 AA          TAX
3BC2 E8          INX
3BC3 D0 01       BNE $3BC6
3BC5 C8          INY
3BC6 98          TYA
3BC7 20 CB 3B    JSR $3BCB
3BCA 8A          TXA
3BCB 86 B4       STX $B4
3BCD 20 75 E7    JSR $E775
3BD0 A6 B4       LDX $B4
3BD2 60          RTS
3BD3 A4 FC       LDY $FC
3BD5 AA          TAX
3BD6 10 01       BPL $3BD9
3BD8 88          DEY
3BD9 65 FB       ADC $FB
3BDB 90 01       BCC $3BDE
3BDD C8          INY
3BDE 60          RTS
3BDF A8          TAY
3BE0 4A          LSR
3BE1 90 0B       BCC $3BEE
3BE3 4A          LSR
3BE4 B0 17       BCS $3BFD
3BE6 C9 22       CMP #$22
3BE8 F0 13       BEQ $3BFD
3BEA 29 07       AND #$07
3BEC 09 80       ORA #$80
3BEE 4A          LSR
3BEF AA          TAX
3BF0 BD 53 3C    LDA $3C53,X
3BF3 B0 04       BCS $3BF9
3BF5 4A          LSR
3BF6 4A          LSR
3BF7 4A          LSR
3BF8 4A          LSR
3BF9 29 0F       AND #$0F
3BFB D0 04       BNE $3C01
3BFD A0 80       LDY #$80
3BFF A9 00       LDA #$00
3C01 AA          TAX
3C02 BD 97 3C    LDA $3C97,X
3C05 85 FF       STA $FF
3C07 29 03       AND #$03
3C09 85 B6       STA $B6
3C0B 98          TYA
3C0C 29 8F       AND #$8F
3C0E AA          TAX
3C0F 98          TYA

3C10 A0 03       LDY #$03
3C12 E0 8A       CPX #$8A
3C14 F0 0B       BEQ $3C21
3C16 4A          LSR
3C17 90 08       BCC $3C21
3C19 4A          LSR
3C1A 4A          LSR
3C1B 09 20       ORA #$20
```

```
3C1D 88          DEY
3C1E D0 FA       BNE $3C1A
3C20 C8          INY
3C21 88          DEY
3C22 D0 F2       BNE $3C16
3C24 60          RTS
3C25 A8          TAY
3C26 B9 B1 3C    LDA $3CB1,Y
3C29 8D 0B 02    STA $020B
3C2C B9 F1 3C    LDA $3CF1,Y
3C2F 8D 0C 02    STA $020C
3C32 A9 00       LDA #$00
3C34 A0 05       LDY #$05
3C36 0E 0C 02    ASL $020C
3C39 2E 0B 02    ROL $020B
3C3C 2A          ROL
3C3D 88          DEY
3C3E D0 F6       BNE $3C36
3C40 69 3F       ADC #$3F
3C42 20 D2 FF    JSR $FFD2
3C45 CA          DEX
3C46 D0 EA       BNE $3C32
3C48 4C CD FD    JMP $FDCD
3C4B C9 24       CMP #$24
3C4D F0 03       BEQ $3C52
3C4F 20 D2 FF    JSR $FFD2
3C52 60          RTS
```

```
M 3C53,3D38
3C53 40 02 45 03 D0 08 40 09 30 22 45 33 D0 08 40 09
3C63 40 02 45 33 D0 08 40 09 40 02 45 B3 D0 08 40 09
3C73 00 22 44 33 D0 8C 44 00 11 22 44 33 D0 8C 44 9A
3C83 10 22 44 33 D0 08 40 09 10 22 44 33 D0 08 40 09
3C93 62 13 78 A9 00 21 81 82 00 00 59 4D 91 92 86 4A
3CA3 85 9D 2C 29 2C 23 28 24 59 00 58 24 24 00 1C 8A
3CB3 1C 23 5D 8B 1B A1 9D 8A 1D 23 9D 8B 1D A1 00 29
3CC3 19 AE 69 A8 19 23 24 53 1B 23 24 53 19 A1 00 1A
3CD3 5B 5B A5 69 24 24 AE AE A8 AD 29 00 7C 00 15 9C
3CE3 6D 9C A5 69 29 53 84 13 34 11 A5 69 23 A0 D8 62
3CF3 5A 48 26 62 94 88 54 44 C8 54 68 44 E8 94 00 B4
3D03 08 84 74 B4 28 6E 74 F4 CC 4A 72 F2 A4 8A 00 AA
3D13 A2 A2 74 74 74 72 44 68 B2 32 B2 00 22 00 1A 1A
3D23 26 26 72 72 88 C8 C4 CA 26 48 44 44 A2 C8 04 22
3D33 10 20 2D 2F 33 AA AA AA AA AA AA AA AA AA AA AA
```

Diverse Splitter

Korrekturen und Hinweise zu früheren Artikeln. Der Herausgeber dankt den aufmerksamen Lesern. Zunächst Herr Wolfgang Masarie aus 8346 Simbach:

Verlegte FIND-Routine des AIM in Heft 35, S.66: Unter Adresse F866 müßte es wohl BNE $F84E heißen, als Branch Always.

Mini-Floppy für AIM 65, Heft 33, S. 6: Im Schaltplan sind Pin A und Pin B von IC5 nicht richtig bezeichnet. Der Kondensatorwert muß 100 pF sein. Ein 7401-Gatter kann MOT nicht steuern (rechts unten). Ich habe eine ähnliche Schaltung aufgebaut und normale Bustreiber ALS33 und ALS34 verwendet, mußte aber den Terminatorchip der Floppy, der 150 Ohm Pullup-Widerstand enthält, mit 330 Ohm austauschen.

Textformatierer ROFF in Heft 34, S. 45: Herr Uwe Metzler aus Berlin schreibt: Nach der Implementierung stellte ich fest, daß bei einer Ausgabe über den User-Vektor das Programm nicht einwandfrei arbeitet, denn der Akku-Inhalt darf nicht zerstört werden. Hier eine kleine Änderung zur Routine PUTC, die jetzt zu einwandfreiem Arbeiten führt:

```
PUTC    JSR PHXY
        PHA
        LDX SKIFLG
        BNE PUTC2
```

```
        CMP #SEMIBL
        BNE PUTC1
        LDA #BLANK
PUTC1   JSR OUTALL
PUTC2   PLA
        JSR PLXY
        RTS
```

ASSMOS 1.0 in Heft 36, S. 12: Beim Label KTMPL wurde erklärt, daß INC jetzt Befehlsklasse 13 ist (hex 0D). Die Speicherstelle wurde jedoch nicht angegeben. Man ändere alt $DEF6 mit $0C zu neu $DED7 zu $0D. So wird auch INC A korrekt ausgeführt.

DISMON in Heft 35: Wenn man bei $EA27 CMP #'L durch CMP#'F ersetzt, entfällt das lästige Scrollen im Video-Betrieb mit CRT2 beim Load des Editors von der Floppy. Die beiden letzten Hinweise gab Herr Harald Grunewald in Stade.

St.Dir. Peter Rix, 2350 Neumünster

Schnelle BCD-Multiplikation

Multiplikationsprogramme arbeiten zumeist mit vielen zeitaufwendigen Verschiebeoperationen. Hier spielt sicher die Tradition des bitweisen Schiebens bei binären Multiplikationen mit. - BCD-Zahlen werden in 4 Bit dargestellt. Wenn man nun den bei der CPU 6502 möglichen Dezimalmodus benutzt, muß man nach jedem Schritt der mehrfachen Addition (vom Multiplikator her bestimmt) jeweils um 4 Bitstellen schieben. - Das nachstehende Programm kommt mit einer einmaligen Verschiebung des Produktes aus (Label PMAL10). Die eigentliche Multiplikation wird von der Routine MPLY ausgeführt, die einmal für jedes Nibble (Dezimalziffer des Multiplikators) aufgerufen wird, und zwar zunächst für die jeweils 'linken' im Byte, dann ab LOWER für die 'rechten', die mit AND#$F präpariert wurden. – Die Multiplikation wird vorbereitet, indem ab MAL für den Multiplikanden durch Addition auf sich selbst sofort das 2- und 4-fache gebildet wird. Im Unterprogramm MPL wird das aktuelle Nibble des Multiplikators mit LSR A bitweise nach rechts verschoben. Jenachdem, was im Carry landet, muß man nacheinander entweder 1-, 2- oder einmal oder zweimal den 4-fachen Multiplikanden mit JSR ADD zum Produkt aufaddieren. – Dieser Algorrithmus dürfte der bisher schnellste sein. Er kann für beliebig breite Datenfelder benutzt werden. Ein Echo ist erwünscht, auch hinsichtlich eines ähnlichen Divisions-Algorithmus.

```
SCHNELLE BCD-MULTIPLIKATION MIT M*M=2M STELLEN
FUER GEPACKTE BCD-ZAHLEN
MULTIPLIKAND UND MULTIPLIKATOR WERDEN NICHT VERAENDERT
DATEINAME: BCDM8                P. RIX, NMS, 31.07.1983
-------------------------------------------------------
0000 N                  =5              ;BYTE-ANZAHL DER FAKTOREN
0000 M                  =N+N            ;STELLENZAHL DER FAKTOREN
0000                    *=8
0008 MPD4               *=*+N+1         ;4-FACHER MULTIPLIKAND
000E MPD2               *=*+N+1         ;2-FACHER MULTIPLIKAND
0014 MPKD               *=*+N+1         ;MULTIPLIKAND
001A MPKR               *=*+N           ;MULTIPLIKATOR
001F PROD               *=*+M           ;PRODUKT
0029 CFLAG              =PROD           ;CARRY-SPEICHER
0029
0029                    *=$200
0200 START    F8        SED
0201          A900      LDA #0          ;PRODUKTFELD LOESCHEN
0203          A209      LDX #M-1
0205 ST1      951F      STA PROD,X
0207          CA        DEX
0208          10FB      BPL ST1         ;NAECHSTES BYTE
020A          8514      STA MPKD        ;1.MPKD-BYTE MUSS NULL SEIN
```

```
020C MPDMAL   A205     LDX #N            ;2- UND 4-FACHEN MULTIPLIKANDEN
020E          18       CLC               ;BERECHNEN
020F MAL      B514     LDA MPKD,X
0211          7514     ADC MPKD,X
0213          950E     STA MPD2,X        ;2-FACHER MULTIPLIKAND
0215          261F     ROL CFLAG         ;CARRY-WECHSEL
0217          750E     ADC MPD2,X
0219          9508     STA MPD4,X        ;4-FACHER MULTIPLIKAND
021B          661F     ROR CFLAG         ;CARRY-WECHSEL
021D          CA       DEX
021E          10EF     BPL MAL           ;NAECHSTES BYTE
0220
0220 UPPER    A204     LDX #N-1          ;OBERE BCD-ZIFFER (BIT 7..4)
0222 UP1      B51A     LDA MPKR,X        ;IN DEN MULTIPLIKATOR-BYTES
0224          4A       LSR A             ;AUSBLENDEN
0225          4A       LSR A             ;UND IN DIE
0226          4A       LSR A             ;UNTERE ZIFFERNPOSITION (BIT 3..0)
0227          4A       LSR A             ;VERSCHIEBEN
0228          F003     BEQ NXTUP         ;FALLS NULL, NAECHSTE ZIFFER
022A          205902   JSR MPLY          ;ZUM MULTIPLIZIEREN
022D NXTUP    CA       DEX
022E          10F2     BPL UP1           ;NAECHSTE ZIFFER
0230
0230 PMAL10   A204     LDX #4            ;PRODUKT MAL 10
0232 PM1      0628     ASL PROD+M-1
0234          2627     ROL PROD+M-2
0236          2626     ROL PROD+M-3
0238          2625     ROL PROD+M-4
023A          2624     ROL PROD+M-5
023C          2623     ROL PROD+M-6
023E          2622     ROL PROD+M-7
0240          2621     ROL PROD+M-8
0242          2620     ROL PROD+M-9
0244          261F     ROL PROD+M-10
0246          ;        """ """"+M-**     ; ** EINFUEGEN 11...M FUER M>10
0246          CA       DEX
0247          D0E9     BNE PM1           ;4-MAL SCHIEBEN
0249 LOWER    A204     LDX #N-1          ;UNTERE BCD-ZIFFER (BIT 3..0)
024B LO1      B51A     LDA MPKR,X        ;IN DEN MULTIPLIKATOR-BYTES
024D          290F     AND #$F           ;AUSBLENDEN
024F          F003     BEQ NXTLO         ;FALLS NULL, NAECHSTE ZIFFER
0251          205902   JSR MPLY          ;ZUM MULTIPLIZIEREN
0254 NXTLO    CA       DEX
0255          10F4     BPL LO1           ;NAECHSTE ZIFFER
0257          D8       CLD
0258 RETN     60       RTS               ;PRODUKT IST FERTIG, ENDE!
-------------------------------------------------------------
0259          ;UNTERPROGRAMM MULTIPLIZIEREN
0259 MPLY     4A       LSR A             ;MULTIPLIKATOR-ZIFFER
025A          9007     BCC MP1           ;MIT MULTIPLIKAND MALNEHMEN
025C          A00C     LDY #MPKD-MPD4    ;UND STELLENRICHTIG
025E          48       PHA               ;ZUM PRODUKT ADDIEREN
025F          207702   JSR ADD           ;MPKD 1-MAL ADDIEREN
0262          68       PLA
0263 MP1      4A       LSR A
0264          9007     BCC MP2
0266          A006     LDY #MPKD-MPD2
0268          48       PHA
0269          207702   JSR ADD           ;MPD2 1-MAL ADDIEREN
```

```
026C           68       PLA
026D MP2       A000     LDY #0
026F           4A       LSR A
0270           B005     BCS ADD            ;MPD4 1-MAL ADDIEREN
0272           F03D     BEQ RET            ;MPD4 NICHT ADDIEREN
0274           207702   JSR ADD            ;MPD4 2-MAL ADDIEREN
0277
0277 ADD       18       CLC                ;STELLENRICHTIGE ADDITION
0278           B524     LDA PROD+N,X       ;DER MULTIPLIKANDEN-VIELFACHEN
027A           790D00   ADC MPD4+N,Y       ;ZUM PRODUKT
027D           9524     STA PROD+N,X
027F           B523     LDA PROD+N-1,X
0281           790C00   ADC MPD4+N-1,Y
0284           9523     STA PROD+N-1,X
0286           B522     LDA PROD+N-2,X
0288           790B00   ADC MPD4+N-2,Y
028B           9522     STA PROD+N-2,X
028D           B521     LDA PROD+N-3,X
028F           790A00   ADC MPD4+N-3,Y
0292           9521     STA PROD+N-3,X
0294           B520     LDA PROD+N-4,X
0296           790900   ADC MPD4+N-4,Y
0299           9520     STA PROD+N-4,X
029B           B51F     LDA PROD+N-5,X
029D           790800   ADC MPD4+N-5,Y
02A0           951F     STA PROD+N-5,X
02A2           ;        """ """"+N-*,"     ; * EINFUEGEN 6...N FUER N>5
02A2           900D     BCC RET            ;FERTIG BEI CARRY-CLEAR
02A4           8A       TXA                ;SONST
02A5           48       PHA                ;STELLENINDEX RETTEN
02A6 AD1       CA       DEX
02A7           B51F     LDA PROD,X
02A9           6900     ADC #0             ;UEBERTRAG ADDIEREN
02AB           951F     STA PROD,X
02AD           B0F7     BCS AD1            ;WEITER, BIS UEBERTRAG NULL IST
02AF           68       PLA
02B0           AA       TAX                ;STELLENINDEX RESTAURIEREN
02B1 RET       60       RTS
02B2                    .END BCDM8
02B2             ERRORS=  0000
```

□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

Kleinanzeigen

Rockwell 65 C 02 Programming Manual DM 25,-, AIM 65: Satz Originaldokum. mit vorst. Buch DM 48,-, Text- und Grafikdrucker CP 80 (neuwertig) DM 800,-, Rockwell 65 C 02 CPU für Aufrüstung DM 42,-, MC 65 - Komplettbausatz (64 kByte RAM) DM 540,-
Regge Elektronik, 2800 Bremen 1, Fesenfeld 57, Tel.: 0421 - 71 114

Zu verkaufen: AIM 65 mit Video/Grafik Platine, 38 K RAM, Video 16 Zeilen á 32 Zch, Grafik max. 256x192 Pixels, D/A, A/D, Ein- u. Ausgänge prov. für Light Pen. Synertec Floppy Controller und 2 Tandon 5 1/4" Floppy in Geh. VB DM 2200. K. Riesen, Dörflistr. 1, CH-8942 Oberrieden.

AKTUELLE
MIKRO-
COMPUTER
LITERATUR
VON
te-wi
Weitere Bücher
Einführung in die Mikrocomputer-Techni
DM 66,–
MC-Bücher:
Apple II Anwenderhandbuch DM 5
Apple II PASCAL DM 59,–
Apple Maschinensprache DM 4
CBM Computerhandbuch
DM 59,–
C64 Computer Handbuch
(2./3. Q. 84) DM 56,–
Mein ATARI Computer
DM 59,–
IBM-PC Handbuch
(2. Q. 84) DM 59,–
CP/M und Word
DM 29,80
VisiCalc, mit
Diskette
DM 79,–
te-wi
te-wi Verlag GmbH
Theo-Prosel-Weg 1
8000 München 40

Assembler

R65C02-Assembler

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

6805/68705 Cross-Assembler

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

6805/68705-Assembler unter FORTH

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

Mathe-ROM für 6502

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

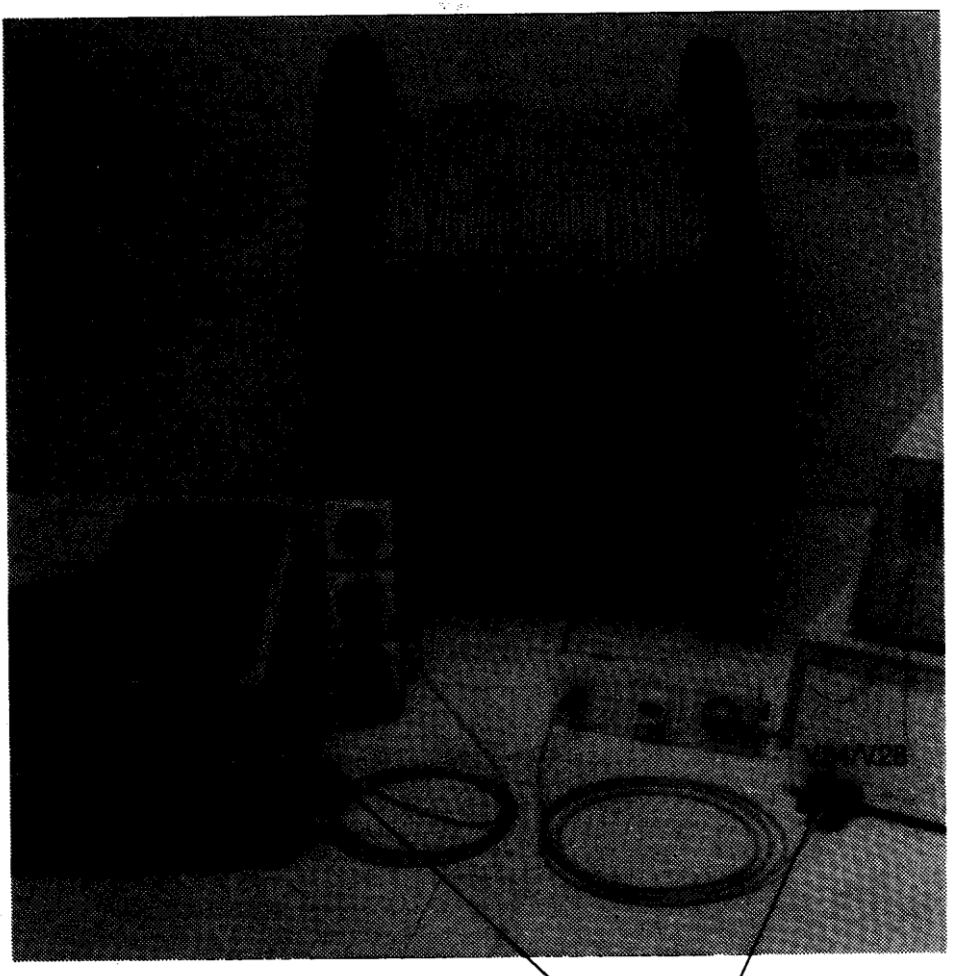
Texterfassung und Lichtsatz

Für die Texterfassung von Büchern, Dokumentationen, Dissertationen etc. mit Fließtext nach klarem Manuskript, auch Sonderzeichen, hervorgehobenen Zwischenüberschriften usw. stehen auftragsweise folgende Möglichkeiten zur Verfügung:

- Sofortige Erfassung und reproreife hier korrigierte Niederschrift mit IBM-Composer (wie in dieser Zeitschrift)
- Texterfassung auf Commodore 8250-Diskette mit späterer Korrekturmöglichkeit, Probeausdruck und anschließendem Lichtsatz oder formatiertem Schreibmaschinensatz
- Regieführung bis zum Druck, auch mit mehrfarbigem Foto auf dem Buchumschlag
- Gestaltung mit Graphikerin, eingabesichere Mitarbeiter

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

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



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

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

DBGM: G 8331 081.9
G 8336 080.8

INGENIEURBÜRO
STECKER

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

DUO Plott Interface

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

Paralleles IEEE 488 - Interface, genormter IEEE-Stecker und Kabel
kompletter Zeichensatz des CBM-Computers
zwei Geräteadressen für Groß-/Grafikmodus und Textmodus
alle Funktionen der Drucker bleiben erhalten, Floppy-Kompatibilität
Deutsche Umlaute, ß und Paragraph
Problemloser Einbau, inklusive deutsches EPSON-Handbuch
Komplettpreis einschl. Kabel, CBM-Grafiksatz und Handbuch incl. MWSt
für EPSON DM 298,--
für ITOH DM 298,--

Umrüstsatz MX80 F/T auf MX80 Typ 3

Aus Ihrem EPSON MX-80 F/T wird der MX-80 Typ 3
Einzelnadelansteuerung, erweiterter Befehlssatz, Elite-Schrift
Preis inkl. MWSt DM 98,--
Komplettpreis Interface, Kabel, Handbuch und Umrüstsatz inkl. MWSt DM 450,--

Für COMMODORE

Deutsche Tastatur mit Original-Tastensätzen (keine Aufkleber)
inkl. deutschem Zeichengenerator
für CBM 8032 DM 98,--
für CBM 8032 und 8096 DM 98,--
nur 8096, ohne Tasten DM 98,--

Speichererweiterung auf 96 K

LOS-kompatibel, inkl. MWSt DM 798,--

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

Drucker:

RX 80 DM 998,-- + MWSt

ITOH 8510 DM 1495,-- + MWSt

EPSON-Druck-Computer FX 80 DM 1295,-- + MWSt

Stellberg Computer-Systeme

COMMODORE EPSON C. ITOH SOFTWARE INTERFACE

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

MICRO MAG

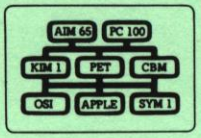
HERAUSGEBER/EDITOR:
DIPL.-VOLKSWIRT ROLAND LÖHR
HANSDORFER STRASSE 4
D-2070 AHRENSBURG
☎ (04102) 5 58 16

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 der Zeitschrift. Es erfolgt keine Neuauflage.

Lieferbar: 'Das Buch 7-13 des 65xx MICRO MAG' zu DM 42,-- sowie die Hefte 14-36 zu DM 7,80/St. (ab 10 St. in einer Sendung: DM 6,--/St.).

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

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

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

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