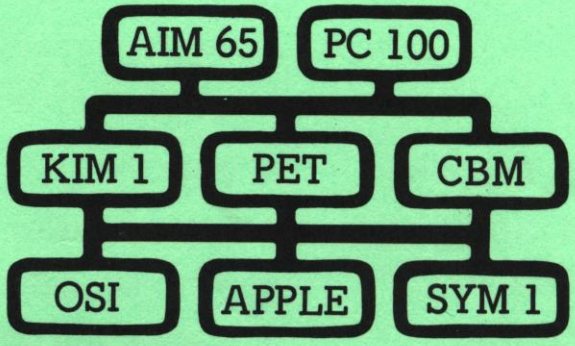
65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

DM 8,50 Nr. 24 April 1982



Inhaltsverzeichnis

FORTH (1)

1. Einleitung

Aus der Leserumfrage im Februar/März geht ein so großes Interesse an der Sprache FORTH hervor, daß eine umfassende Darstellung für das Programmieren jetzt dringend nötig ist. In amerikanischen Zeitschriften finden sich zwar verstreut Beiträge für diese Sprache, aber es sieht so aus, als wenn das einführende Lehrbuch noch fehlte. Dem Erwerber eines FORTH für den AIM 65 steht ein etwa 300seitiges Buch in englischer Sprache zur Verfügung (Forth user's manual), das die notwendigen Hantierungen am Computer erklärt und viele kurze Beispiele für den Gebrauch der Befehle gibt, das aber kein Lehrbuch für Problemlösungen sein will. Die Dokumentationen der verschiedenen für CBM-Rechner erhältlichen FORTH-Implementierungen sind hier nicht bekannt, es dürfte sich aber auch dort kaum um systematische Einführungen handeln.

Wir beginnen daher in diesem Heft mit einer Fortsetzungsserie, die ursprünglich als ein Buch geplant war, dessen Abschluß und dessen Herstellung noch länger dauern würde. Seine Information sollte möglichst schnell zur Verfügung stehen. Es ist dabei das Ziel, dem Leser solide Grundlagen für das Programmieren mit FORTH zu vermitteln. Der zunächst abgedruckten Überschau in diesem Heft folgen schnell einfache Beispiele zum Kennenlernen der Sprache. Es folgt eine systematische Darstellung des Befehlsvorrates. mit zahlreichen Programmierbeispielen. Besondere Abschnitte sind dann der Durchdringung der tieferen Sprachstruktur und der Interfaceprogrammierung gewidmet.

Eine Sprache sollte am besten mit Übungen am Computer erlernt werden. Wir gehen vorwiegend vom FORTH V1.3 für den AIM 65 aus, wie es von Rockwell in Form von zwei Festwertspeichern angeboten wird. Es ist eine sehr genaue Implementierung des in den USA entwickelten FIG FORTH der FORTH Interest Group. Die Darstellungen sind damit für alle Betreiber eines solchermaßen basierenden FORTH geeignet.

2. Eine erste Übersicht

2.1 Wichtige Merkmale des FORTH

Bei der Auswahl einer Programmiersprache stellt man sich die Frage, welche Hilfsmittel sie für Problemlösungen bietet. Dabei ist folgende 'Umgebung' zu betrachten:

Erlernbarkeit der Sprache,
Übertragbarkeit der Lösung von einem Rechner zum anderen,
die von der Sprache unterstützten Datentypen und ihre Rechengenauigkeit,
Befehlsausführung in zeitkritischen Anwendungen, Anschluß von Maschinensprache,
Zusammenarbeit mit externen Speichermedien für Daten,
das Interface Mensch-Maschine bei der Benutzung der Sprache,
Erweiterung des Befehlssatzes durch den Anwender und
Sicherheit und Produktivität der Programmierung, Fehlererkennung.

Bei so vielen Gesichtspunkten kann die Antwort hier nicht in einem allgemeinen Absatz gegeben werden. Der Leser wird sich nach dem Studium der Abschnitte die Antworten z.T. selber geben müssen, er kennt sein eigenes Zutrauen und seine geplanten Anwendungen. Die nachstehenden Aussagen sind daher als vorläufig zu betrachten;

Erlernbarkeit: Viele Befehlsworte bestehen nur aus einem (abstrakten) Zeichen. Die Anlehnung an Begriffe der englischen Sprache ist geringer als in anderen Programmiersprachen,

wie BASIC, PASCAL, FORTRAN usw.. FORTH erlaubt jedoch die beliebige Umbenennung von Befehlsworten durch den Anwender. Statt z.B. 'EMIT' könnte er neu formulieren:

: ASCII-AUS EMIT ;

Mit dem neuen Befehlswort ASCII-AUS erreicht er also das Gleiche wie mit EMIT, nämlich die Ausgabe eines auf dem Datenstack abgelegten Zeichens gemäß ASCII-Codetabelle.

Es ist also niemand gezwungen, den Befehlsvorrat in seiner ursprünglichen gelegentlich abstrakten Form zu benutzen. Der Anwender könnte also durchaus auf Deutsch oder eine andere Sprache umformulierte Befehle verwenden. Hier liegt Anwenderfreundlichkeit, allerdings um den Preis, daß der Gedankenaustausch mit anderen FORTH-Betreibern erschwert würde.

Unter FORTH stehen dem Anwender 200 und mehr Befehlsworte zur Verfügung. Das ist mehr, als ein programmierbarer Taschenrechner oder Assembler bzw. BASIC eines Mikrocomputers zu lernen fordert. Die Fülle der möglichen Dienstleistungen einer Sprache birgt immer die Gefahr der Verunsicherung des Benutzers in sich. - Die am häufigsten benutzten Befehlsworte stellen aber nur eine kleine überschaubare Untermenge des Befehlsvorrates dar. Es wird also wesentlich darauf ankommen, hier die seltener benutzten Möglichkeiten 'aufschlagbar' zu machen. Die Frage der Erlernbarkeit wird damit reduziert.

Die Vorteile einer höheren Programmiersprache sollen u.a. darin liegen, daß sie nicht nur eine bequeme Programmierung/Problemlösung ermöglicht, sondern es daneben gestattet, den Quelltext des Programmes auch auf einer anderen für diese Sprache vorbereiteten Maschine ausführen zu können **(Übertragbarkeit/Portabilität)**. Wie von Radio Eriwan ist zu antworten: 'Im Prinzip ja!'. Jede Sprache bildet ihre Dialekte heraus, wir haben es an BASIC. PASCAL und anderen gesehen, auch bei FORTH. Der mitgelieferte Befehls-Grundvorrat ist unterschiedlich. Aber: FORTH besitzt die Kraft, Befehlsworte nachzuimplementieren. Man muß also nicht weitere Befehle hinzukaufen. Wie es scheint, ist das hier behandelte FIG FORTH der Hauptdialekt. Wer also 'EMIT' zu 'ASCII-AUS' umformuliert, spricht seinen eigenen Dialekt, was (wenigstens gegenüber der Öffentlichkeit) zu vermeiden ist.

Zu Datentypen und Rechengenauigkeit: Das hier besprochene AIM-FORTH (FIG FORTH) unterstützt von Haus aus Rechenkonstante und -variable mit aufrufbarem Namen und einer internen Genauigkeit von 16 Bit, entsprechend dem üblichen Integer-Wertebereich. Neben den vier Grundrechenarten stehen keine weiteren höheren mathematischen Funktionen zur Verfügung. Es kann mit Zahlen doppelter Genauigkeit gerechnet werden (32 Bit, Wertebereich also plus/minus 2,14 Milliarden). Nicht implementiert ist die Verarbeitung von Gleitkommazahlen, die bei Bedarf vom Anwender nachvollzogen werden müßte.

Unterstützt ist die Ein- und Ausgabe von ASCII-Zeichen und Zeichenketten. Die Einrichtung von so nützlichen Datentypen wie Stringvariablen muß vom Anwender besorgt werden, ebenso die nützlichen Stringfunktionen z.B. des BASIC. Hierfür gibt es Einrichtungsvorschläge. - FORTH enthält einen internen Compiler (wir werden darauf noch eingehen). Mit ihm ist es möglich, beliebige Datentypen zu erklären, also nicht nur Gleitkommazahlen und Stringvariable. Damit ist bereits die **Erweiterungsfähigkeit** angesprochen. Sie bezieht sich nicht nur auf die Umbenennung vorhandener Funktionen, sondern auf Datentypen und auszuführende beliebige Funktionen oder Vokabulare, die der Anwender definiert und mit einem Befehlswort aufruft. - In der anwenderfreundlichen Erweiterung liegt der besondere Reiz des FORTH. Die neuen Vokabulare könnten eine Steuersprache oder ein Cross-Assembler für ein anderes System sein. Hierzu wird der Leser nützliche Anregungen finden.

Zu den zeitkritischen Anwendungen und zum Anschluß schnell ausführender Routinen in Maschinensprache:Die Benutzung jeder höheren Programmiersprache bedingt einen gewissen Verwaltungsüberbau zur Kontrolle der Abarbeitung. Auf die Unterschiede zwischen Compilersprachen und Interpretersprachen werden wir noch eingehen. Die notwenige Verwaltung mag sich in der Kompilierungsphase bemerkbar machen, wenn der Sprachkompiler unmittelbar ausführende (binär codierte) Maschinenprogramme erzeugt, oder aber in der Ausführungsphase, wenn ein sprachliches Mittelding von einem interpretierenden Verwaltungsprogramm in den Aufruf von Routinen umgesetzt wird, die in Maschinensprache geschrieben sind. Solche Routinen werden letztlich zur Abarbeitung eines jeden Programmes benötigt. Das Bestreben ist im allgemeinen darauf gerichtet, zur Zeit der Programmausführung (runtime) möglichst wenig Zeitverlust in Kauf zu nehmen. Kompilersprachen sind hier zeitlich überlegen, sie beanspruchen allerdings mehr Speicherplatzaufwand. FORTH ist zum Teil eine Interpretersprache, es gibt also ein internes Verwaltungsprogramm, das den Ablauf steuert. Der Einsprung in reine Maschinenprogramme, die mit Namen aufgerufen werden, ist mit nur ganz geringem Zeitverlust möglich. Hier liegt die Stärke für die maschinennahe und zeitkritische Programmierung.

Die Zusammenarbeit des FORTH mit externen Speichermedien (Cassette, Diskette) ist geregelt, es ist allerdings dem Anwender überlassen, die Datenpuffer und die Einsprungadressen in die benötigten Treiberprogramme systemabhängig einzurichten.

Das Interface Mensch-Maschine ist durch verschiedene Prompts, d.h. Aufforderungen des FORTH und durch Fehlermeldungen besorgt. Wie aber auch in anderen Sprachen sind besondere Anforderungen für die Bedienerführung auszuprogrammieren, z.B. 'Eingabe Datum?'. Es liegt also auch hier in der Sorgfalt des Programmierers, die Anforderungen zu prüfen und Dateneingaben auf ihre Gültigkeit zu überprüfen.

2.2 Interpreter- und Kompilersprachen

Zur Einordnung des FORTH: Als digtitale Maschine kann der Computer nur binäre Information in Form von Befehlen, Adressen und codierten Daten abarbeiten. Jede Computersprache wird letzendlich in der Form von solchen Maschinenbefehlen ausgeführt. Damit setzt jede Sprache Hilfsmittel voraus, die die benötigte binäre Information für die Maschine erzeugen. - Das einfachste Hilfsmittel ist ein Assemblerprogramm, das aus einem eingegebenen Programmquelltext das Maschinenprogramm erzeugt. Der Programmtext muß dabei nach den Regeln und mit dem Befehlsvorrat der Assemblersprache des betreffenden Computers geschrieben sein. Man kann auch etwas vereinfachend sagen: Jedem binären Muster, das die Maschine als Operationscode verarbeiten kann, entspricht ein Befehlswort der Assemblersprache. Es gibt also eine umkehrbar eindeutige Zuordnung zwischen Operationscode und Assemblerbefehl (mit seiner Adressierungsart). Assembler ist damit die einfachste Programmiersprache, sie führt ohne Umwege zu ausführendem Maschinencode.

BASIC, PASCAL, FORTRAN, COBOL usw. sind höhere Programmiersprachen. Für sie gibt es zwei Möglichkeiten, aus den in der jeweiligen Sprache formulierten Anweisungen zum eigentlich ausführenden Maschinencode zu kommen, die Interpretierung und die Kompilierung.

Beim Interpreterprinzip befindet sich der Programmquelltext (z.B. unter BASIC formuliert) zur Zeit der Programmausführung auf der Maschine. (Meistens sind dabei die eigentlichen Befehle der Sprache schon zu 'tokens' von 1 Byte Länge komprimiert.) Ein Interpreterprogramm bringt diesen Quelltext zur Ausführung, indem es in Maschinensprache geschriebene Unterprogramme aufruft und zur Ausführungszeit die notwendige Verwaltungsarbeit besorgt: Verfolgung der Anweisungen und Anweisungszeilen, Fortführen der Pointer und der sich ergebenden Bedingungen, Aufsuchen von Konstanten, Variablen usw.. Auch die Syntax-

prüfung wird im allgemeinen zur Laufzeit durchgeführt. Der Interpreter lernt von Programmlauf zu Programmlauf nichts hinzu, er muß immer wieder die selben Statements ausdeuten und Werte suchen. Das bedingt - relativ zum reinen Assemblerprogramm - eine langsame Ausführung, hat aber den Vorteil, daß Programmabschnitte anwenderfreundlich schnell geändert werden können, weil der Programmtext auf der Maschine ist. - Zusammenfassend fordert das Interpreterprinzip also, daß zur Zeit der Ausführung der Quelltext, der Interpreter und ein 'runtime package' ausführender Maschinenprogramme auf der Maschine anwesend sind.

Die Implementierung reiner Compilersprachen auf Mikrocomputern ist gegenüber der Anwendung des Interpreterprinzips noch selten. Kompilierung heißt letztendlich, daß aus dem Programmquelltext reiner Maschinencode erzeugt wird. Das Quellprogramm kann also erst dann ausgeführt werden, wenn ein Umwandlungsprogramm, der Kompiler, tätig gewesen ist. Kompilieren ist also eine Vorverarbeitung. Zur runtime wird der Quelltext nicht mehr auf der Maschine benötigt, auch nicht das Umwandlungsprogramm. Kompilierte Programme sind speicheraufwendig und im allgemeinen auch nicht so optimert, wie man es unter Assemblerprogrammierung erreichen kann.

Volle Kompilierung sprengt sehr schnell die Speichergrenzen eines kleinen Systems. Man hat daher gemischte Formen gebildet. Typisch sind hier die meisten Verwirklichungen des PASCAL: Ein 'Kompiler' erzeugt hier aus dem PASCAL-Quelltext einen künstlichen Zwischencode. Dieser kann zur Programmausführungszeit von einem P-Code-Interpreter zeitgünstig auf einem 'runtime package' zur Ausführung gebracht werden.

2.3. Einordnung des FORTH

Wo nun in dieser Landschaft der Sprachverwirklichungen steht FORTH? Es zunächst einmal eine 'gefädelte' Interpretersprache (TIL, Threaded Interpretive Language). Als solche hat sie einen Interpreter, der den Programmzeiger weiterverfolgt, die Verwaltung besorgt und die letztlich ausführenden Maschinenprogramme aufruft. FORTH kommt dabei mit einem kleinen Kern (nucleus) von Maschinencode aus, den sog. primitives oder primaries. Das Interessante ist nun, daß die weiteren höheren Befehlswörter, die secondaries durch die Aufreihung von primaries und schon vorhandenen secondaries (die also 'stammesgeschichtlich' älter sind) gebildet werden. Für die Bildung neuer Befehlsworte ist im FORTH ein Kompiler enthalten, der aus eingegebenen Quelltexten neue Befehlsworte zu bilden gestattet. DerQuelltext wird nach der Kompilierung nicht mehr gebraucht. Das neue Befehlswort enthält neben einem Kopfteil mit dem Namen und Verwaltungsinformation nur noch die Wortadressen der benutzten älteren primaries, secondaries und ggfs. Variablen. Der entstehende Code ist sehr kompakt und führt zu einer im allgemeinen schnellen Ausführungszeit. Die Kompilierfähigkeit betrifft nicht nur neue Befehle, sondern auch Konstante, Variable, Assemblerroutinen in Maschinensprache sowie nach dem Willen des Programmierers neue Kompiler für Datentypen und neue Vokabulare.

FORTH ist also auch eine Kompilersprache, allerdings nicht in der Form, daß sie reinen Maschinencode erzeugt. Amerikanische Quellen nennen FORTH wegen der beschränkten Zahl der schon implementierten höheren Funktionen gelegentlich eine 'Rumpfsprache' (framework language) oder auch als intermediäre Sprache. Unter dem Blickwinkel mancher Anwender mag FORTH ggfs. als ein sehr leistungsfähig erweitertes Betriebssystem angesehen werden. Damit ist keine Abwertung verbunden, denn die Meriten des FORTH liegen zweifellos in seiner Anpassungsfähigkeit.

2.4 Geschichtliches

FORTH ist die Abkürzung von FOURTH (Fourth Generation Computer Language). Die Sprache wurde nach vielen Vorversuchen seit Anfang der siebziger Jahre von Charles H. Moore geschaffen (heute Direktor der FORTH INC. Hermosa Beach, CA) und erstmals 1971 für die Datenerfassung an einem Radioteleskop in den USA eingesetzt. Seither haben sich zahlreiche Firmen und Organisationen um die Weiterentwicklung und Verbreitung des FORTH bemüht. Insbesondere ist die FORTH Interest Group (FIG), P.O. Box 1105 in San Carlos, CA 94070 zu nennen, die auch regelmäßig Mitteilungen veröffentlicht.

2.5 Die Grammatik des FORTH

Die Syntax des FORTH ist denkbar einfach:

* Ein Befehlswort kann einen Namen haben, der aus bis zu 32 beliebigen Zeichen besteht. Zeichen bedeutet hier: Buchstabe, Ziffer oder Sonderzeichen - mit Ausnahme des Leerzeichens (hex 20)
* Ein oder beliebig viele Leerzeichen (hex 20) werden benutzt, um aufeinanderfolgende Befehlsworte oder Operatoren gegeneinander abzugrenzen (im Gegensatz zu BASIC wo man komprimiert schreiben kann: 10FORI=!TO100:)
* Ein CR (hex 13, Carriage Return) ist ein gleichwertiger Befehlsbegrenzer Im Tischrechner-Modus löst ein 'CR' die Abarbeitung der eingegebenen Programmzeile aus.

2.6 Prompts

Im Tischrechner-Modus antwortet das System nach Abarbeitung einer Programmzeile entweder mit a) 'OK', wenn alles gut ging, b) mit dem als nicht bekannt beanstandetem Befehlswort und einem Fragezeichen (z.B. LOG ?) oder aber c) mit einer Fehlermeldung.

2.7 Umgekehrte Polnische Notation (UPN)

FORTH ist eine stack-orientierte Sprache. Wer schon mit einem Taschenrechner von Hewlett Packard gerechnet hat, kennt die Abfolge: 1. Operand ENTER, 2. Operand ENTER, Operationstaste. Die Operation wirkt an den beiden zuvor auf den Datenstack beförderten Operanden, beseitigt sie dort und hinterläßt das Ergebnis der Operation zuoberst auf dem Datenstack, von wo eine Weiterverarbeitung oder Ausgabe möglich ist.

Die Abfolge unter FORTH sieht nicht anders aus. Die Addition zweier Zahlen bedingt die analoge Abfolge der Eingabe:

4 5 + CR (Carriage Return)

Man bemerke, daß die Operanden 4 und 5 durch Zwischenräume voneinander getrennt sind und daß auch das Operationszeichen '+' entsprechend abgetrennt ist.

Da jeder bessere Computer eine Tippfehlerkorrektur in der offenen Befehlszeile zulassen sollte, erfolgt die Befehlsausführung erst nach Zeilenabschluß mit der 'CR'-Taste. Wenn sich dann das System mit (einem evtl. unterstrichenen) 'OK' zurückmeldet, befindet sich das Ergebnis zuoberst auf dem Stack, wo eine Weiterverwertung oder von wo aus eine Ausgabe erfolgen kann. Also:

4 5 + (CR) OK

Diese Eingabefolge wird abgekürzt auch als UPN (Umgekehrte Polnische Notation), RPN (Reverse Polish Notation) oder als Postfix-Notierung bezeichnet. Vom Standpunkt der Maschine her erlaubt sie eine besonders einfache Abarbeitung, für den Programmierer ist sie etwas gewöhnungsbedürftig, denn normalerweise denkt und schreibt er:

4 + 5 =

oder unter BASIC notiert er in die Variable A z.B.:

A = 4 + 5

Wir benutzten soeben die Begriffe 'Datenstack' und 'zuoberst'. Hierzu ist etwas Vertiefung notwendig:

2.8 Der Hardware-Stack

Wer maschinensprachlich programmiert hat, weiß, daß es im System einen Hardware-Stack gibt, einen Speicherbereich, in den die CPU bei Unterprogrammaufrufen und Interrupts die Programm-Rückkehradresse, den Maschinenstatus und ggfs. weitere Maschinenregister rettet. Diese Einrichtung bleibt bei der Benutzung einer höheren Programmiersprache unberührt, weil sie das elektrische Verhalten der Hardware betrifft. Bei der CPU 6502 befindet sich der Hardware-Stack immer in der Speicherseite 1, bei 01FF (hex) beginnend und ggfs. herunterkommend bis 0100. Mit jeder Beschickung des Stacks werden also die in dieser Speicherseite benutzten Adressen kleiner. Die übliche Wortwahl 'ein Byte liegt zuoberst auf dem Stack' ist daher gleichzusetzen mit 'es befindet sich auf der niedrigsten bisher benutzten Stackadresse'.

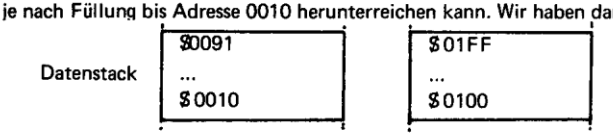
Die Benutzung des Hardware-Stacks zur Übergabe von Parametern (Werten, Zeichen, Adressen) an andere Programmteile ist für den Programmierer eine sehr bequeme Angelegenheit, weil er sich um die Adressierung auf dem Stack nicht kümmern muß. Mit jeder Stack-Beschickung vermindert sich der von der CPU selbst verwaltete Stackpointer, mit jeder Stack-Entnahme erhöht er sich automatisch.

Die Parameterübergabe über den Stack ist eine Teilmenge der möglichen Parameterübergaben. Häufig benutzt werden auch die Parameterübergabe in den Maschinenregistern (Akkumulator/en, Indexregistern/n) sowie die Wertablage an festen Speicheradressen (Briefkastenverfahren). FORTH erwartet demgegenüber, daß der Programmierer die Parameter zuvor auf den Hardware- oder bevorzugt auf den Datenstack gebracht hat. Das 'Interpreterverfahren' liegt damit weitgehend in der Programmierungsphase und in der Geschicklichkeit des Programmierers.

2.9 Der Datenstack

Der Datenstack des AIM-FORTH ist (im Gegensatz zum von der CPU verwalteten Hardwarestack) ein von der Betriebssoftware unterhaltener Stack. Er wurde eingerichtet, um Parameter (Operanden und Zeiger auf Operanden) an definierter Stelle an die ausführenden Routinen übergeben zu können. Er gleicht dem Arbeitsplatz eines einäugigen Beamten, der die hereinkommenden Vorgänge immer zuoberst auf seinem Aktenstapel findet und von oben weg abarbeitet, ohne sich um das Eingangsdatum zu kümmern, wenn man von gelegentlichen Stackmanipulationen einmal absieht.

Es gibt CPUs, die einen besonderen Anwender-Stack schon von der Architektur her haben, so z.B. der 6809 von Motorola, und die daher für stack-orientierte Sprachen gut vorbereitet sind. Wo ein solcher zweiter Stack nicht vorhanden ist, muß man ihn für Sprachen wie FORTH oder PASCAL per Dienstprogramm simulieren. Dem Programmierer kann es eigentlich einerlei sein, an der Benutzung der Sprache ändert sich für ihn dadurch nichts. - Für das FORTH benutzt der AIM einen Datenstack, der in der Zero Page bei hex 0091 beginnt und der je nach Füllung bis Adresse 0010 herunterreichen kann. Wir haben damit:



Es ist der Vorteil der höheren Sprachen, daß sich der Anwender um solche Adressierungen im allgemeinen nicht kümmern muß. Gleichwohl ist erwähnenswert, daß die im Bereich von 0000 bis 00FF ausführenden Befehle der CPU 6502 (Zeropage) nur 2 Byte pro Befehl benötigen und schneller ausführen, als absolut (extended) adressierte. Weiterhin ist zu bemerken, daß beim AIM-FORTH das Indexregister X die Funktion des Daten-Stackpointers übernimmt, eine von der Architektur der CPU her einzig richtige Festlegung und Benutzung. Der Benutzer kann beim Einsprung in eigene maschinensprachliche Routinen weiterhin darauf vertrauen, daß ihm das zweite Indexregister, nämlich Y, zu '0' gesetzt übergeben wird und daß er es verwüstet hinterlassen kann. Wildes Parken wird hier also ausnahmsweise geduldet, nicht jedoch beim X-Register!

2.10 Besonderheiten der Parameterübergabe über den Datenstack

Wer in BASIC programmiert hat,weiß, daß eine Konstante oder Variable dort dadurch angelegt wird, daß man sie an beliebiger Stelle im Programmablauf benutzt, z.B. 10 A=4: B=A+5. Bei den stack-orientierten Sprachen ist es anders: Konstante und Variable müssen ausdrücklich erklärt werden. Erst danach kann man sie benutzen. Die Wertzuweisung erfolgt durch einen besonderen Schreibvorgang, der vom Zuweisungszeichen '=' verschieden ist.

In jedem Fall muß sich der Programmierer vor Augen halten, daß die Reihenfolge der Stackbeschickung und Stack-Entsorgung in seiner eigenen Verantwortung liegt. Das FORTH-System erinnert in keiner Weise, mit welchen Werten/Datentypen die letzten stack-wirksamen Operationen erfolgten und ob mit ihnen irgendwelche Variablennamen verbunden waren.

Die Beobachtung des Datenstacks gehört damit zu den wichtigsten Aufgaben in der Programmierphase. Die Menge der den Stack beschickenden und entlastenden Operationen ist innerhalb eines Bearbeitungszyklus' ausgewogen zu halten.

Wegen der umgekehrten polnischen Notation und wegen des Verschwindens der auf dem Stack abgelegten Parameter in der 'Anonymität' der dort versammelten Bytes ist FORTH gewöhnungsbedürftig. Gleichwohl ist diese Sprache in sich transparent.

2.11 Der Nutzen eines Editors

Programme sind bekanntlich nie auf Anhieb 'fertig', man braucht einen Textspeicher und den ihn verwaltenden und bearbeitenden Editor, um Programmzeilen verändern zu können. Dem Benutzer eines AIM 65 steht ein solches Dienstleistungsprogramm bereits mit der Grundausstattung zur Verfügung. Bei der Auswahl eines FORTH für andere Rechner achte man darauf, daß ihm ein solcher Editor zur Verfügung steht, möglicherweise ist er selbst in FORTH geschrieben.

Einige Literaturhinweise zu FORTH

Folgende Zeitschriften veröffentlichten Hefte mit dem Schwerpunkt-Thema FORTH: BYTE im August 1980, Dr. Dobb's Journal im September 1981, INTERACTIVE (Rockwell) im Januar 1982. Das letztgenannte Heft enthält etwa 30 Verbesserungen zum FORTH-Anwenderhandbuch für den AIM 65.

Im BYTE-Verlag erschien 1981 das Buch 'Threaded Interpretive Languages', Autor Loeliger. Es enthält vor allem Implementierungshinweise für den Prozessor Z80, Aufbau des inneren Interpreters, der primaries und der secondaries.

R.L. #

Michael Zimmermann, 6102 Pfungstadt

BASIC-Disassembler für 6809

Der nachstehende Disassembler wurde geschrieben, um das nicht dokumentierte Betriebsprogramm des Color-Computers (TRS) zu durchleuchten. Da es sich um eine reine BASIC-Maschine handelt, wurde dieses immer nützliche Untersuchungs-Instrument in BASIC geschrieben. Damit ist der Disassembler auf andere Maschinen übertragbar, wenn man von kleinen Eigenheiten des hier benutzten BASIC-Dialektes absieht. Das betrifft vor allem den PCLEAR-Befehl, der entfallen kann, möglicherweise auch den MID$-Befehl. Ebenso ist der HEX$-Befehl nicht als Standard unter BASIC anzusehen. Er ist ggfs. durch ein Unterprogramm auszufüllen.

Auf eine detaillierte Beschreibung des Programmes soll hier verzichtet werden, denn in der nachfolgenden Programm-Liste sind ausreichend Kommentare enthalten, die die genaue Verfolgung des Ablaufes ermöglichen sollten. Hinsichtlich der Befehlsstruktur des 6809 wird auf die Darstellungen in den früheren Heften dieser Zeitschrift verwiesen.

Zum Verständnis sei auf das Folgende hingewiesen: Die Zeilen bis 460 dienen lediglich der Dokumentation. Sie beschreiben hauptsächlich die Variablen. Die Zeile 470 schränkt den Bereich der hier nicht benötigten Graphik-Anwendungen ein. Ab Zeile 490 werden die symbolischen Operationscodes definiert und abgespeichert.

Von den bei anderen BASIC-Disassemblern angewandten Technik mit READ und DATA wurde abgewichen, um Platz zu sparen. - In der Tabelle OC$(15) sind in jeweils 5 Bytes der Typ eines Befehles und dessen symbolischer Code abgelegt. Der High-Nibble (vorderes Halbbyte) des Opcodes bestimmt den Index von OC$, sozusagen die Zeile der Tabelle. Das untere Halbbyte ergibt die Position innerhalb der Zeile, dh. die Tabellenspalte.

Die Befehlsschlüssel nach den Postbytes hex 10 und 11 sind in den Tabellen OZ$ und OE$ abgespeichert. Hier ist es wegen der nur sporadisch über den Wertebereich verteilten Befehle nicht sinnvoll, diese positionsmäßig abzuspeichern. Daher werden hier 7 Bytes lange Tabellenelemente verwendet. Sie enthalten vor Typ und mnemonischem Code noch das Opcode-Byte in hexadezimaler Form.

Auf eine separate Speicherung der Codes für die Long Branches wurde verzichtet, stattdessen werden die entsprechenden Elemente aus dem Array OC$ mitbenutzt. - Vom 6809 unbenutzte Opcodes werden durch die Zeichenfolge @@@ angezeigt. Die Zahlen für die Operationstypen haben folgende Bedeutung:

0 indirekt, 1 Byte lang
1 nicht benutzt
2 direkt, 2 Byte lang
3 immediate, 2 Byte lang
4 relative, 2 Byte lang
5 relative, 3 Byte lang
6 extended, 3 Byte lang
7 indexed, 2-4 Byte lang
8 immediate, 3 Byte lang

Bei den Längenangaben bleiben die Prebytes unberücksichtigt Ein Postbyte wird nur bei der indizierten Adressierung verwendet und folgerichtig auch dort die entsprechende Befehlslänge ermittelt und verarbeitet. Ab Zeile 700 beginnt der eigentliche Programmablauf.

```
10 REM ----- 6 8 0 9   -   D I S A S S E M B L E R
20 REM               I N   B A S I C
30 REM
40 REM
50 REM
60 REM        COPYRIGHT (1982) MICHAEL ZIMMERMANN
70 REM
80 REM --- BESCHREIBUNG DER VARIABLEN
90 REM
100 REM  A = ADRESSE ALS DEZIMALZAHL
110 REM  A$ = ADRESSE ALS HEXA-WERT
120 REM  AH = HIGH-BYTE DER ADRESSE ALS DEZIMALZAHL
130 REM  AH$ = HIGH-BYTE DER ADRESSE ALS HEXA-WERT
140 REM  AI = EINGEGEBENE ADRESSE ALS DEZIMALZAHL
150 REM  AL = LOW-BYTE DER ADRESSE ALS DEZIMALZAHL
160 REM  AL$ = LOW-BYTE DER ADRESSE ALS HEXA-WERT
170 REM  D = AUSGABEEINHEIT 0=BILDSCHIRM, 2=DRUCKER
180 REM  EA = EFFEKTIVE ADRESSE IN DEZIMAL
190 REM  EH = HIGH-BYTE EFFEKTIVE ADRESSE IN DEZIMAL
200 REM  EL = LOW-BYTE EFFEKTIVE ADRESE IN DEZIMAL
210 REM  EL$ = LOW-BYTE EFFEKTIVE ARESSE ALS HEXA-WERT
220 REM  FI = FLAG IM POSTBYTE FUER INDIREKTE ADRESSIERUNG BZW.
230 REM       FLAG FUER SPRUNG ZURUECK BEI 5-BIT-ADRESSE
240 REM  F5 = FLAG FUER 5-BIT-ADRESSE IM POST-BYTE
250 REM  HN = HIGH-NIBBLE DES BEFEHLSBYTES
260 REM  I = EINGEGEBENES ZEICHEN ALS DEZIMALZAHL
270 REM  I$ = EINGEGEBENES ZEICHEN ALS HEXA-WERT
280 REM  IX = ZUERST POSTBYTE, DURCH OPERATIONEN INDEXFORM BZW.
281 REM  4-BIT-ADRESSE
290 REM  LN = LOW-NIBBLE DES BEFEHLSBYTES
300 REM  M$ = BEFEHLSMODE D.H. BEFEHLSTYP
310 REM  O = BEFEHLSBYTE ALS DEZIMALZAHL
320 REM  O$ = BEFEHL IN MNEMONISCHER FORM
330 REM  OC$(15) = TABELLE DER MNEMONISCHEN BEFEHLSCODES
340 REM  OE = BEFEHLSBYTE NACH PREBYTE ALS DEZIMALZAHL
350 REM  OE$ = TABELLE DER MNEMON. BEFEHLSCODES FUER PREBYTE $10

360 REM  OX$ = BEFEHLSBYTE ODER PREBYTE ALS HEXA-WERT
370 REM  OY$ = BEFEHLSBYTE NACH PREBYTE ALS HEXA-WERT
380 REM  OZ$ = TABELLE DER MNEMON. BEFEHLSCODES FUER PREBYTE $10

390 REM  O1 = ERSTES BYTE DES OPERANDEN ALS DEZIMALZAHL
400 REM  O1$ = ERSTES BYTE DES OPERANDEN ALS HEXA-WERT
410 REM  O2 = ZWEITES BYTE DES OPERANDEN ASL DEZIMALZAHL
420 REM  O2$ = ZWEITES BYTE DES OPERANDEN ALS HEXA-WERT
430 REM  PB$ = POSTBYTE ALS HEXA-WERT
440 REM  RR = REGISTERNUMMER, INDEXREGISTER BEI POSTBYTE
450 REM       WEITERHIN REGISTER BEI PUSH UND PULL, EXG UND TFR
460 REM  SA$ = ADRESSE IN SYMBOLISCHER FORM
470 PCLEAR 1
480 REM --- AUFBAU TABELLE INSTRUKTIONENTYP UND SYMBOLISCHER COD
E
490 DIM OC$(15)
500 OC$(0)="2NEG O$$$$O$$$$2COM 2LSR O$$$$2ROR 2ASR 2ASL 2ROL 2D
EC O$$$$2INC 2TST 2JMP 2CLR "
510 OC$(1)="A$$$$B$$$$ONOP OSYNCO$$$$O$$$$5LBRA5LBSRO$$$$ODAA 3O
RCCO$$$$3ANDCOSEX 3EXG 3TFR "
```

```
520 OC$(2)="4BRA 4BRN 4BHI 4BLS 4BHS 4BLO 4BNE 4BEQ 4DVC 4BVS 4B
PL 4BMI 4BGE 4BLT 4BGT 4BLE "
530 OC$(3)="7LEAX7LEAY7LEAS7LEAU3PSHS3PULS3PSHU3PULUO$$$$ORTS OA
BX ORTI 3CWAIOMUL O$$$$OSWI "
540 OC$(4)="ONEGAO$$$$O$$$$OCOMAOLSRAO$$$$ORORAOASRAOASLAOROLAOD
ECAO$$$$OINCAOTSTAO$$$$OCLRA"
550 OC$(5)="ONEGBO$$$$O$$$$OCOMBOLSRBO$$$$ORORBOASRBOASLBOROLBOD
ECBO$$$$OINCBOTSTBO$$$$OCLRB"
560 OC$(6)="7NEG O$$$$O$$$$7COM 7LSR O$$$$7ROR 7ASR 7ASL 7ROL 7D
EC O$$$$7INC 7TST 7JMP 7CLR "
570 OC$(7)="6NEG O$$$$O$$$$6COM 6LSR 6$$$$6ROR 6ASR 6ASL 5ROL 6D
EC O$$$$6INC 6TST 6JMP 6CLR "
580 OC$(8)="3SUBA3CMPA3SBCA8SUBD3ANDA3BITA3LDA O$$$$3EORA3ADCA3O
RA 3ADDA8CMPX4BSR 8LDX O$$$$"
590 OC$(9)="2SUBA2CMPA2SBCA2SUBD2ANDA2BITA2LDA 2STA 2EORA2ADCA2O
RA 2ADDA2CMPX2JSR 2LDX 2STX "
600 OC$(10)="7SUBA7CMPA7SBCA7SUBD7ANDA7BITA7LDA 7STA 7EORA7ADCA7
ORA 7ADDA7CMPX7JSR 7LDX 7STX "
610 OC$(11)="6SUBA6CMPA6SBCA6SUBD6ANDA6BITA6LDA 6STA 6EORA6ADCA6
ORA 6ADDA6CMPX6JSR 6LDX 6STX "
620 OC$(12)="3SUBB3CMPB3SBCB8ADDD3ANDB3BITB3LDB O$$$$3EORB3ADCB3
ORB 3ADDB8LDD O$$$$8LDU O$$$$"
630 OC$(13)="2SUBB2CMPB2SBCB2ADDD2ANDB2BITB2LDB 2STB 2EORB2ADCB2
ORB 2ADDB2LDD 2STD 2LDU 2STU "
640 OC$(14)="7SUBB7CMPB7SBCB7ADDD7ANDB7BITB7LDB 7STB 7EORB7ADCB7
ORB 7ADDB7LDD 7STD 7LDU 7STU "
650 OC$(15)="6SUBB6CMPB6SBCB6ADDD6ANDB6BITB6LDB 6STB 6EORB6ADCB6
ORB 6ADDB6LDD 6SRD 6LDU 6STU "
660 REM ----- TABELLE FUER POSTBYTE $10
670 OZ$="3FOSWI283BCMPD8C8CMPY8E8LDY 932CMPD9C2CMPY9E2LDY 9F2STY
 A37CMPDAC7CMPYAE7LDY AF7STY B36CMPDBC6CMPYBE6LDY BF6LDY CE8LDS
DE2LDS DF2STS EE7LDS EF7STS FE6LDS FFSTS "
680 REM --- TABELLE FUER POSTBYTE $11
690 OE$="3FOSWI383BCMPU8C8CMPS932CMPU9C2CMPSA37CMPUAC7CMPSB36CMP
UBC6CMPS"
700 REM
710 REM ----- BEGINN DES PROGRAMMES
720 REM
730 REM --- ANFORDERN DER AUSGABEEINHEIT
740 INPUT "AUSGABE AUF ";D
750 OPEN "O",-D,"DIS"
760 GOSUB 3320
770 IF AI<>O THEN A=AI
780 REM ----- HOLEN OP-CODE
790 GOSUB 3140
800 OX$=HEX$(O)
810 REM --- ENTNEHMEN MODE UND MNEMONISCHER CODE AUS TABELLE
820 M$=MID$(OC$(HN),LN*5+1,1)
830 O$=MID$(OC$(HN),LN*5+2,4)
840 OY$="";O1$="";O2$="";PB$="";SA$=""
850 REM --- FESTSTELLEN PRE-BYTE
860 IF O<16 OR O>17 THEN GOTO 970
870 REM --- VERARBEITEN OP-CODE BEI PRE-BYTE
880 OE=PEEK(A)
890 A=A+1
900 OY$=HEX$(OE)
910 IF O=16 GOSUB 2430
920 REM ----- VERARBEITEN LONG-BRANCH
```

```
930 IF O=16 AND OE>32 AND OE<49 GOSUB 2500
940 REM --- VERARBEITEN POSTBYTE $11
950 IF O=17 GOSUB 2610
960 REM --- VERARBEITEN ADRESSIERUNGSARTEN
970 GOSUB 1050
980 REM --- AUSDRUCKEN DES AUFBEREITETEN BEFEHLES
990 GOSUB 1100
1000 REM --- AUSGABE FORTSETZEN WENN KEINE TASTE GEDRUECKT
1010 REM     NEUE ADRESSE ANFORDERN BEI BETAETIGTER TASTE
1020 IF INKEY$ <> "" GOTO 760
1030 GOTO 790
1040 REM --- VERTEILER ADRESSIERUNGSARTEN
1050 IF M$="" THEN M$="O"
1060 ON ASC(M$)-47 GOSUB 1300,1070,1330,1400,1510,1590,1760,1820
,2320
1070 RETURN
1080 REM --- DRUCKEN
1090 REM --- ZUERST FUEHRENDE NULLEN EINSETZEN
1100 IF LEN(OX$)=1 THEN OX$="O"+OX$
1110 IF LEN(OY$)=1 THEN OY$="O"+OY$
1120 IF LEN(O1$)=1 THEN O1$="O"+O1$
1130 IF LEN(O2$)=1 THEN O2$="O"+O2$
1140 IF LEN(PB$)=1 THEN PB$="O"+PB$
1150 REM --- JETZT AUSGEBEN AUF EINHEIT D
1160 IF OY$="" THEN OY$=PB$:PB$=""
1170 Z$="                    "
1180 MID$(Z$,1,LEN(A$))=A$
1190 MID$(Z$,6,LEN(OX$))=OX$
1200 MID$(Z$,9,LEN(OY$))=OY$
1210 MID$(Z$,12,LEN(PB$))=PB$
1220 MID$(Z$,15,LEN(O1$))=O1$
1230 MID$(Z$,18,LEN(O2$))=O2$
1240 Z$=Z$+O$+" "+SA$
1250 PRINT Z$:IF D=O THEN 1280
1260 IF D=1 THEN Z$=STR$(A)+" "+Z$
1270 PRINT#-D,Z$
1280 RETURN
1290 REM --- O = INHERENT 1 BYTE
1300 RETURN
1310 REM --- 2 = DIRECT 2 BYTE
1320 REM       HOLEN OPERAND ALS HEXA-WERT UND AUFBAU SYMBOLISCH
E ADRESSE
1330 O1$=HEX$(PEEK(A))
1340 A=A+1
1350 SA$="$"+O1$
1360 RETURN
1370 REM --- 3 = IMMEDIATE 2 BYTE
1380 REM     HOLEN OPERAND ALS DEZIMALWERT, UMSETZEN IN HEXA-WER
T
1390 REM     AUFBAU SYMBOLISCHE ADRESSE
1400 O1=PEEK(A)
1410 O1$=HEX$(O1)
1420 A=A+1
1430 SA$="##"+O1$
1440 REM --- SONDERBEHANDLUNG FUER TFR, EXG, PSH UND PUL
1450 IF MID$(O$,1,3)="TFR" OR MID$(O$,1,3)="EXG" THEN GOSUB 2700

1460 IF MID$(O$,1,3)="PSH" OR MID$(O$,1,3)="PUL" THEN GOSUB 2940
```

```
1470 RETURN
1480 REM --- 4 = RELATIVE 2 BYTE
1490 REM     HOLEN OPERAND ALS DEZIMALZAHL, UMSETZEN IN HEXA-WER
T,
1500 REM     AUSRICHTEN AN NULL UND VERZWEIGEN
1510 RA=PEEK(A)
1520 O1$=HEX$(RA)
1530 IF RA>127 THEN RA=RA-256
1540 GOTO 1660
1550 REM --- 5 = RELATIVE 3 BYTE
1560 REM     HOLEN DER OPERANDEN-BYTES ALS DEZIMALZAHL, UMSETZEN
 IN
1570 REM     HEXA-WERT, AN NULL AUSRICHTEN, EFFEKTIVE ADRESSE BE
RECHNEN
1580 REM     UND SYMBOLISCHE ADRESSE AUFBAUEN
1590 O1=PEEK(A)
1600 O1$=HEX$(O1)
1610 A=A+1
1620 O2=PEEK(A)
1630 O2$=HEX$(O2)
1640 RA=O1*256+O2
1650 IF RA>32767 THEN RA=RA-65536
1660 A=A+1
1670 EA=A+RA
1680 EH=INT(EA/256)
1690 EL=EA-EH*256
1700 EL$=HEX$(EL)
1710 IF LEN (EL$)=1 THEN EL$="0"+EL$
1720 SA$="$"+O1$+O2$+"="$"+HEX$(EH)+EL$
1730 RETURN
1740 REM --- 6 = EXTENDED 3 BYTE
1750 REM     HOLEN DER OPERANDEN-BYTES ALS HEXA-WERT, AUFBAU SYM
B.ADRESSE
1760 O1$=HEX$(PEEK(A))
1770 A=A+1
1780 O2$=HEX$(PEEK(A)):IF LEN(O2$)=1 THEN O2$="0"+O2$
1790 A=A+1
1800 SA$="$"+O1$+O2$
1810 RETURN
1820 REM --- 7 = INDEXED 2+ BYTE
1830 REM     POSTBYTE ALS DEZIMALZAHL HOLEN UND IN HEXA-WERT UMS
ETZEN
1840 IX=PEEK(A)
1850 PB$=HEX$(IX)
1860 A=A+1
1870 REM --- AUFBEREITEN DES POSTBYTE IN FLAG 5-BIT-ADRESSE, REG
ISTERNUMMER
1880 REM     UND INDIREKT-FLAG
1890 F5=INT(IX/128):IX=IX-F5*128
1900 RR=INT(IX/32):IX=IX-RR*32
1910 FI=INT(IX/16):IX=IX-FI*16
1920 REM --- IN IX STEHT NUR NOCH INDEXTYP BZW. DISPLACEMENT BEI
 5 BIT-OFFSET
1930 REM --- EINSETZEN INDEXREGISTER
1940 SA$=",X"
1950 IF RR=1 THEN SA$=",Y"
1960 IF RR=2 THEN SA$=",U"
1970 IF RR=3 THEN SA$=",S"
```

```
1980 REM --- EINSETZEN PC-REALTIV
1990 IF IX=12 OR IX=13 THEN SA$=",PCR"
2000 REM --- 5-BIT-OFFSET UND INDIREKTE ADRESSIERUNG VERZWEIGEN
2010 IF F5=1 GOTO 2100
2020 IF FI=1 THEN GOTO 2070
2030 REM --- 5-BIT-OFFSER VORWAERTS
2040 SA$="$"+HEX$(IX)+SA$
2050 GOTO 2280
2060 REM --- 5-BIT-OFFSET RUECKWAERTS
2070 SA$="-$"+HEX$(ABS(IX-16))+SA$
2080 GOTO 2280
2090 REM ----- KEIN OFFSET
2100 IF IX=4 THEN GOTO 2270
2110 REM --- VERARBEITEN AUTO-INCREMENT/DECREMENT
2120 IF IX=0 THEN SA$=SA$+"+":GOTO 2280
2130 IF IX=1 THEN SA$=SA$+"++":GOTO 2270
2140 IF IX=2 THEN SA$=",-"+RIGHT$(SA$,1):GOTO 2280
2150 IF IX=3 THEN SA$=",--"+RIGHT$(SA$,1):GOTO 2270
2160 REM --- ACCUMULATOR-OFFSET AUFBAUEN
2170 IF IX=5 THEN SA$="B"+SA$:GOTO 2270
2180 IF IX=6 THEN SA$="A"+SA$:GOTO 2270
2190 IF IX=11 THEN SA$="D"+SA$:GOTO 2270
2200 O1$=HEX$(PEEK(A)):A=A+1
2210 REM --- BEI 16-BIT-OPERAND ZWEITES BYTE HOLEN
2220 IF IX=8 OR IX=12 THEN GOTO 2250
2230 O2$=HEX$(PEEK(A)):A=A+1:IF LEN(O2$)=1 THEN O2$="0"+O2$
2240 REM --- SYMBOL. ADRESSE AUS OPERANDEN AUFBAUEN
2250 SA$="$"+O1$+O2$+SA$
2260 REM --- ERGAENZEN SYMBOL.ADRESSE BEI INDIREKTER ADRESSIERUN
G
2270 IF F1=1 THEN SA$="("+SA$+")"
2280 RETURN
2290 REM --- 8 = IMMEDIATE 3 BYTE
2300 REM      HOLEN ERSTES UND ZWEITES BYTE DES OPERANDEN,
2310 REM      ZWEITES BYTE EVTL. UM FUEHRENDE NULL ERGAENZEN
2320 O1$=HEX$(PEEK(A))
2330 A=A+1
2340 O2$=HEX$(PEEK(A)):IF LEN(O2$)=1 THEN O2$="0"+O2$
2350 A=A+1
2360 REM --- SYMBOLISCHE ADRESSE AUS CODE FUER ADRESSIERUNG UND
2370 REM      OPERAND AUFBAUEN
2380 SA$="##$"+O1$+O2$
2390 RETURN
2400 REM --- VERARBEITEN OPCODE NACH PREBYTE 10
2410 REM      DURCHSUCHEN BEFEHLSCODE-TABELLE NACH HEXA-KOMBINATI
ON
2420 REM      WENN GEFUNDEN ENTNEHMEN MODE UND MNEMON. CODE AUS T
ABELLE
2430 FOR I=1TO161 STEP7
2440 IF OY$=MID$(OZ$,I,2) THEN GOTO 2470
2450 NEXT I
2460 GOTO 2490
2470 M$=MID$(OZ$,I+2,1)
2480 O$=MID$(OZ$,I+3,4)
2490 RETURN
2500 REM --- VERARBEITEN LONG BRANCH
2510 REM      ISOLIEREN LOW-NIBBLE AUS BEFEHLSBYTE
2520 REM      MIT DEM LOW-NIBBLE MNEM. BEFEHL AUS TABELLE ENTNEHM
EN
```

```
2530 REM      UND ZEICHEN L DAVOR SETZEN, MODE AUF 5 STELLEN
2540 LN=OE-INT(OE/16)*16
2550 M$="5"
2560 O$="L"+MID$(OC$(2),LN*5+2,3)
2570 RETURN
2580 REM --- VERARBEITEN OPCODE NACH PREBYTE 11
2590 REM      DURCHSUCHEN BEFEHLSCODE-TABELLE NACH HEXA-KOMBINATI
ON
2600 REM      WENN GEFUNDEN ENTNEHMEN MODE UND MNEMON. CODE AUS T
ABELLE
2610 FOR I=1TO70 STEP7
2620 IF OY$=MID$(OE$,I,2) THEN GOTO 2650
2630 NEXT I
2640 GOTO 2670
2650 M$=MID$(OE$,I+2,1)
2660 O$=MID$(OE$,I+3,4)
2670 RETURN
2680 REM --- VERARBEITEN ADRESSE BEI TFR UND EXG
2690 REM --- LOESCHEN SYMBOLISCHE ADRESSE
2700 SA$=""
2710 REM --- ISOLIEREN HIGH-NIBBLE ALS REGISTER
2720 RR=INT(O1/16)
2730 GOSUB 2790
2740 REM --- ISOLIEREN LOW-NIBBLE ALS REGISTER
2750 RR=O1-RR*16
2760 REM --- EINSETZEN KOMMA ALS TRENNUNG DER REGISTERBEZEICHNUN
HG
2770 SA$=SA$+","
2780 REM --- EINSETZEN DER REGISTERBEZEICHNUNG
2790 IF RR=0 THEN SA$=SA$+"D":GOTO 2910
2800 IF RR=1 THEN SA$=SA$+"X":GOTO 2910
2810 IF RR=2 THEN SA$=SA$+"Y":GOTO 2910
2820 IF RR=3 THEN SA$=SA$+"U":GOTO 2910
2830 IF RR=4 THEN SA$=SA$+"S":GOTO 2910
2840 IF RR=5 THEN SA$=SA$+"PC":GOTO 2910
2850 IF RR=8 THEN SA$=SA$+"A":GOTO 2910
2860 IF RR=9 THEN SA$=SA$+"B":GOTO 2910
2870 IF RR=10 THEN SA$=SA$+"CC":GOTO 2910
2880 IF RR=11 THEN SA$=SA$+"DP":GOTO 2910
2890 REM --- EINSETZEN VON **** WENN KEIN GUELTIGER REGISTER
2900 SA$=SA$+"****"
2910 RETURN
2920 REM --- VERARBEITEN ADRESSE BEI PUSH UND PULL
2930 REM      ZUERST AUSBLANKEN SYMBOLISCHE ADRESSE
2940 SA$=""
2950 REM --- ABFRAGEN DER EINZELNEN BITS DES DIREKT-OPERANDEN UN
D
2960 REM      BEI GESETZTEM BIT REGISTERBEZEICHNUNG EINSETZEN
2970 REM      UND BIT-WERT VOM DIREKTOPERANDEN ABZIEHEN FUER
2980 REM      WEITERE BEARBEITUNG
2990 RR=INT(O1/128):O1=O1-RR*128:IF RR=1 THEN SA$=SA$+"PC,"
3000 RR=INT(O1/64):O1=O1-RR*64
3010 IF RR=1 AND MID$(O$,4,1)="U" THEN SA$=SA$+"S,"
3020 IF RR=1 AND MID$(O$,4,1)="S" THEN SA$=SA$+"U,"
3030 RR=INT(O1/32):O1=O1-RR*32:IF RR=1 THEN SA$=SA$+"Y,"
3040 RR=INT(O1/16):O1=O1-RR*16:IF RR=1 THEN SA$=SA$+"X,"
3050 RR=INT(O1/8):O1=O1-RR*8:IF RR=1 THEN SA$=SA$+"DP,"
3060 RR=INT(O1/4):O1=O1-RR*4:IF RR=1 THEN SA$=SA$+"B,"
```

```
3070 RR=INT(O1/2):O1=O1-RR*2:IF RR=1 THEN SA$=SA$+"A,"
3080 IF O1=1 THEN SA$=SA$+"CC,"
3090 REM --- AUSBLANKEN DES LETZTEN KOMMA IN REGISTERAUSGABE
3100 MID$(SA$,LEN(SA$),1)=" "
3110 RETURN
3120 REM --- HOLEN OP-CODE
3130 REM --- AUFSPALTEN ADRESSE IN HIGH- UND LOW-BYTE ALS DEZIMA
LWERT
3140 AH=INT(A/256)
3150 AL=A-AH*256
3160 REM --- HOLEN BEFEHLSBYTE ALS DEZIMALZAHL
3170 O=PEEK(A)
3180 REM --- AUFSPALTEN DES BEFEHLSBYTES IN HIGH- UND LOW-NIBBLE

3190 HN=INT(O/16)
3200 LN=O-HN*16
3210 REM --- ERHOEHEN DER ADRESSE
3220 A=A+1
3230 REM --- UMSETZEN DER ADRESSE IN HIGH- UND LOW-HEXA-WERTE
3240 REM      UND EVTL. ERGAENZEN UM VORNULL
3250 AH$=HEX$(AH)
3260 IF LEN(AH$)=1 THEN AH$="0"+AH$
3270 AL$=HEX$(AL)
3280 IF LEN(AL$)=1 THEN AL$="0"+AL$
3290 A$=AH$+AL$
3300 RETURN
3310 REM --- EINGABE DER STARTADRESSE
3320 PRINT CHR$(13);
3330 PRINT "STARTADRESSE ";
3340 AI=0
3350 REM --- ZEICHEN EINGEBEN
3360 I$=INKEY$:IF I$="" THEN 3360
3370 IF ASC(I$)=13 THEN 3560
3380 IF ASC(I$)=32 THEN 3600
3390 REM --- ECHO DES EINGEGEBEN ZEICHENS
3400 PRINT I$;
3410 REM --- UMSETZEN DES EINGEGEBENEN ZEICHENS IN DEZIMALZAHL
3420 I=ASC(I$)-48
3430 REM --- UMSETZEN VON WERT GROSSER DEZIMAL 10
3440 IF I>9 THEN I=I-7
3450 REM --- PRUEFEN AUF GUELTIGES ZEICHEN
3460 IF I<0 THEN 3530
3470 IF I>15 THEN 3530
3480 REM --- ALTE EINGABE UM 4BIT NACH LINKS SCHIEBEN
3490 REM      UND NEUE EINGABE ANFUEGEN
3500 AI=AI*16+I
3510 GOTO 3360
3520 REM --- FRAGEZEICHEN AUSGEBEN BEI FEHLERHAFTER EINGABE
3530 PRINT "?";
3540 GOTO 3360
3550 REM --- NORMIEREN DES EINGABEWERTES AUF 2 BYTE LAENGE
3560 AI=AI-(INT(AI/65536)*65536)
3570 PRINT#-D,CHR$(13)
3580 IF D <> 0 THEN PRINT
3590 RETURN
3600 CLOSE -D
3610 END                                                    #
3620 REM
```

Ing. grad. Horst Steder, 6000 Frankfurt

Fast Assembler (AIM)

Im 'AIM-Spezial Nr. 10' (Heft 21 des MICRO MAG) wurde eine Methode beschrieben, den AIM-Assembler durch Ausschalten des Displays um den Faktor 5 schneller zu machen. Dazu ist allerdings ein Umkopieren des Original-ROMs nowendig. Die nachfolgende kleine Routine ermöglicht nun das Gleiche mit dem unveränderten Original-ROM.

```
0190 B0 BCS 0197
0192 A9 LDA #00
0194 85 STA AC
0196 60 RTS
0197 A5 LDA AC
0199 D0 BNE 01A4
019B A5 LDA 23
019D F0 BEQ 01A4
019F E6 INC AC
01A1 4C JMP D0C3
01A4 68 PLA
01A5 68 PLA
01A6 20 JSR FAD0
01A9 4C JMP D10A
```

Man setzt den User Input Handler ($108/109) auf die Startadresse (hier $190) und beantwortet nach Aufruf des Assemblers den Prompt 'IN=' mit 'U' statt 'M'. Diese einfache Methode ist leider nur beim Assemblieren des im Editor-RAM residenten Quelltextes möglich.

Das folgende Programm, das hier seit zwei Jahren erfolgreich eingesetzt wird, stellt eine Erweiterung des in Heft 9, Seite 17 vorgestellten 'TPASM' dar. Es bietet:

FAST-Methode durch Umgehung des Displays

Erzeugung von Zeilen-Nummern aus dem Source-Text

File-Linking über den User-Vektor.

Wie auch schon om oben zitierten 'AIM Spezial' erwähnt, erlaubt das Assembler-ROM kein File-Linking über den User Input-Vektor, weil an entscheidender Stelle ein CLC vergessen (?) wurde. Diese Schwierigkeit konnte durch eine Stackabfrage überwunden werden.

Nun zur Anwendung des Programmes: Die Bedienung erfolgt genau wie beim Standard-Assembler unter Verwendung der User-Commands. Folgende Optionen sind möglich:

Source-Input:	IN=U=M	(Memory)
	IN=U=T	(AIM-Tape)
	IN=U=A	(Anwender-Routine)
List-Output:	OUT=U=C	(Centronics, parallel)
	OUT=U=S	(serielle Ausgabe über TTY mit 300 Baud on default)
	OUT=U=A	(Anwender-Routine).

Für die Anwender-Option sind folgende IN-/OUT-Handler in der Page 1 vorgesehen:

Input	= $16C/16D
Output	= $16E/16F.

Das Programm wird durch den Start mit dem 'G'-Kommando des Monitors (hier bei hex 5000) initialisiert und meldet sich mit dem Prompt 'LINAS V2.1'. Danach sind die User-Vektoren in $108 bis 10B gesetzt, ferner die Zahl der Zeilen je Seite auf 66 und die Zahl der Leerzeilen für Formularvorschub auf 6. Die BAudrate ist auf 300 eingestellt. Alle diese Parameter lassen sich nach der Initialisierung noch ändern. Wird z.B. die Zeilenzahl in $D9 auf 00 gesetzt, so erfolgt

kein paging. Der Assembler wird wie üblich mit (N) aufgerufen. Hierzu noch einige Beispiele:

1) Quelltext im Speicher, Listung auf Centronics, Objectcode auf Tape:

```
(N)
ASSEMBLER
FROM=1000 TO=2000
IN=UIN=M
LIST?Y
LIST-OUT=U=C
OBJ?Y
OBJ-OUT=T
```

2) Quelltext in einem Anwender-File, Listung auf Anwender, Object auf Tape:

```
(N)
ASSEMBLER
FROM=1000 TO=2000
IN=UIN=A        (muß entsprechend initialisiert sein)
LIST?Y
LIST-OUT=U=A    (wie oben)
OBJ?Y
OBJ-OUT=T
```

3) Quelltext auf Tape, Listung seriell (TTY), Objectcode auf Memory:

```
(N)
ASSEMBLER
FROM=1000 TO=2000
IN=UIN=T
LIST?Y
LIST-OUT=S
OBJ?N
```

Zum Gebrauch der anwenderbestimmten Treiberroutinen (A) folgende Hinweise: Wie beim U-Kommando des AIM gelten folgende Regeln:

a) Die Software-Treiber müssen Unterprogramme sein.
b) Carry Clear = Sprung in die Initialisierung,
   Carry Set = Sprung in die Anwender-Routine.
c) Bei Anwendung des Output-Handlers unter (A) braucht (im Gegensatz zum U-Kommando) kein 'PLA' vorgesehen zu werden.

Wie bereits erwähnt, ist auch bei (A) ein File-Linking durchführbar. Damit ist also die Verwendung einer Floppy Disk ohne Probleme ermöglicht. Allerdings ist beim Assemblieren von mehreren Files eine zusätzliche Anfangsinformation nötig, da ja vor dem Start der Name des ersten Files eingegeben werden muß. Hierzu kann das X-Register dienen, das nach dem Aufruf des Assemblers definiert den Wert 1 enthält. Ein Beispiel dafür findet sich ganz am Schluß dieser Darstellung.

Zwischen PASS1 und PASS2 wurde übrigens auch hier eine Warteschleife für U=T und U=A eingebaut (Start von PASS2 erst nach Drücken der Space-Taste). Falls das z.B. beim Floppy-Betrieb stört, kann man einfach die Zeilen 305-307 fortlassen.

Noch etwas: Der Assembler erwartet den jeweiligen File-Namen im Namen-Buffer für den AIM-Tape ($A42E, max. 5 Zeichen). Eine Floppy-Routine muß also für den entsprechenden Eintransport sorgen. Weitere Erklärungen möge der Leser bitte den Kommentaren in der nachfolgenden Liste entnehmen.

```
0000  0000          ;*************************************************
0001  0000          ;*              ----- L I N A S -----            *
0002  0000          ;*  FAST ASSEMBLER WITH LINE-NUMBERS AND FILE-   *
0003  0000          ;*    LINKING CAPABILITY VIA THE USER INPUT      *
0004  0000          ;*                COPYRIGHT BY H.STEDER  09/80   *
0005  0000          ;*************************************************
0006  0000
0007  0000 BAUDLO         =$C               ;VALUE FOR RATE OF 300 BAUD
0008  0000 BAUDHI         =$C2
0009  0000 NLINES         =66               ;NUMBER OF LINES/PAGE
0010  0000 NPGE           =6                ;LINE-FEEDS FOR PAGING
0011  0000 PSFLG          =$23
0012  0000 LINCNT         =$D9
0013  0000 PAGE           =$DA
0014  0000 PRIFLG         =$DB
0015  0000 LINUM          =$DD              ;COUNTER FOR LINE-#
0016  0000 IFLG1          =$E9              ;USER INPUT FLAG
0017  0000 STFLG          =$EA              ;START FLAG FOR PASS2
0018  0000 SAVA           =$EB
0019  0000 SAVX           =$EC
0020  0000 SMFLG          =$ED
0021  0000 ICNT           =$EE
0022  0000 EQFLG          =$EF
0023  0000 LCNTR          =$F0
0024  0000 CRFLG          =$F1
0025  0000 CNTR           =$F2
0026  0000 INUSR          =$16C             ;USER INPUT VECTOR FOR 'A'
0027  0000 OUTUSR         =$16E             ;USER OUTPUT VECTOR FOR 'A'
0028  0000 ASMIN          =$D0C3            ;ASSEMBLER INPUT LOOP
0029  0000 ASMCHR         =$D10A            ;ASM READS A CHAR FROM BUFFER
0030  0000 AIM65          =$E1A1
0031  0000 LOADTA         =$E32F
0032  0000 EQUAL          =$E7D8
0033  0000 FNAM           =$E8A2
0034  0000 RDRUB          =$E95F
0035  0000 KEPR           =$E970
0036  0000 REDOUT         =$E973
0037  0000 OUTALL         =$E9BC
0038  0000 TIBYTE         =$ED3B
0039  0000 OUTTTY         =$EEA8
0040  0000 MREAD          =$FAD0
0041  0000
0042  0000                *=$5000           ;+++ START OF ROUTINE
0043  5000
0044  5000                ------------------
0044  5000                --- INIT-ROUTINE ---
0044  5000                ------------------
0045  5000 INITA  A000    LDY #0
0046  5002 INIA   B94652  LDA MS1,Y         ;DISPLAY MESSAGE
0047  5005        20BCE9  JSR OUTALL
0048  5008        C8      INY
0049  5009        C00D    CPY #13
0050  500B        D0F5    BNE INIA
0051  500D        A042    LDY #NLINES       ;SET DEFAULT VALUES FOR
0052  500F        84D9    STY LINCNT        ;NUMBER OF LINES/PAGE (=66)
0053  5011        A006    LDY #NPGE         ;AND LINE FEEDS (=6)
0054  5013        84DA    STY PAGE          ;FOR PAGING
0055  5015        A0C2    LDY #BAUDHI       ;SET BAUD RATE FOR OUTTTY
0056  5017        A90C    LDA #BAUDLO       ;TO THE DEFAULT VALUE (=300)
0057  5019        8D17A4  STA $A417
0058  501C        8C18A4  STY $A418
0059  501F        A9A9    LDA #<USERIN      ;SET USER INPUT VECTOR
0060  5021        A051    LDY #>USERIN
0061  5023        8D0801  STA $108
0062  5026        8C0901  STY $109
0063  5029        A936    LDA #<TPASM       ;SET USER OUTPUT VECTOR
0064  502B        A050    LDY #>TPASM
0065  502D        8D0A01  STA $10A
0066  5030        8C0B01  STY $10B
0067  5033        4CA1E1  JMP AIM65         ;BACK TO MONITOR
0068  5036
0069  5036                ------------------
0069  5036                ASSEMBLER-FORMATTER
0069  5036                ------------------
0070  5036 TPASM  B036    BCS TDSP          ;INITIALISATION:
0071  5038        20D8E7  JSR EQUAL
```

```
0072  503B TPASM1 A200    LDX #0
0073  503D         86F2    STX CNTR          ;RESET COLUMN COUNTER
0074  503F         86F0    STX LCNTR         ;RESET LINE COUNTER
0075  5041         86ED    STX SMFLG         ;RESET SEMI-FLAG
0076  5043         205FE9  JSR RDRUB         ;WHICH OUTPUT DEVICE?
0077  5046         A240    LDX #$40
0078  5048         86DB    STX PRIFLG        ;SET BIT 6 OF OUTPUT FLAG
0079  504A         C943    CMP #'C'          ;CENTRONICS INTERFACE?
0080  504C         D011    BNE OUT1
0081  504E         A2FF    LDX #$FF
0082  5050         8E03A0  STX $A003         ;PAD AS OUTPUT TO CENTRONICS
0083  5053         A20A    LDX #%00001010    ;SET CA1 AS 'BUSY' INPUT (NEG)..
0084  5055         8E0CA0  STX $A00C         ;AND CA2 AS PULSE STROBE
0085  5058         A97F    LDA #$7F
0086  505A         8D0EA0  STA $A00E         ;CLEAR INTERRUPT ENABLE REG
0087  505D         D00C    BNE RETURN        ;BRANCH ALWAYS
0088  505F OUT1   06DB    ASL PRIFLG        ;SET BIT 7 OF OUTPUT FLAG
0089  5061         C941    CMP #'A'          ;USER DEVICE?
0090  5063         D004    BNE OUT2
0091  5065         18      CLC               ;CLEAR CARRY FOR INIT.
0092  5066         6C6E01  JMP (OUTUSR)      ;TO USER DEVICE
0093  5069 OUT2   06DB    ASL PRIFLG        ;SERIAL OUT, RESET PRINTER FLAG
0094  506B RETURN 4C1B51  JMP CRLF2         ;RESET PRINTER
0095  506E
0096  506E         ; --- START OF FORMATTING ROUTINE ---
0097  506E TDSP   86EC    STX SAVX          ;SAVE X-REG
0098  5070         A223    LDX #35           ;DEF START COL# FOR COMMENT FIELD
0099  5072         86EE    STX ICNT
0100  5074         68      PLA
0101  5075         297F    AND #$7F          ;STRIP OFF PARITY BIT
0102  5077         85EB    STA SAVA          ;SAVE ACCU
0103  5079         C90D    CMP #$0D          ;'CR' ?
0104  507B         D005    BNE N1            ;NO 'CR', CHECK FOR '='
0105  507D         A201    LDX #1
0106  507F         86F1    STX CRFLG         ;'CR', SET FLAG, DON'T EXECUTE
0107  5081         60      RTS
0108  5082
0109  5082 N1     C93D    CMP #'='          ;FIRST '=' IN LINE?
0110  5084         D015    BNE N6
0111  5086         A6F1    LDX CRFLG         ;IF CR-FLAG SET AT FIRST '='
0112  5088         F009    BEQ N2
0113  508A         201B51  JSR CRLF2         ;THEN EXECUTE CR
0114  508D         A201    LDX #1
0115  508F         86EF    STX EQFLG         ;SET EQUAL FLAG
0116  5091         D062    BNE R1            ;RETURN
0117  5093 N2     A6EF    LDX EQFLG         ;SECOND '=' ?
0118  5095         F05B    BEQ NEXT          ;NO, OUTPUT CHAR
0119  5097         46EF    LSR EQFLG         ;CLEAR EQUAL FLAG
0120  5099         F05A    BEQ R1            ;RETURN
0121  509B N6     A6F1    LDX CRFLG
0122  509D         F03A    BEQ N10
0123  509F         C93B    CMP #';'          ;FIRST CHAR AFTER 'CR' A SEMICOLON?
0124  50A1         D024    BNE N8
0125  50A3         46F1    LSR CRFLG         ;YES, CLEAR CR-FLAG
0126  50A5         A201    LDX #1
0127  50A7         86ED    STX SMFLG         ;AND SET SEMI-FLAG
0128  50A9         D04A    BNE R1
0129  50AB NC     A5F2    LDA CNTR
0130  50AD         C922    CMP #34           ;IF OPERAND FIELD EXCEEDS COL# 34
0131  50AF         9004    BCC SP1           ;THEN SHIFT COMMENT FIELD
0132  50B1         6901    ADC #1            ;BACK ACCORDINGLY
0133  50B3         85EE    STA ICNT
0134  50B5 SP1    A6F2    LDX CNTR
0135  50B7         E4EE    CPX ICNT          ;CURRENT POSITION=COL #35?
0136  50B9         F005    BEQ N7
0137  50BB         20F050  JSR SPACE         ;NO, INSERT SPACES UNTIL #=35
0138  50BE         D0F5    BNE SP1
0139  50C0 N7     A93B    LDA #';'
0140  50C2         20F250  JSR NEXT          ;OUTPUT A SEMICOLON
0141  50C5         D00B    BNE NN            ;AND FIRST CHAR OF COMMENT
0142  50C7 N8     A6F2    LDX CNTR
0143  50C9         E00C    CPX #12           ;CURRENT POS <=#12 ?
0144  50CB         B009    BCS N9
0145  50CD         46F1    LSR CRFLG         ;YES, CLEAR CR-FLAG AND
0146  50CF         207751  JSR CONTR         ;INSERT SPACES UNTIL POS #12
0147  50D2 NN     A5EB    LDA SAVA
0148  50D4         101C    BPL NEXT          ;JMP ALWAYS
```

```
0149  50D6 N9      207151 JSR RESET        ;START NEW LINE
0150  50D9 N10     A5EB   LDA SAVA
0151  50DB         A6ED   LDX SMFLG        ;SEMIFLAG SET ?
0152  50DD         F00D   BEQ N11          ;NO
0153  50DF         46ED   LSR SMFLG        ;YES, CLEAR FLAG
0154  50E1         C92E   CMP #'.'         ;IF A "." FOLLOWS A ";"
0155  50E3         D0C6   BNE NC
0156  50E5         207151 JSR RESET        ;THEN START COMMENT FIELD...
0157  50E8         A93B   LDA #';'         ;AT COLUMN #13
0158  50EA         D006   BNE NEXT
0159  50EC N11     C97D   CMP #$7D         ;ANY '}' (FILLER CHAR)?
0160  50EE         D002   BNE NEXT
0161  50F0 SPACE   A920   LDA #$20         ;YES, REPLACE WITH SPACE
0162  50F2 NEXT    203151 JSR ASMOUT
0163  50F5 R1      A6EC   LDX SAVX
0164  50F7         60     RTS
0165  50F8
0166  50F8         ;    *** UTILITY SUBROUTINES ***
0167  50F8 CRLF1   A90D   LDA #$0D         ;CR/LF TO OUTPUT DEVICE
0168  50FA         203151 JSR ASMOUT
0169  50FD         A90A   LDA #$A
0170  50FF         203151 JSR ASMOUT
0171  5102         A6D9   LDX LINCNT       ;IF LINE COUNT =00, THEN..
0172  5104         F014   BEQ RET0         ;NO PAGE ADVANCE AT ALL
0173  5106         E6F0   INC LCNTR        ;INCREMENT LINE COUNT
0174  5108         A6F0   LDX LCNTR
0175  510A         E4D9   CPX LINCNT       ;PAGE FULL (DEFAULT=66 LINES) ?
0176  510C         900C   BCC RET0         ;NOT YET
0177  510E PGADV   A4DA   LDY PAGE         ;YES, MOVE TO NEXT PAGE
0178  5110 PGDV    A90A   LDA #$A          ;BY INSERTION OF..
0179  5112         203151 JSR ASMOUT
0180  5115         88     DEY              ;6 LINE FEEDS (DEFAULT)
0181  5116         D0F8   BNE PGDV
0182  5118         84F0   STY LCNTR        ;RESET LINE COUNTER
0183  511A RET0    60     RTS
0184  511B
0185  511B CRLF2   20F850 JSR CRLF1
0186  511E         A523   LDA PSFLG        ;PASS 2 FLAG SET?
0187  5120         D005   BNE LIN1         ;YES
0188  5122         209251 JSR NOPR         ;NO, JUST PRINT 6 SPACES
0189  5125         F003   BEQ LIN2         ;JMP ALWAYS
0190  5127 LIN1    208151 JSR LINUMR       ;PRINT LINE-#
0191  512A LIN2    A900   LDA #0
0192  512C         85F2   STA CNTR
0193  512E         85F1   STA CRFLG
0194  5130         60     RTS
0195  5131
0196  5131 ASMOUT  E6F2   INC CNTR
0197  5133         A6DB   LDX PRIFLG       ;WHICH OUTPUT DEVICE?
0198  5135         D003   BNE NXTOUT
0199  5137         4CA8EE JMP OUTTTY       ;SERIAL OUTPUT
0200  513A NXTOUT  1004   BPL CENTR       ;CENTRONICS OUT
0201  513C         38     SEC
0202  513D         6C6E01 JMP (OUTUSR)     ;USER DEVICE
0203  5140 CENTR   AA     TAX              ;SAVE CHAR
0204  5141 HDSHK   A902   LDA #2           ;WAIT FOR ACKNOWLEDGE
0205  5143         2C0DA0 BIT $A00D        ;'BUSY' OFF ?  (IFR-BIT 1)
0206  5146         F0F9   BEQ HDSHK        ;NO, WAIT
0207  5148         8E01A0 STX $A001        ;CHAR TO CENTRONICS
0208  514B         8A     TXA
0209  514C         60     RTS
0210  514D
0211  514D ADDRS   A533   LDA $33          ;ADDRS HI-BYTE
0212  514F         205A51 JSR NUMA1
0213  5152         A532   LDA $32          ;ADDRS LO-BYTE
0214  5154 ADDR1   205A51 JSR NUMA1
0215  5157         4CF050 JMP SPACE
0216  515A
0217  515A NUMA1   48     PHA              ;HEX TO 2 ASCII-CHARS
0218  515B         4A     LSR A
0219  515C         4A     LSR A
0220  515D         4A     LSR A
0221  515E         4A     LSR A
0222  515F         206551 JSR NOUT
0223  5162         68     PLA
0224  5163         290F   AND #$F
0225  5165 NOUT    18     CLC
```

```
0226  5166          6930    ADC #$30
0227  5168          C93A    CMP #$3A
0228  516A          9002    BCC LT10
0229  516C          6906    ADC #6
0230  516E LT10     4CF250  JMP NEXT
0231  5171
0232  5171 RESET    201B51  JSR CRLF2        ;START NEW LINE
0233  5174          204D51  JSR ADDRS
0234  5177
0235  5177 CONTR    20F050  JSR SPACE
0236  517A          A6F2    LDX CNTR
0237  517C          E00C    CPX #12
0238  517E          D0F7    BNE CONTR
0239  5180 RET1     60      RTS
0240  5181
0241  5181 LINUMR    A5DE    LDA LINUM+1
0242  5183          C999    CMP #$99        ;IF LINE-# >9899..
0243  5185          B00B    BCS NOPR        ;THEN DON'T PRINT
0244  5187          205A51  JSR NUMA1       ;PRINT LINE-#
0245  518A          A5DD    LDA LINUM
0246  518C          205451  JSR ADDR1
0247  518F          4CF050  JMP SPACE
0248  5192 NOPR     A206    LDX #6          ;INSERT 6 SPACES
0249  5194          86F2    STX CNTR
0250  5196          4C7751  JMP CONTR
0251  5199
0252  5199 LINADD    F8      SED             ;INCREMENT LINE NUMBER
0253  519A          18      CLC             ;IN DECIMAL MODE
0254  519B          A5DD    LDA LINUM
0255  519D          6901    ADC #1
0256  519F          85DD    STA LINUM
0257  51A1          A5DE    LDA LINUM+1
0258  51A3          6900    ADC #0
0259  51A5          85DE    STA LINUM+1
0260  51A7          D8      CLD
0261  51A8          60      RTS
0262  51A9
0263  51A9 USERIN    B02C    BCS USRIN       ;ASSEMBLER INPUT LOOP
0264  51AB          A299    LDX #$99
0265  51AD          86DD    STX LINUM       ;PRESET LINE-# COUNTER
0266  51AF          86DE    STX LINUM+1
0267  51B1          A201    LDX #1
0268  51B3          86EA    STX STFLG       ;SET START FLAG (PASS2)
0269  51B5          A02A    LDY #$2A        ;DISPLAY "IN=" AND..
0270  51B7          2070E9  JSR KEPR        ;GET INPUT DEVICE
0271  51BA          85E9    STA IFLG1       ;STORE IN USER INFLAG
0272  51BC          C941    CMP #'A'        ;USER?
0273  51BE          D004    BNE TAPE
0274  51C0 NXTF1     18      CLC
0275  51C1          6C6C01  JMP (INUSR)
0276  51C4 TAPE     C954    CMP #'T'        ;TAPE?
0277  51C6          D0B8    BNE RET1        ;NO, MEMORY
0278  51C8          CA      DEX             ;X=0 FOR INPT FILE FLAG
0279  51C9          20A2E8  JSR FNAM        ;OPEN FILE FOR TAPE
0280  51CC NXTFI1    4C2FE3  JMP LOADTA      ;GET FILE
0281  51CF
0282  51CF NXTFIL    A5E9    LDA IFLG1
0283  51D1          C941    CMP #'A'
0284  51D3          F0E8    BEQ NXTF1       ;GET USER FILE
0285  51D5          D0F5    BNE NXTFI1      ;GET TAPE FILE
0286  51D7
0287  51D7 USRIN    A5E9    LDA IFLG1       ;TEST USER INFLAG
0288  51D9          C94D    CMP #'M'        ;MEMORY?
0289  51DB          F04B    BEQ MREAD1
0290  51DD          C941    CMP #'A'        ;USER?
0291  51DF          F004    BEQ STKTST
0292  51E1          C954    CMP #'T'        ;TAPE ?
0293  51E3          D043    BNE MREAD1      ;NO, MUST BE MEMORY
0294  51E5 STKTST    ADFD01  LDA $1FD        ;CHECK STACK WHETHER A...
0295  51E8          C9D6    CMP #$D6        ;".FIL" STATEMENT WAS FOUND
0296  51EA          D007    BNE PS20
0297  51EC          ADFC01  LDA $1FC        ;(RETURN ADDR = $D698)
0298  51EF          C997    CMP #$97
0299  51F1          F0DC    BEQ NXTFIL      ;IF SO, GET NEXT TAPE FILE
```

```
0300  51F3 PS20   A523    LDA PSFLG       ;PASS1 OR PASS2 ?
0301  51F5        F01F    BEQ PS22        ;STILL PASS1
0302  51F7        A5EA    LDA STFLG       ;PASS2 STARTED ?
0303  51F9        F01B    BEQ PS22        ;YES, GO AHEAD
0304  51FB        46EA    LSR STFLG       ;CLEAR START FLAG (PASS2)
0305  51FD PS21   2073E9  JSR REDOUT      ;WAIT FOR <SPACE>...
0306  5200        C920    CMP #' '        ;TO START PASS2
0307  5202        D0F9    BNE PS21
0308  5204        A5E9    LDA IFLG1
0309  5206        C941    CMP #'A'
0310  5208        D006    BNE LTPE
0311  520A        20C051  JSR NXTF1
0312  520D        4CC3D0  JMP ASMIN       ;TO START OF ASMBLR INPUT
0313  5210 LTPE   202FE3  JSR LOADTA      ;GET FILE FROM TAPE
0314  5213        4CC3D0  JMP ASMIN
0315  5216 PS22   A5E9    LDA IFLG1
0316  5218        C941    CMP #'A'        ;INPUT FROM USER FILE?
0317  521A        D006    BNE TBYT        ;NO, MUST BE TAPE
0318  521C        20C151  JSR NXTF1+1     ;GET NEXT CHAR FROM USER
0319  521F        4C2852  JMP MREAD1+3
0320  5222 TBYT   203BED  JSR TIBYTE      ;READ FROM TAPE BUFFER
0321  5225        4C2B52  JMP *+6
0322  5228 MREAD1 20D0FA  JSR MREAD       ;READ FROM MEMORY
0323  522B        AA      TAX             ;SAVE CHAR
0324  522C        A5EA    LDA STFLG       ;IN PASS2 ?
0325  522E        F008    BEQ CONT        ;YES, STAY IN LOOP
0326  5230        A523    LDA PSFLG       ;FLAG FOR 2ND PASS SET?
0327  5232        F00C    BEQ RET2        ;NO, JUST RETURN
0328  5234        C6EA    DEC STFLG       ;YES, RESET START FLAG...
0329  5236        F0DB    BEQ PS22-3      ;AND START PASS2
0330  5238 CONT   8A      TXA             ;RESTORE CHAR
0331  5239        C90D    CMP #$0D        ;IS IT CR?
0332  523B        D003    BNE RET2        ;NO, DO NOTHING
0333  523D        209951  JSR LINADD      ;INCREMENT LINE NUMBER
0334  5240 RET2   68      PLA             ;ADJUST STACK
0335  5241        68      PLA
0336  5242        8A      TXA             ;RESTORE CHAR
0337  5243        4C0AD1  JMP ASMCHR      ;JMP INTO CHAR INPUT LOOP
0338  5246
0339  5246 MS1    2020    .BYT '  LINAS  V2.1 '
0340  5254
0341  5254                .END
0341  5254         ERRORS= 0000
```

```
0000  0000        ;BEISPIEL FUER USER INPUT ANHAND DER NORMALEN
0001  0000        ;TAPE ROUTINE....
0002  0000
0003  0000 FNAM          =$E8A2
0004  0000 LOADTA        =$E32F
0005  0000 TIBYTE        =$ED3B
0006  0000
0007  0000        ; USER INPUT VECTOR : $16C=00
0008  0000        ;                     $16D=02
0009  0000
0010  0000              *=$200
0011  0200
0012  0200        B009    BCS USER        ;CARRY SET = INPUT ROUTINE
0013  0202 INIT   CA      DEX             ;WENN X=1 WAR, DANN FILENAME EINGEBEN
0014  0203        D003    BNE INIT1       ;NAECHSTEN FILE EINLESEN
0015  0205        20A2E8  JSR FNAM        ;FILE-NAME EINGEBEN
0016  0208 INIT1  4C2FE3  JMP LOADTA      ;TAPE FILE EINLESEN
0017  020B USER   4C3BED  JMP TIBYTE      ;NAECHSTEN CHAR EINLESEN
0018  020E
0019  020E              .END
0019  020E         ERRORS= 0000
```

Dipl.-Ing. Wolfgang Lange, 1000 Berlin 42

Strukturiert und schnell: PL/65

Von den drei im Jahr 1981 für den AIM 65 neu angekündigten Sprachen schien PL/65 für einige der Anwendungen des Verfassers besonders geeignet zu sein. Der vorliegende Artikel beruht auf den während einer halbjährigen - nicht ausschließlichen - Anwendung dieser Sprache gewonnenen Erfahrungen. Neben der Beschreibung der Sprache und der Bedienung des Compilers werden anhand eines Beispiels ihre Stärken und Schwächen diskutiert.

Der Lieferumfang von PL/65 besteht aus zwei 4 KB-ROMs für den Adreßbereich $B000-CFFF und einem etwa 135-seitigem Handbuch. PL/65 ist eine höhere, blockstrukturierte Programmiersprache mit Ähnlichkeiten zu PL/1 und ALGOL. Wie auch (2) entnommen werden kann, existieren bereits viele dieser, besonders für die Systemprogrammierung von Mikrorechnern geeignete Sprachen (PLMX, ZSPL, PLI-80, PL/M). Die für den AIM vorliegende Version ist eine aufwärtskompatible Untermenge von SYSTEM 65 PL/65. Das Übersetzungsprogramm für diese Sprache ist ein sog. Pre-Compiler, d.h. es erzeugt keinen lauffähigen Object Code, sondern symbolischen Assembler Source Code, der dann vom bekannten Assembler umgewandelt wird. Es sei hier bereits gesagt: nicht nur das Assembler-ROM ist für die Anwendung von PL/65 erforderlich, sondern auch einige Kenntnisse der Assemblersprache.

Sprachmerkmale in Stichworten: Blockstrukturen mit BEGIN ... END, Kontrollstrukturen mit IF ... THEN ... ELSE, FOR ... TO ... BY, WHILE und CASE.Variablentypen: BYTE, WORD, CHAR und eindimensionale Arrays mit max. 256 Elementen dieser Typen. Angabe von Konstanten im dualen, oktalen, sedezimalen oder dezimalen Zahlensystem zugelassen. Addition, Subtraktion, Move, Shift, Rotate und Vergleichs- und logische Operantoren können durch Angabe eines Quantifiers für Felder von 1-256 Byte verwendet werden. Einzelne Statements oder ganze Blöcke von Assemblercode können ins Programm eingesetzt werden. Stackoperationen, Interruptverarbeitung und CALL-Aufrufe werden voll unterstützt. Tabelle 1 enthält eine Liste der PL/65 Wörter mit kurzer Definition.

Tabelle 1: Definition der PL/65-Wörter

Keyword	Definition
.AND	Logical AND operator
BEGIN;	Start of BEGIN;-END; block
BRK	6502 assembler software instruction
BYTE	8-bit data declaration
BY	As in FOR-TO-BY
CALL	Call subroutine statement
CASE	Same as GOTO
CHAR	8-bit ASCII data declaration
DATA	8-bit data byte declaration
DATAW	16-bit data word array definition
DCL	Statement to declare variables
DEC	Decrement 8-bit byte statement
DECW	Decrement 16-bit word statement
DEF	Assigns values to variables
.DFILE	Disk file linkage statement
DO;	Start of DO;-END; block
ELSE	As in IF-THEN-ELSE
END;	End of BEGIN;-END; or DO;-END; block
ENTRY	Initialize values for program assembly
!EOR	Exclusive OR operator
EXIT;	Compiler termination statement
FOR	As in FOR-TO-BY

```
GOTO_       Unconditional Jump or computed GOTO
HALT;       Program termination statement
IF_         As in IF-THEN-ELSE
INC_        Increment 8-bit byte statement
INCW_       Increment 16-bit word statement
INIT_       Initialize program variables/data storage
.OR_        logical OR operator
RETURN;     Return from subroutine statement
ROL_        Rotate bites left statement
ROR_        Rotate bits right statement
RTI;        Return from interrupt statement
SHL_        Shift bits left statement
SHR_        Shift bits right statement
STACK_      Save values on the 6502 pushdown stack
!TFILE_     Tape file linkage statement
THEN_       As in IF-THEN-ELSE
TO_         As in FOR-TO-BY
WHILE_      While statement
WORD_       16-bit data declaration
UNSTACK     Retrieve values from the 6502 pushdown stack
```

Im folgenden soll auf Einzelheiten der Programmierung und Eigenarten der Sprache eingegangen werden. Als Programmbeispiel wurde eine allgemein verwendbare Routine zur Erzeugung von Großbuchstaben (BIG) unter Verwendung der DOTPATTERN-Tabelle des Monitor-ROMs ausgewählt und zu einem Hauptprogramm ergänzt.

Die i.a. gute Lesbarkeit strukturierter Programme beruht im wenstlichen auf drei Prinzipien: 1. Deklaration aller Konstanten, Variablen und Felder, 2. Top-Down-Design, d.h. schrittweises Verfeinern der Programmlogik in überschaubaren Einheiten und 3. Verdeutlichung der Struktur durch Blockbegrenzung (DO .. END) und entsprechende Druckaufbereitung. Eine sinnvolle Tabulation muß jedoch hier vom Programmierer vorgenommen werden, indem pro einzurückender Ebene ein 'I' vorangestellt wird, aus dem beim Auflisten vier Spaces werden.

Dem aufmerksamen Leser wird bei der kurzen Aufzählung der Sprachmerkmale das Fehlen von Ein-/Ausgabeanweisungen aufgefallen sein. Das ist hier leider richtig, es gibt kein PRINT, INPUT, GET etc., sondern nur die üblichen Monitorroutinen stehen zur Verfügung. Die Zeilen 42 - 48 zeigen, wie man das Einlesen einer Zeile mit der Möglichkeit des Löschens von Zeichen programmieren kann. Da der Akku und die anderen Register in PL/65 explizit nicht angesprochen werden können, muß hier für die erforderliche Abfrage jedes eingelesenen Zeichens ein Assemblerbefehl eingeschoben werden.

Es wurde bereits erwähnt, daß durch Subskribierung nur 256 Elemente adressiert werden können. Deshalb wird man bei längeren Tabellen, wie im Fall von DOTPAT mit 320 Byte, die indirekte Adressierung (wird durch das vor die Variable gesetzte '&' in Zeile 85 eingeschaltet) entsprechend (IND),Y wählen. Hierzu muß die Adresse einer Variablen in ein Wort in der Zero Page gebracht werden, wozu die Kennzeichnung '##' in Zeile 82 dient. Weiterhin sind für die schrittweise Adressierung in Tabellen, aber auch zum Zählen mit Wortbreite, die Befehle INCW und DECW sehr hilfreich.

Neben den Kontrollstrukturen ist das mächtigste Statement in PL/65 sicherlich das Assignment, mit dem für einzelne Bytes oder Felder Zuweisungen mit arithmetischen oder logischen Funktionen durchgeführt werden (s. die Zeilen 25 und 83 - 85 im Beispiel). Trotz der vielen Beispiele im Manual stieß der Verfasser mehrmals auf Zuweisungen, die seiner Meinung nach nicht im erwarteten Sinn aufgelöst wurden. Beim Aufspüren solcher, aber auch eigener logischer Fehler, helfen meist nur Assemblerkenntnisse weiter, um den vom Pre-Compiler erzeugten Zwischencode analysieren zu können. - Ein Beispiel: Die nachfolgenden Befehle führen nicht zum gleichen Ergebnis, wie man durch Analyse des erzeugten Codes leicht feststellen kann (FELD und TEXT sind Arrays, LEN ist ein Quantifier):

a)
```
;I=1; FELD.LEN=TEXT(I);
      LDA #1
      STA I
      LDY #0
      LDA LEN
      STA R0
      LDA I
      TAX
ZZ0003 LDA TEXT,Y
      STA FELD,Y
      INX
      INY
      DEC R0
      BNE ZZ0003
```

b)
```
;FELD.LEN=TEXT(1);
      LDY #0
      LDA LEN
      STA R0
ZZ0003 LDA 1+TEXT
      STA FELD,Y
      INX              (!?)
      INY
      DEC R0
      BNE ZZ0003
```

Auf diese Unterschiede der Verwendung von Konstanten bzw. Variablen bei der Subskribierung wird nirgends hingewiesen!

Wenn auch der Gebrauch des Assignment einer Vielzahl von Einschränkungen unterliegt und umfangreiche Berechnungen in mehrere Statements aufgelöst werden müssen (der Algorithmus der Zeilen 82 - 85 lautet z.B. in BASIC: OUT = DOTPAT(WERT + 64*SPALTE) AND 2↑ZEILE), ist die Lesbarkeit dennoch deutlich besser als in Assembler. Zur Beseitung eventueller Unklarheiten dienst oft auch ein Blick in die umfangreiche (48 Seiten) Auflistung von PL/65 Source Statements und den daraus erzeugten Assembler Codes. Hieraus lassen sich auch bei wirklich zeitkritischen Routinen leicht Anregungen für die Komprimierung und Beschleunigung des erzeugten Codes gewinnen. Der Verfasser rät jedoch von solcher Manipulation des Zwischencodes ab, da sie ja nach jeder neuen Übersetzung des Programmes erneut von Hand eingefügt werden müßte. Bei Echtzeitproblemen sollten die relevanten Routinen lieber separat in Assembler formuliert und der fertige Object Code dann vom PL/65-Programm aufgerufen werden.

Angesichts der beobachteten Geschwindigkeit der hier ausprobierten Benchmark-Programme ist das sicher selten nötig. Dazu folgende Zahlen: Ein Programm zur Berechnung aller Primzahlen zwischen 3 und 16384 nach dem Algorithmus 'Sieb des Eratosthenes' wurde auf dem AIM 65 in BASIC und PL/65 getestet, weiterhin liegen vom APPLE II Zeiten für mehrere Sprachen vor. Ergebnis: AIM 65 + PL/65 = 6,4 sec., APPLE II + UCSD PASCAL = 51,6 sec., APPLE II + Integer BASIC = 232 sec., AIM 65 + BASIC = 308 sec. (weitere s. (2)).

Nun zur Hantierung: Das PL/65-Quellprogramm wird im Editor erzeugt. Der Übersetzer bietet dann nach seinem Start zwei verschiedene Modi an (Rockwell nennt sie irreführenderweise PASS1 OR 2, obwohl es sich nicht um eine Compilierung mit zwei voneinander abhängigen Passes handelt): 1. Prüfung von Syntax und Struktur und Ausgabe eines formatierten Listings, 2. Prüfung wie 1) und Ausgabe des R6500 Assembler Codes, wobei die PL/65-Statements als Kommentare eingefügt werden.

Weiterhin kann der Übersetzungsprozeß wie auch beim Assembler mit IN= und OUT= mit verschiedenen Devices arbeiten. Der optimale Komfort ergibt sich natürlich, wenn zum Entwickeln und Testen die PL/65-Source, die Assembler-Source, die Symbol Table und das Objectprogram nebeneinander Platz im RAM haben. Unter Berücksichtigung des temporär vom Compiler benötigten Platzes von etwa 0 - $5F und $200 - $49F wird hierfür im Handbuch eine Memory Map für ein 4 KB-System vorgeschlagen, die jedoch nur für sehr kleine Programme ausreicht (das vorliegende Beispiel benötigt schon etwa 9 KB !). Das Handbuch enthält weiterhin ein sehr nützliches Programm 'Buffer Handler', das es erlaubt, zwei Editor-Buffer nebeneinander zu verwalten und außerdem einen User-Output für PASS2 so initialisiert, daß man mit IN=M und OUT=U die Ausgabe in den 2. Buffer erfolgt, der dann bei der folgenden Assemblierung mit IN=M angesprochen wird.

Dieses Programm wurde beim Verfasser so erweitert, daß bei PASS1 mit OUT=U die beim Beispiel verwendete Formatierungsroutine mit Kopfzeile, Seitenzähler und Statement-Numerierung angeschlossen werden kann. Bei kleinen Systemen wäre hier auch eine Erweiterung sinnvoll, die bei der

Ausgabe des Assembler Codes die Kommentare zwischen den Anführungszeichen unterdrückt, um so Platz zu sparen. Ein Schönheitsfehler des Compilers ist übrigens das Fehlen von Optionen für das Ein-/Ausschalten des Listings (wie .OPT LIS/NOL beim Assembler). Es gibt auch keine Angabe der Fehlerart, sondern unter der fehlerhaften Zeile wird eine Zeile mit '???...?ERROR' so positioniert, daß das letzte Fragezeichen auf den Beginn des fehlerhaften Wortes zeigt. Dies ist jedoch oft nicht der wirkliche Fehler, und manchmal, wie beim fehlenden 'END' eines Blockes oder fehlendem 'EXIT', bricht die Kompilierung auch mit einem Sprung auf einen BRK-Befehl ab. Die dabei für A, X, Y gezeigten Werte zwischen 0 und 2 werden nirgends erläutert. Fehlende, falsch geschriebene oder unzulässige Variablennamen können erst bei der Assemblierung 'angemeckert' werden, weil der Übersetzer selbst keine Symbol Table führt.

Der Verfasser hofft, mit diesem Beispiel und den allgemeinen Erläuterungen weitere Interessenten für die Sprache PL/65 und einen Gedankenaustausch in dieser Zeitschrift zu gewinnen. Manche häufig wiederkehrende Problemesind auch für PL/65 erst mit Erweiterungen elegant lösbar. Hier existiert bereits eine kleine Bibliothek von Assembler-Routinen für erweiterte Integer-Arithmetik, Zahlenwandlungen und Ein-/Ausgabe, mit denen diese Aufgaben gelöst werden. Nach Meinung des Verfassers wird diese Sprache besonders bei Programmen für Update und Retrieval größerer Datenmengen (z.B. Literaturstellen- oder Schallplattenarchiv), für Textver- und -bearbeitung und in der Systemprogrammierung wegen ihrer guten Lesbarkeit und Kompaktheit und Ausführungsgeschwindigkeit des Objecprogrammes viele Liebhaber finden.

Literaturhinweise:

(1) PL/65 User's Manual, Rockwell International Corporation 1980
(2) J. Gilbreath: A High-Level language Benchmark, BYTE 9/81

```
17.03.82  AIM65 - LAMON        *** FILE= PU-50 ***        SEITE  01

0001 ;"TEST-PROGRAM *UEBERSCHRIFT*";
0002 ;;
0003 ;"Einige Beispiele fuer die Programmierung in PL/65 .";
0004 ;"Das Programm gibt eingegebene Texte in Blockschrift";
0005 ;"auf dem Drucker aus.";
0006 ;;
0007 ;"Wolfgang Lange  15.03.82 / Version 1.0 ";
0008 ;;
0009 ;"=== VARIABLEN-VEREINBARUNGEN ===";
0010 ;;
0011 ;DEF OUTALL=$E9BC; DEF CRLF=$E9F0; DEF COMIN=$E1A1;
0012 ;DEF RDRUB=$E95F; DEF CR=$0D;
0013 ;DEF CURPO2=$A415; DEF DIBUFF=$A438;
0014 ;;
0015 ;DEF *=$115; "TEXT IM TAPE-BUFFER";
0016 ;DCL BUFLEN; DCL BUFFER(18);
0017 ;;
0018 ;"=== PROGRAMMSTART ===";
0019 ;ENTRY $200;
0020 ;BEGIN;
0021 ;F0=0; WHILE F0=0 BEGIN;
0022 ;      "EINGABEAUFFORDERUNG : * ";
0023 ;      OUT='*'; CALL OUTALL; CALL INLINE; OUT=1;
0024 ;      DEC CURPO2; "CR WURDE MITGEZAEHLT";
0025 ;      BUFFER.CURPO2=DIBUFF(OUT); "UEBERTRAGEN OHNE *";
0026 ;      BUFLEN=CURPO2 - 1; "ADRESSIERUNG AB BUFFER(0)";
0027 ;      IF BUFLEN>10 THEN BUFLEN=10; "WEGEN DRUCKBREITE";
0028 ;      CALL CRLF; " # SOLL ENDEBEDINGUNG SEIN";
0029 ;      IF BUFFER(0)='#' THEN F0=1;
0030 ;      ELSE              CALL    BIG;
0031 ;END;
```

```
0032 ;GOTO COMIN; "ZURUECK ZUM MONITOR";
0033 ;END;
0034 ;;
0035 ;"===== UNTERPROGRAMME =====";
0036 ;;
0037 ;"*** SUB ZEILE EINLESEN ***";
0038 ;;
0039 ;"TEXT VON TASTATUR IN DEN DIBUFF (MIT DELETE)";
0040 ;"EINGABE MIT CR ABSCHLIESSEN";
0041 ;;
0042 ;INLINE : BEGIN;
0043 ;      F1=0; WHILE F1=0 BEGIN;
0044 ;          CALL RDRUB; 'STA ZWI' ; "AKKU NUR IM ASSEMBLER";
0045 ;          IF ZWI=#CR THEN F1=1;
0046 ;      END;
0047 ;      RETURN;
0048 ;END;
0049 ;;
0050 ;"*** SUB GROSSBUCHSTABEN ***";
0051 ;;
0052 ;"DAS AUFRUFENDE PROG. INITIALISIERT DEN DRUCKER.";
0053 ;"JE NACH ZEICHENGROESSE (5, 10, 16.5 CHAR./LINE)";
0054 ;"DARF BUFLEN MAXIMAL DEN WERT 5, 11 ODER 18 ENT-";
0055 ;"HALTEN. DER ASCII-TEXT STEHT IM BUFFER.";
0056 ;"IN DIESER FORM NUR FUER AIM 65 DA DER ";
0057 ;"CHARACTER-GENERATOR DES MONITORS VERWENDET WIRD.";
0058 ;;
0059 ;"W.LANGE 12.03.82 / VERSION 2.0";
0060 ;;
0061 ;DEF BLANK=$20; DEF BLOCK=$FF; DEF ESC=$1B; "FUER OKI 80";
17.03.82  AIM65 - LAMON       *** FILE= PU-50 ***       SEITE  02


0062 ;DEF DOTPAT=$F2E1; "TABELLE IM MONITOR-ROM";
0063 ;DEF PATADR=$64 ; "FUER INDIREKTE ADRESSIERUNG";
0064 ;;
0065 ;BIG : BEGIN;
0066 ;OUT=#ESC; CALL OUTALL; OUT='8'; CALL OUTALL;CALL CRLF;
0067 ;"DRUCKER AUF 8 LPI EINGESTELLT (ESC,8 BEI OKI)";
0068 ;FOR ZEILE = 0 TO 6       DO;
0069 ;      FOR CHAR = 0 TO BUFLEN      DO;
0070 ;          CALL AUSGAB;
0071 ;      END;
0072 ;      CALL CRLF;
0073 ;END;
0074 ;OUT=#ESC; CALL OUTALL; OUT='6'; CALL OUTALL; CALL CRLF;
0075 ;RETURN;
0076 ;END;
0077 ;;
0078 ;"AUSGABE VON 5 SPALTEN PRO ZEICHENZEILE";
0079 ;AUSGAB: BEGIN;
0080 ;      WERT=BUFFER(CHAR) .AND $3F; "ASCII-WERT AUSRICHTEN";
0081 ;      FOR SPALTE = 0 TO 8 BY 2   DO;
0082 ;          PATADR = ##DOTPAT; "TABELLENANFANG";
0083 ;          PATADR.2 = PATADR + WERT; "ADRESSENRECHNUNG";
0084 ;          ZWI.2 = PROD64(SPALTE); PATADR.2 = PATADR + ZWI;
0085 ;          OUT=POT2(ZEILE); OUT=&PATADR .AND OUT;
0086 ;          "EINZELABFRAGE DER 7 BIT MIT AUFSTEIGENDER";
```

```
0087 ;          "WERTIGKEIT UND DRUCK ENTSPR. 0 ODER 1 .";
0088 ;          IF OUT = 0 THEN OUT=#BLANK;
0089 ;             ELSE    OUT=#BLOCK;
0090 ;          CALL OUTALL;
0091 ;       END;
0092 ;       OUT=#BLANK; CALL OUTALL; CALL OUTALL; "ZEICHENABSTAND
0093 ;RETURN;
0094 ;END;
0095 ;;
0096 ;"=== ARBEITSFELDER UND KONSTANTEN ===";
0097 ;;
0098 ;"POTENZEN VON 2 FUER BIT-ABFRAGE";
0099 ;POT2 : DATA 1,2,4,8,16,32,64;
0100 ;"PRODUKTE VON 64 FUER TABELLENADR.";
0101 ;PROD64 : DATAW 0,64,128,192,256;
0102 ;"ARBEITSFELDER :";
0103 ;DCL WERT WORD INIT(0); DCL ZWI WORD INIT(0);
0104 ;DCL CHAR, SPALTE, ZEILE, OUT, F0, F1, S2;
0105 ;;
0106 ;EXIT;
0107
0108
ERRORS= 00
```

Friedrich Geissler, 8120 Weilheim

Graphik-Plot (2)

Das Graphik-Plot-Programm in Heft 23, Seite 43, für den AIM 65/PC 100 wird mit den folgenden Assemblerroutinen komplettiert:

```
                ;-----------------------------
0F61              LOESCHEN
0F61
0F61            ;-----------------------------
0F61
0F61 LOE    A20D    LDX #13         ;DAS UNTERPROGRAMM LOESCHT
0F63        A063    LDY #99         ;DEN GESAMTEN BILDSPEICHER
0F65        A900    LDA #00         ;-- ES VERAENDERT A,X,Y
0F67        85A0    STA BYTE
0F69        A909    LDA #TABAN
0F6B        85A1    STA BYTE+1
0F6D F1     A900    LDA #00
0F6F        91A0    STA (BYTE),Y
0F71        88      DEY
0F72        C0FF    CPY #$FF
0F74        D0F7    BNE F1          ;WEITER LOESCHEN
0F76        A064    LDY #100
0F78        CA      DEX
0F79        F00E    BEQ ENDE        ;ENDE WENN X UND Y
0F7B        18      CLC             ; =0
0F7C        A964    LDA #100
0F7E        65A0    ADC BYTE
0F80        85A0    STA BYTE
0F82        9002    BCC F2
0F84        E6A1    INC BYTE+1
0F86 F2     4C6D0F  JMP F1          ;WEITER LOESCHEN
0F89
0F89 ENDE   60      RTS
```

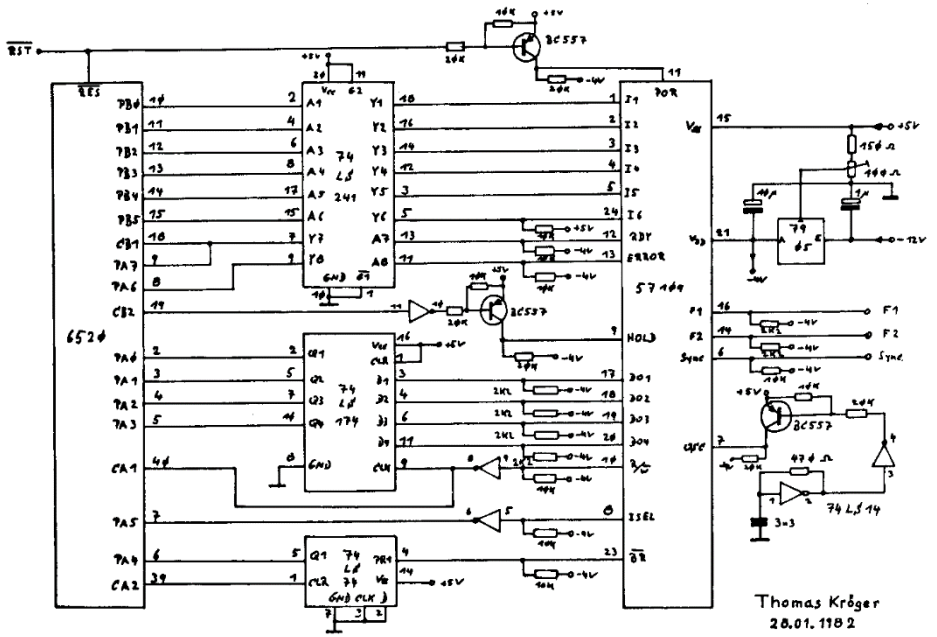
```
0F8A                SETZEN

0F8A SETZE    AD77A4 LDA IDOT        ;DAS UNTERPROGRAMM SETZT
0F8D          C963   CMP #99         ;EINEN PUNKT (X-Y)
0F8F          1013   BPL Q2
0F91          AD78A4 LDA IOUTL       ;UEBERPRUEFUNG
0F94          C963   CMP #99
0F96          100C   BPL Q2          ;ES MUSS X IN $A478=42104
0F98          201C0F JSR SET         ;UND Y1 IN $A477=42103
0F9B          AC78A4 LDY IOUTL       ;UEBERGEBEN WERDEN
0F9E          B1A0   LDA (BYTE),Y
0FA0          05A3   ORA YBIT        ;X,Y1 <= 99
0FA2          91A0   STA (BYTE),Y
0FA4 Q2       60     RTS

0FA5             LINIE LOESCHEN

0FA5          ;DAS UNTERPROGRAMM IST NUR FUER DIE
0FA5          ;AUSGABE VON MEHRDIMENSIONALEN FUNKTIONEN
0FA5          ;NOTWENDIG.(DAMIT DIE KURVE NICHT
0FA5          ;'DURCHSICHTIG' WIRD )
0FA5          ;EIN LOESCHEN VOM BASIC HER WAERE ZU LANGSAM.
0FA5
0FA5 LOET     AD78A4 LDA IOUTL       ;DAS UNTERPROGRAMM LOESCHT
0FA8          C963   CMP #99         ;EINE SENKRECHTE LINIE
0FAA          1046   BPL STOP        ;AN DER STELLE X
0FAC          AD77A4 LDA IDOT        ;VON Y1 BIS Y2(AUSSCHL.)
0FAF          C963   CMP #99
0FB1          103F   BPL STOP
0FB3          AD76A4 LDA IOFFS
0FB6          C963   CMP #99         ;UEBERPRUEFUNG
0FB8          1038   BPL STOP
0FBA          AD77A4 LDA IDOT
0FBD          CD76A4 CMP IOFFS       ;Y1 < Y2 ?
0FC0          300A   BMI Q3
0FC2          A8     TAY
0FC3          AD76A4 LDA IOFFS       ;UMORDNUNG
0FC6          8D77A4 STA IDOT
0FC9          8C76A4 STY IOFFS
0FCC Q3       CE76A4 DEC IOFFS
0FCF          EE77A4 INC IDOT
0FD2 Q4       AD77A4 LDA IDOT        ;UEBERGEBEN WERDEN MUSS :
0FD5          CD76A4 CMP IOFFS       ;X IN $A478 = 42104
0FD8          1018   BPL STOP        ;Y1 IN $A477 = 42103
0FDA          201C0F JSR SET         ;Y2 IN $A476 = 42102
0FDD          A5A3   LDA YBIT        ;X,Y1,Y2<=99
0FDF          49FF   EOR #$FF
0FE1          85A3   STA YBIT
0FE3          AC78A4 LDY IOUTL
0FE6          B1A0   LDA (BYTE),Y
0FE8          25A3   AND YBIT
0FEA          91A0   STA (BYTE),Y
0FEC          EE77A4 INC IDOT
0FEF          4CD20F JMP Q4
0FF2 STOP     60     RTS             ;ENDE    ;ENDE
```

#

Thomas Kröger, 2330 Eckernförde

6502 steuert Arithmetikprozessor

Die Firma National Semiconductor hat vor einigen Jahren den Number Oriented Processor MM 57 109 auf den Markt gebracht. Dieses MOS/LSI-IC hat folgenden Steckbrief: Dateneingabe in UPN (umgekehrt polnische Notation), 8-stellige Mantisse, 2-stelliger Exponent, Vier-Register-Stack (x,y,z,t), ein Memory-Register, trigonometrische Funktionen, logarithmische Funktionen, y^x, e^x, Pi, ERROR-Anzeige, zwei Flag-Ausgänge, Branch-Möglichkeiten.



Mit einer Handvoll von TTL-ICs läßt sich dieser Prozessor leicht an jeden Mikrocomputer anschliessen. Man könnte damit z.B. ein BASIC-Programm durch Entfernen der Rechenroutinen auf vielleicht 5-6 K abmagern und hätte trotzdem komfortablere Arithmetik-Funktionen. - Da das IC mit ca. DM 40,-- nicht gerade billig ist, sollte man beim Umgang damit größte Vorsicht walten lassen (der Autor übernimmt keinerlei Garantie für Hard- und Software, bei ihm hat es gleichwohl funktioniert!).

Hardware

Das Schaltbild zeigt den kompletten Aufbau zum Anschluß an eine 6520 PIA. Die Versorgungsspannung von 9 V erhält man, indem man den Massepunkt V_{DD} auf -4 V legt. Damit sind fast alle Ein- und Ausgänge LP-TTL ansteuerbar. Da die Ausgänge nur eine Low Power Lasteinheit treiben können, sind unbedingt LS-Bausteine vorzusehen! Die Eingänge POR, HOLD und OSC benötigen einen Spannungshub von 9 V und sind dadurch nicht TTL-kompatibel.

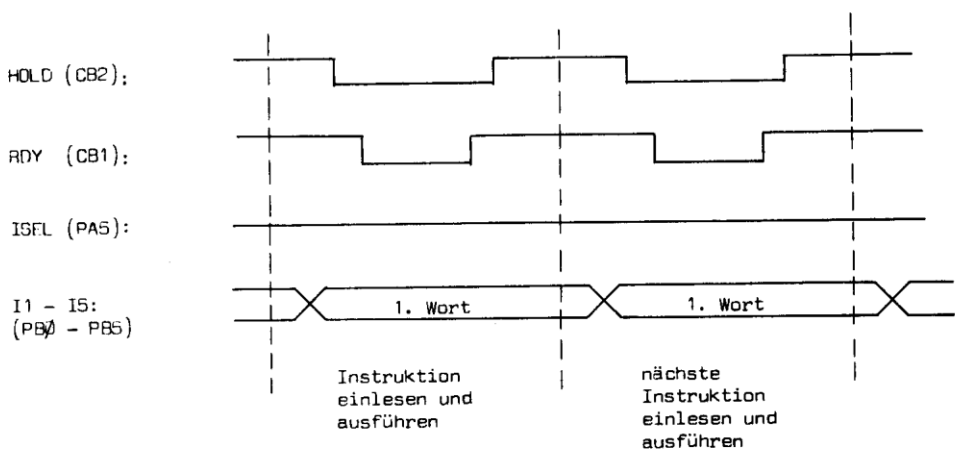
Am Ausgang SYNC steht die Oszillatorfrequenz durch 4 geteilt zur Verfügung. Bei f_{osz} = 360 kHz sind das 90 kHz, was einem Microcycle von 11 Mikrosekunden entspricht. Die Ausgänge F 1 und und F 2 sind durch Software steuerbar, sie lassen sich getrennt entweder pulsen oder setzen. Die Eingänge I1 bis I6 sowie die Ausgänge RDY, ERROR, R/W und ISEL werden über Puffer mit dem Port verbunden.

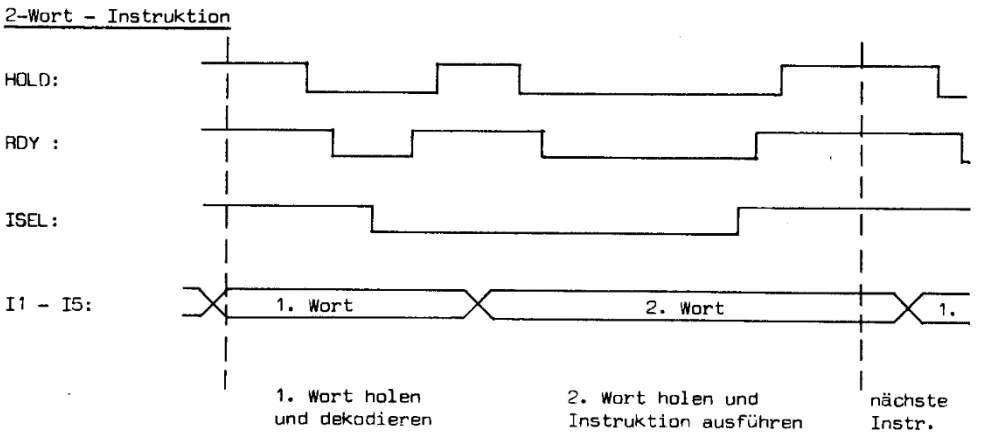
Die vier Dateneingänge DO1 bis DO4 sollten über ein latch zwischengespeichert werden. Das Einlesen erfolgt in diesem Falle mit der negativen Flanke des R/W-Signals. Die positive Flanke setzt das IRQ-Flag des 6520 und signalisiert die Gültigkeit der an Port PA0-PA3 anliegenden Daten. Der Zustand des BRANCH-Ausganges wird im 74LS74 zwischengespeichert und kann bei Bedarf über PA4 eingelesen und gleichzeitig zurückgesetzt werden.

Programmablauf im 57 109

Alle Dateneingaben zum 57 109 erfolgen über die Eingänge I1 - I6. Die Daten werden byteweise seriell und asynchron eingegeben. Es gibt im wesentlichen zwei Formate: 1- und 2-Wort-Instruktionen. Der Programmablauf für die Dateneingabe sieht folgendermaßen aus (6520/Port B im Handshake-Modus, CA2 im Puls-Modus):

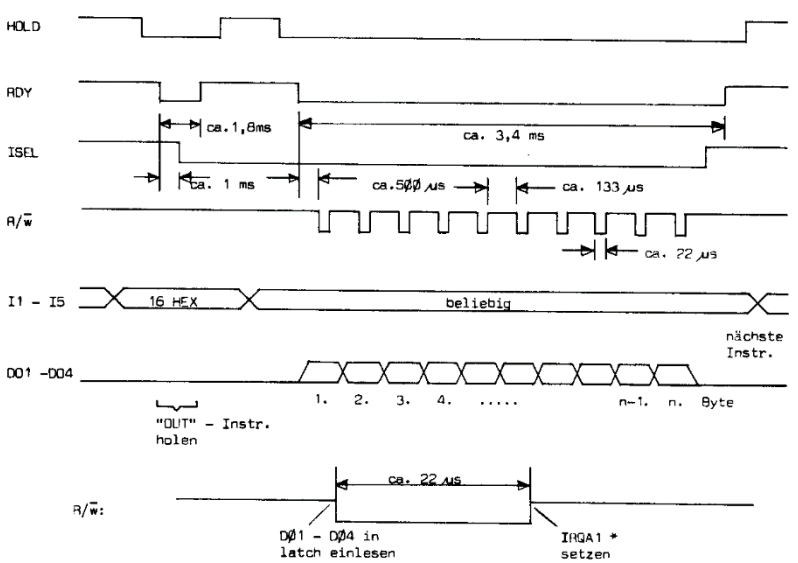
1-Wort - Instruktion





Datenausgabe

Die 'OUT'-Instruktion ist eine 2-Wort-Instruktion. Der Befehl OUT wird im ersten Wort mit hex 16 eingegeben. Das zweite Wort kann beliebig sein, z.B auch hex 16. Etwa 500 s nachdem RDY das zweite Mal auf '0' gegangen ist, beginnt die Datenausgabe. Das Programm braucht während dieser Zeit nur ab und zu das IRQ-Flag von CA1 abzufragen. Es hat dann 130 µs Zeit, die Daten zu lesen und abzuspeichern, bis das latch neue Daten einliest.



* R/W wird invertiert! Das heißt, der CA 1 des 6520 wird von einer negativen Flanke getriggert!

Daten werden in den folgenden zwei Formaten ein- und ausgegeben:

Fließkomma (FK)

1. Byte: DO4 enthält das Vorzeichen '1'=negativ, '0'=positiv, DO1-DO3='0'
2. Byte gibt die Position des Dezimalpunktes an, z.B hex 0B= hinter dem 3. Byte
   hex 04= hinter dem 10. Byte
3. Byte = höchstwertige Mantissenziffer (linke Ziffer) usw. bis 10. Byte = rechte Ziffer.

Exponentialschreibweise (SN)

1. Byte = Höchstwertige Exponentialstelle (linke Ziffer)
2. Byte = Letztwertige Exponentialstelle (rechte Ziffer)
3. Byte = Vorzeichen Exponent (Darstellung wie oben)
4. Byte ist permanent hex 0B, da das Komma immer an der gleichen Stelle steht
5. Byte = Höchstwertige Mantissenstelle (linke Ziffer), die immer ungleich Null ist
   usw. bis 12. Byte = Letztwertige Mantissenziffer (rechte Ziffer).

Die folgende Software kann zwischen der Notierung FK und SN (Exponential-, wissenschaftliche Schreibweise) unterscheiden, indem sie das 4. Byte untersucht. Ist es gleich hex 0B, so ist die Ausgabe im SN-Modus, ist es kleiner/gleich hex 09, so ist die Ausgabe im FK-Modus. Die 10 Byte im FK-Mode oder die 12 Byte im SN-Mode können dann für eine Anzeige oder die Weiterverarbeitung in das richtige Format gebracht werden.

Auf eine tabellarische Auflistung der zahlreichen Befehle des 57 109 und der Ausführungszeiten wird hier verzichtet. Der Leser findet sie im Datenblatt für den Chip. Zum Beispiel sei erwähnt, daß eine Sinusberechnung zwischen 0,62 bis 1,06 Sekunden dauern kann wenn die o.a. Taktfrequenz benutzt wird.

Software

Die folgende Software wurde zur Verdeutlichung der Abläufe sehr übersichtlich und einfach gestaltet. Es wurde auf jegliche Programmiertricks verzichtet. Der Anwender kann sich auf Grund der Beispiele leicht eigene Programme entwickeln. Besondere Aufmerksamkeit verdienen die maschinenspezifischen Unterprogramme:

CRLF	Zeilenrücklauf und Zeilenvorschub ausgeben
PRTBYT	Ein Hex-Zeichen als 2 ASCII-Zeichen ausgeben, z.B. '0B' als 30, 42
OUTSP	Ein SPACE ausgeben
GETBYT	2 ASCII-Zeichen in eines packen und im Akku bei RTS übergeben

Zur Vereinfachung wurde folgende Einteilung vorgenommen:

Normale Instruktionen: 00 - 3F hex
Inverse Instruktionen (2-Wort): 40 - 7F hex
OUT-Instruktionen (2-Wort): 80 - BF hex
BRANCH-Instruktionen (2-Wort): C0 - FF hex

Addition: 15 + 7 = 22 Berechnungsbeispiel

Eingabe: (HEX)

1.	2F	Master Clear
2.	Ø1	1
3.	Ø5	5
4.	21	ENTER
5.	Ø7	7
6.	39	+
	Das Ergebnis (22) steht jetzt im x-Register.	
7.	8Ø	"OUT" Ergebnis ⟶ 22

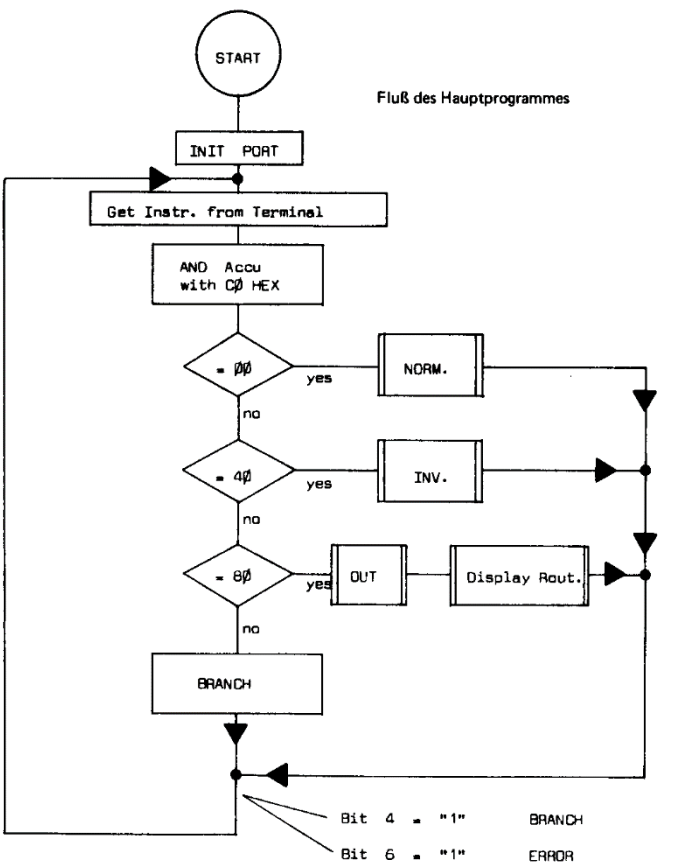
Berechnung eines Sinus: $\sin \frac{\pi}{3} = Ø,866$ Berechnungsbeispiel

1. ØD π
2. 21 ENTER
3. Ø3 3
4. 3C $\div$

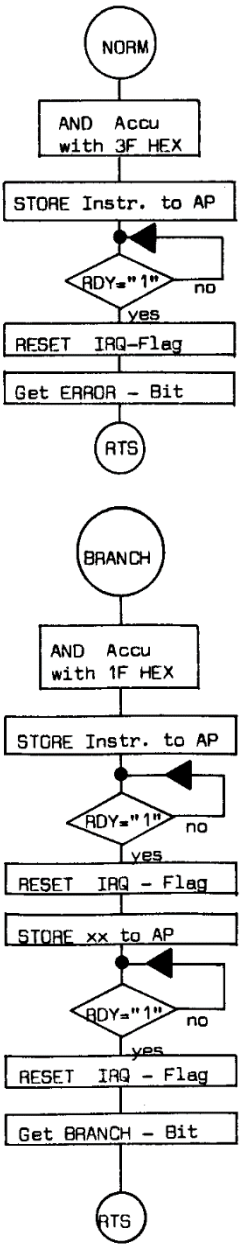
   Es steht 1,Ø4719 im x-Register.
5. 2C Umwandlung in Grad

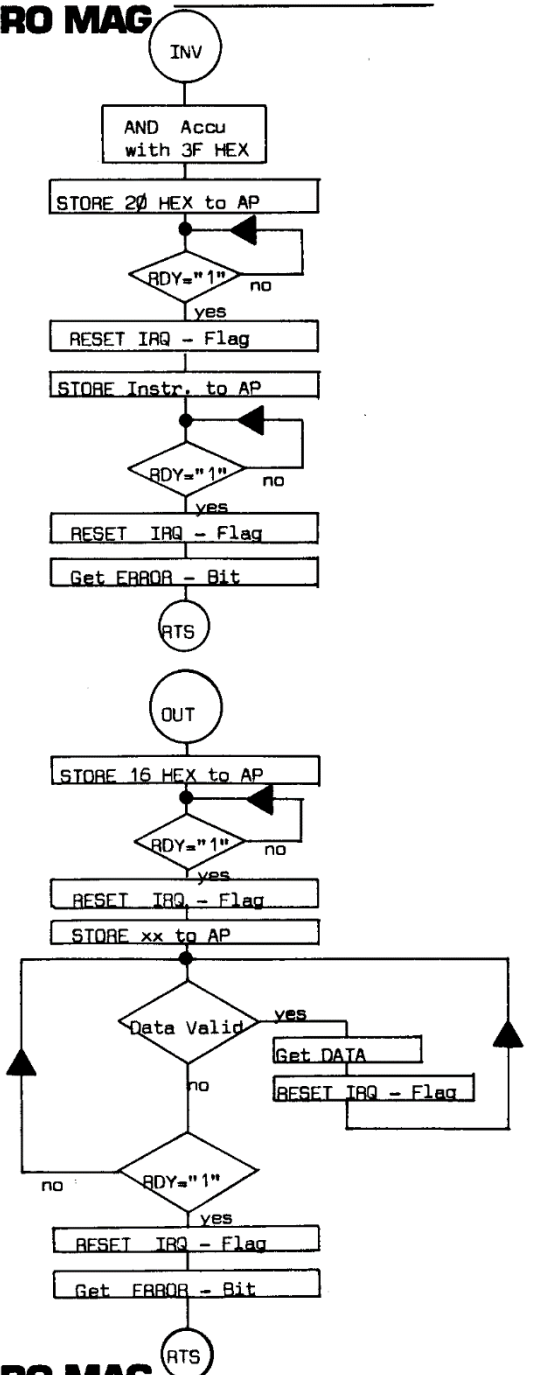
   Es steht 6Ø im x-Register.
6. 24 sin x (x in Grad!)

   Es steht Ø,866 im x-Register
7. 8Ø "OUT" Ergebnis ⟶ Ø,866Ø254Ø



Fluß des Hauptprogrammes





Fluß der Unterprogramme

```
0010:                   main routine
0020:
0030: 0300                      org   $0300
0040:
0050: 0300              cnt     =     $00e0
0060: 0300              buffer  =     $0100
0070: 0300              crlf    =     $1e2f   print cr / lf
0080: 0300              prtbyt  =     $1e3b   print (a) as 2 ascii
0090: 0300              outsp   =     $1e9e   print 1 space
0100: 0300              getbyt  =     $1f9d   2 ascii into a
0110: 0300              norm    =     $0200
0120: 0300              inv     =     $0213
0130: 0300              branch  =     $0234
0140: 0300              out     =     $0253
0150: 0300              init    =     $0283
0160:
0170: 0300 20 83 02     start   jsr   init
0180: 0303 20 9e 1e     next    jsr   outsp
0190: 0306 20 9d 1f             jsr   getbyt
0200: 0309 aa                   tax
0210: 030a 29 c0                andim $c0     mask
0220: 030c a8                   tay
0230: 030d 8a                   txa           instr. in a
0240: 030e c0 00                cpyim $00     norm ?
0250: 0310 f0 14                beq   inorm
0260: 0312 c0 40                cpyim $40      inv ?
0270: 0314 f0 16                beq   iinv
0280: 0316 c0 80                cpyim $80     out ?
0290: 0318 f0 18                beq   iout
0300: 031a 20 34 02             jsr   branch
0310: 031d 20 3b 1e             jsr   prtbyt
0320: 0320 20 9e 1e             jsr   outsp
0330: 0323 4c 03 03             jmp   next
0340: 0326 20 00 02     inorm   jsr   norm
0350: 0329 4c 03 03             jmp   next
0360: 032c 20 13 02     iinv    jsr   inv
0370: 032f 4c 03 03             jmp   next
0380: 0332 20 53 02     iout    jsr   out
0390:
0400: 0335 20 2f 1e             jsr   crlf    display rout.
0410: 0338 a9 00                ldaim $00
0420: 033a 85 e0                sta   cnt
0430: 033c a6 e0        load    ldx   cnt
0440: 033e bd c0 01             ldax  buffer
0450: 0341 20 3b 1e             jsr   prtbyt
0460: 0344 20 9e 1e             jsr   outsp
0470: 0347 e8                   inx
0480: 0348 86 e0                stx   cnt
0490: 034a e0 0c                cpxim $0c
0500: 034c d0 ee                bne   load
0510: 034e 20 2f 1e             jsr   crlf
0520: 0351 4c 03 03             jmp   next

0100: 0200                      org   $0200
0110:
0120: 0200              buffer  =     $0100   input buffer
0130: 0200              pad     =     $1610   port a data register
```

```
0140: 0200              pac     =       $1611   port a control reg.
0150: 0200              pbd     =       $1612   port b data register
0160: 0200              pbc     =       $1613   port b control reg.

0230:                   subroutine  -  normal instructions
0240:                   ( instr. in a )
0250:
0260: 0200 29 3f        norm    andim $3f       mask instr.
0270: 0202 8d 12 16             sta   pbd       store to ap
0280: 0205 2c 13 16     waita   bit   pbc       wait on rdy = 1
0290: 0208 10 fb                bpl   waita
0300: 020a ad 12 16             lda   pbd       reset irq - flag
0310: 020d ad 10 16             lda   pad       get error bit
0320: 0210 29 40                andim $40       1 = error
0330: 0212 60                   rts
0340:
0350:
0360:                   subroutine  -  inverse instructions
0370:                   ( instr.in a )
0380:
0390: 0213 29 3f        inv     andim $3f       mask instr.
0400: 0215 a2 20                ldxim $20       store inv to ap
0410: 0217 8e 12 16             stx   pbd
0420: 021a 2c 13 16     waitb   bit   pbc       wait on rdy = 1
0430: 021d 10 fb                bpl   waitb
0440: 021f ae 12 16             ldx   pbd       reset irq - flag
0450: 0222 8d 12 16             sta   pbd       store instr.
0460: 0225 2c 13 16     waitc   bit   pbc       wait on rdy = 1
0470: 0228 10 fb                bpl   waitc
0480: 022a ad 12 16             lda   pbd       reset irq - flag
0490: 022d ad 10 16             lda   pad       get error bit 6
0500: 0230 29 40                andim $40       1 = error
0510: 0232 60                   rts
0520: 0233 ea                   nop

0570:                   subroutines  -  branch instructions
0580:                   ( instr. in a )
0590:
0600: 0234 29 1f        branch  andim $1f       mask instr.
0610: 0236 8d 12 16             sta   pbd       store to ap
0620: 0239 2c 13 16     waitd   bit   pbc       wait on rdy = 1
0630: 023c 10 fb                bpl   waitd
0640: 023e ad 12 16             lda   pbd       reset irq - flag
0650: 0241 8d 12 16             sta   pbd       store xx to ap
0660: 0244 2c 13 16     waite   bit   pbc       wait on rdy = 1
0670: 0247 10 fb                bpl   waite
0680: 0249 ad 12 16             lda   pbd       reset irq - flag
0690: 024c ad 10 16             lda   pad       get branch bit 4
0700: 024f 29 10                andim $10       1 = branch
0710: 0251 60                   rts
0720: 0252 ea                   nop
0730:
0740:
0750:                   subroutine - output
0760:
0770: 0253 a9 16        outda   ldaim $16
0780: 0255 8d 12 16             sta   pbd       store out to ap
0790: 0258 2c 13 16     waitf   bit   pbc       wait on rdy = 1
```

```
0800: 025b 10 fb          bpl   waitf
0810: 025d ad 12 16       lda   pbd     reset irq - flag
0820: 0260 8d 12 16       sta   pbd     store xx
0830: 0263 a2 00          ldxim $00
0840: 0265 2c 11 16  getda bit   pac     data valid ?
0850: 0268 30 0e          bmi   data
0860: 026a 2c 13 16       bit   pbc     ready ?
0870: 026d 10 f6          bpl   getda
0880: 026f ad 12 16       lda   pbd     reset irq - flag
0890: 0272 ad 10 16       lda   pad     get error bit 6
0900: 0275 29 40          andim $40     1 = error
0910: 0277 60             rts
0920:
0930: 0278 ad 10 16  data lda   pad     get data
0940: 027b 29 5f          andim $5f     mask
0950: 027d 9d 00 01       stax  buffer  store data
0960: 0280 e8             inx
0970: 0281 d0 e2          bne   getda   next byte
0980:
0990:
1000:                subroutine -  init port
1010:
1020: 0283 a9 00     init ldaim $00
1030: 0285 8d 11 16       sta   pac
1040: 0288 8d 13 16       sta   pbc
1050: 028b 8d 10 16       sta   pad
1060: 028e a9 ff          ldaim $ff
1070: 0290 8d 12 16       sta   pbd
1080: 0293 a9 2c          ldaim $2c
1090: 0295 8d 11 16       sta   pac
1100: 0298 a9 26          ldaim $26
1110: 029a 8d 13 16       sta   pbc
1120: 029d 60             rts
```

#

CBM: Abschaltung des Interrupts

Beim CBM und PET wird bekanntlich 60mal in der Sekunde ein Interrupt erzeugt, um eine Tastaturabfrage vorzunehmen. Die Zeiten außerhalb des Interrupts stehen für die Abarbeitung von Anwenderprogrammen zur Verfügung. In Meßanordnungen nun kommen Daten oder Signalwechsel oft so schnell herein, daß die von der Interruptbearbeitung beanspruchte Zeit eine ausreichend schnelle Reaktion des Computers verhindert. In solchen Fällen möchte man alle Verzögerungen durch unnötige Arbeiten des Computers ausschließen und sich voll auf die Meß- oder Steueraufgabe konzentrieren. Man muß den normalen Interrupt ausschließen. Wie geschieht das?

Die im CBM benutzten Interfacebausteine 6520 (PIA) und 6522 (VIA) können an den Kontroll-Pins CA1, CA2, CB1 und CB2 nach entsprechender Initialisierung per Programm abfallende oder aufsteigende elektrische Impulsflanken erkennen und in Abhängigkeit davon an der CPU 6502 einen IRQ-Interrupt auslösen.

Beim CBM ist der Pin CB1 der unter Adresse hex E810-E813 erreichbaren PIA auf die Erkennung fallender Impulsflanken programmiert. Die 60mal pro Sekunde ankommenden Flanken werden vom Mutterquarz des Computers über unterteilende TTL-Schaltkreise an diesen Pin herangeführt. und außerdem ist im zugehörigen Kontrollregister der PIA unter E813 programmiert, daß eine solche Impulsflanke einen Interrupt auslösen darf: Wenn wir mit dem TIM-Monitor diese Speicher-

zelle aufschlagen, so finden wir in ihr hex 3D, d.h. das niederwertigste Bit ist zu '1' gesetzt, gleichbedeutend mit 'Interrupt Enable für CB1' (siehe Hardware-Handbuch zur PIA).

Wenn wir dieses niederwertigste Bit 0 unter Adresse E813 zu '0' setzen können, dann ist das Interrupt Enable ausgeschaltet, obwohl noch immer die elektrischen Impulse am Pin CB1 ankommen. Die Maschine ist also für ein von uns bestimmbares zeitliches Intervall der intensiven Reaktion oder des intensiven Rechnens von allen unnötigen Tätigkeiten befreit. Dazu programmieren wir in Maschinensprache:

```
LDA  $E813   Kontrollregister der PIA, Seite B laden
AND  #$FE    Bit 0 abmaskieren
STA  $E813   ins Kontrollregister zurückschreiben
RTS          Ende des Unterprogrammes
```

und rufen dieses Programm mit dem SYS-Befehl auf. Hernach ist der Interrupt und damit die Tastaturabfrage abgeschaltet. An den Schluß eines solchen Anwenderprogrammes gehört daher ein SYS-Befehl, der den Vorgang wieder umkehrt. Dazu wird folgendes Programm angesprungen:

```
LDA  $E813   Kontrollregister laden
ORA  #$1     Bit 0 hinzu-ODERn
STA  $E813   zurückschreiben
RTS
```

Das Gleiche können wir mit einigen Befehlen unter BASIC erreichen:

```
10    A=PEEK(59411)   Kontrollregisterinhalt holen und mit AND maskieren
20    B=A AND254
30    POKE59411,B     Interrupt Enable ausschalten
.....                 hier steht das Anwenderprogramm
1000  POKE59411,A     Wiedereinschalten des Interrupts.
```

Wenn wir zu Studienzwecken ein Anwenderprogramm in Betrieb nehmen, das eine größere Arbeit für den Computer bedeutet, z.B. das Füllen und Bearbeiten eines Arrays, so werden wir feststellen, daß es bei abgeschaltetem Interrupt um etwa 7% schneller ausführt. Aus den gemessenen Laufzeiten und unter Berücksichtigung der Tatsache, daß der Interrupt ohne Abschaltung 60mal in der Sekunde eintritt, können wir dann zurückrechnen, daß jede Interruptbearbeitung etwa 1,1 Millisekunden beansprucht. In dieser zeitlichen Größenordnung liegt also die normale äußerste Reaktionsgeschwindigkeit des Rechners. Wenn wir jedoch den Interrupt ausschalten und uns maschinensprachlicher Programmierung bedienen, können wir je nach Meßanordnung und Verarbeitungsaufgabe Reaktionsgeschwindigkeiten im Bereich von 20 bis 50 Mikrosekunden erreichen! R.L. #

Dirk Sanders, 6100 Darmstadt

Low Cost Typenraddrucker

für AIM 65 (6502)

Die seit einiger Zeit von Olivetti und Quelle angebotenen Typenrad-Schreibmaschinen (Preis ca. 1.100 DM) können mit diesem Programm als Ausgabeeinheit am AIM 65 benutzt werden. Selbstverständlich können sie auch an andere Systeme angepaßt werden. Eine Typenrad-Maschine hat gegenüber einem Matrixdrucker den Vorteil der sauberen Schriftausgabe und läßt sich darüber hinaus auch als gute 'normale' Schreibmaschine benutzen. Dieser Vorteil wird allerdings mit einer langsameren Ausgabegeschwindigkeit erkauft.

A) Eigenschaften der hier beschriebenen Ansteuerung

```
* Anbindung an  Assembler:  OUT=U
                BASIC:      SAVE OUT=U
                -Variable:  SYS, dann mit PRINT oder X=USR(Y) zum Initialisieren,
                            mit POKE OUTFLF,U umschalten, dann mit PRINT
```

Monitor: DILINK umschalten
Programm: JSR ...

* 14 Verbindungsleitungen notwendig (keine Elektronik)
* Druck in schnellstmöglicher Geschwindigkeit
* Daraus folgt eine Einschränkung: Nach einem CR über volle DIN A4-Breite dürfen höchstens 10 SPACE folgen (s.u.)
* Drucker-Tastatur bleibt weiterhin benutzbar
* Mit einem 'Seitentrenner' kann der Ausdruck vom übertragenen Text her unterbrochen werden (ASCII $40 = Klammeraffe), Fortsetzung des Druckes nach Tastendruck auf der AIM-Tastatur.
* Ansteuerbare Maschinen: Olivetti Praxis 35
  Olivetti Praxis 30
  Quelle, Bestell-Nr. 027.304

(Die Elektronik ist bei allen Typen genau gleich.)

B) Elektronik im Drucker

Die Schreibmaschine enthält zwei Ein-Chip-CPU's von MOSTEK: Den 3870 mit 2K/8 ROM und 64 Byte RAM (Slave), der die Typenradpositionierung besorgt und den 3872 mit 4K/8 ROM + 128 Byte RAM (Master), der alles übrige kontrolliert. Auf der Platine unter der Tastatur finden sich weiterhin einige diskrete Bauelemente.

C) Kurzbeschreibung der Maschinenfunktionen

Die Tastatur wird vom Master abgefragt. Wenn eine Taste gedrückt ist, wird das Typenrad vom Positionierungs-Motor ausgerichtet (der Slave kontrolliert dabei Position, Richtung und Geschwindigkeit). Der Servive-Motor läuft an, der Druckmagnet kuppelt ihn an den Farbbandtransport und an den Druckhammer. Der Druckhammer schlägt jetzt die Type gegen Farbband und Papier, der eigentliche Druck ist damit beendet. Der Step-Motor bewegt jetzt noch die Druckeinheit auf die nächste Buchstabenposition. Wenn jetzt noch ein Zeichen im Schreibmaschinenspeicher ist (max. 12 Zeichen), so dreht das Typenrad sofort auf die neue Position, im anderen Fall in eine Grundstellung. Durch den Speicher ist die Eingabe in kleinen Grenzen unabhängig von der Ausgabe.

D) Erläuterung der Ansteuerlogik

Der AIM steuert den Drucker über die Tastatur-Matrixleitungen und ahmt damit Tastendrücke auf der Schreibmaschine nach. Um die Tasten zur rechten Zeit zu 'drücken', werden Steuersignale von der Maschine abgenommen.

1. Zugriff auf die Drucker-Tastatur

Leitungen: LINE0 bis 7, TIME0 und TAKT. Die Tastatur funktioniert ähnlich wie die des AIM (siehe Anwenderhandbuch, Kap. 7.2.8) und besteht aus einer Matrix von 2 mal 8 Leitungen, TIME und LINE. Der 3872 shiftet einen LOW-Impuls durch die TIME-Leitungen und empfängt ihn bei gedrückter Taste an einer der LINE-Leitungen. Um nicht alle 16 Port-Pins der AIM-VIA 6522 auf diese Matrix zu verschwenden, erhält der AIM den (unregelmäßigen) Takt, mit dem an der Schreibmaschine die TIME-Leitungen nacheinander LOW werden, und zwar von Pin 25 des 3872. Dabei wird der LOW-Impuls in der ersten TIME-Leitung (0) als Startimpuls ausgewertet. Zusammen mit dem Shift-Takt kann der AIM dann bestimmen, auf welcher TIME-Leitung sich der LOW-Impuls gerade befindet und dann im richtigen Moment eine der LINE-Leitungen auf LOW ziehen.

2. Optimale Geschwindigkeit

Leitung: DRUCK. 'Schreibt' der AIM den zweiten Buchstaben schon, während der erste noch gedruckt wird, so dreht das Typenrad nicht in die Grundstellung zurück, sondern stellt sofort auf den nächsten Buchstaben ein. Andererseits darf der AIM nicht zu schnell schreiben, sonst gehen Buchstaben verloren. Die Leitung DRUCK vom Druckmagneten (s.o.) meldet dem AIM den Ausdruck eines Zeichens. Damit schreibt der AIM geschwindigkeitsangepaßt. Da jedoch bei Simulierung der Funktionstasten und bei SPACE der Druckmagnet nicht betätigt wird, erfolgt in diesen Fällen keine Rückmeldung. Der AIM ist hier so programmiert, daß er die Zeit für einen SPACE abwartet. Deshalb ist eine gewisse Vorsicht bei langdauernden Funktionstasten angebracht: Unter den näch-

sten 10 Zeichen sollte ein druckbares sein. Auf ein 'CR', das einen Wagenrücklauf über die volle Breite einer DIN A4-Seite bewirkt, sollten maximal 10 Spaces folgen, auf ein 'CR' über eine halbe Seitenbreite höchstens 30 Spaces, da der Drucker sonst nicht nachkommt. Bei größeren Einrückungen benutze man also einen Tabulatorsprung oder drucke einen Punkt '.' in eine der ersten Schreibpositionen in der neuen Zeile. Innerhalb einer Zeile sind beliebig viele Spaces möglich.

3. Keyboard- und Shiftumschaltung

Leitungen: KB und Shift. Der KB-Schalter der Maschine ist auf II zu stellen, sonst wird PA7 gewaltsam auf Masse gelegt und kann nicht umschalten.

E) Anmerkungen

Seitentrenner: Da ASCII $40 in BASIC eine Zeile löscht, diesen mit PRINT CHR$(64) ausgeben. Im Quellprogramm nach einem Semikolon einsetzen.

Umlaute: ä = F1, ö = F2, ü = F3, ß liegt auf dem Doppelkreuz.

Mögliche Erweiterungen: Da alle Verbindungen zur Druckertastatur vorhanden sind, müßte diese auch als Eingang für AIM 65 verwendbar sein. Der AIM zählt dann den Takt mit und prüft, welche der LINE-Leitungen durch Tastendruck LOW wird. Aus TIME- und LINE-Wert könnte dann die betätigte Taste bestimmt werden.

Der Step-Motor wird mit vier Leitungen bedient: 3872 Pin 3, 4, 5, 6. Zur Steuerung wird ein kompliziertes Impulsschema verwendet. Wenn ein Leser dieses durchdringen könnte, dann wäre ein perfekter Randausgleich möglich und auch ein Rückwärtsdrucken, indem der AIM den Step- und Zeilenvorschubmotor direkt bedient. Der Step-Motor muß wegen der vorhandenen Möglichkeit, die Schreibbreite von 10 auf 12 oder 15 Buchstaben je Zoll (pitch) umzuschalten, in ganz kleinen Schritten steuerbar sein (zwei Umschaltbrücken links oben auf der Tastaturplatine. Eine Umschaltung wird erst nach 'CR' registriert). Für diese Erweiterung werden vermutlich sechs weitere Verbindungskabel erforderlich sein, die man von Anfang an schon mit vorsehen sollte.

Die Tastaturmatrix der Schreibmaschine ist durch folgende Belegungen gekennzeichnet:

```
 TIME:     0     1     2     3     4     5     6     7
-------------------------------------------------------
L  0 |     *     l     2     3     4     6     5   TAB
I  1 |   REL     ä     ö     l     k     j     h   (-)
N  2 |  LÖSCH    -     .     ,     m     n     b   HALB
E  3 |     *     q     w     e     r     t     7   MARGL
.. 4 |     *     a     s     d     f     z   ***   MARGR
   5 |  LOCK     ü     p     o     i     u    BS   TABSET
   6 |   REP     '     ß     0     9     8    LF    (--
   7 |     *     y     x     c     v     g SPACE TABCL
-------------------------------------------------------
     -                                  *  = keine Funktion
     -                                ***  = blokiert alles


-  Bit     :   7     6     5     4     3     2     1     0
   steuert:  KB     TIME.......   SHIFT  LINE........
   LOW     =  KB I                 ON
   HIGH    =  KB II                OFF


-  Beispiel :  "*" ist auf      76543210
      -        Taste   "9"      100 110   TIME = 4, LINE = 6
      -        auf KB II       1
      -        mit  Shift            0
      -        ================ 11000110   = $C6
```

Folgende Belegungen der Hex-Bytes weichen von der ASCII-Norm ab:

$04 = TABCL	$06 = REPEAT	$08 = Backspace
$09 = TAB	$0A = (---	$14 = TABSET
$16 = |	$17 = _	$19 = (---
$23 = ß	$3C = ¨	$3E = q
$5B = Ä	$5C = §	$5D = Ö
$5E = Ü	$5F = _	
$7B = ä	$7C = |	
$7D = ö	$7E = ü	$80 = ^
$81 = (-|-)	$82 = HALB	$83 = MARGL
$84 = MARGR	$85 = °	$86 = ²
$87 = ³	$88 = löschen	$89 = REL
$8A = (---	$8B = `	$8C = ´

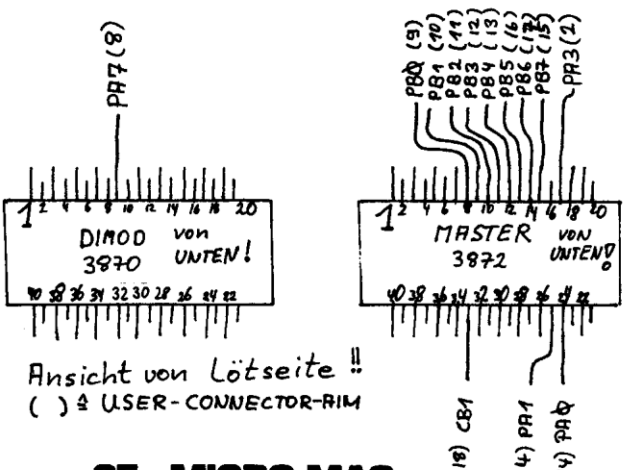
F) Hinweise für den Umbau

Demontage: 4 Schrauben sind am Boden der Maschine zu lösen. Man schraube beide Walzenköpfe ab (normales Rechtsgewinde), dann wird der linke Arm des Klappdeckels aus dem Gelenk gebogen. An der Tastatur sind 2 Schrauben zu entfernen, auf der Platine sind 3 Muttern zu öffnen. Man klappt die Tastatur auf, womit die Lötstellen zugänglich sind.

```
- Verbindung Drucker - Aim
   PB0 - PB7 an LINE0 - LINE7 = 3872 (MASTER): PIN  8 - 15
   also PB   0  1  2  3  4  5  6  7
   an  PIN   8  9 10 11 12 13 14 15
   PA0       an DRUCK          = 3872 (MASTER): PIN 24
   PA1       an TAKT           = 3872 (MASTER): PIN 25
   PA3       an SHIFT          = 3872 (MASTER): PIN 17
   PA7       an KB             = 3870 (SLAVE ): PIN  9
   CB1       an TIME0          = 3872 (MASTER): PIN 33
   MASSE     an MASSE          = 3872 (MASTER): PIN 20
```

Verbindung Schreibmaschine mit dem AIM

Das benutzte Kabel ist irgendwo herauszuführen. Oder: Man führt ein 14- (20-)poliges Flachkabel von der Tastaturplatine rechts zu einer Buchse, die über dem Trafo eingebaut wird. - Alle Angaben ohne Gewähr!



```
0000          ;STEUERPROGRAMM FUER TYPENRAD-DRUCKER
0000
0000          ;TYP:    "OLIVETTI PRAXIS 35"
0000          ;UND     "OLIVETTI PRAXIS 30"
0000          ;AUCH    "QUELLE BEST NR. 027.304"
0000
0000          ;              3/1982    D. SANDERS
0000
0000                    *=$0E3F
0E3F
0E3F TIME1              =25000          ;MINDESTZEIT FUER SPACE
0E3F TIME2              =1000           ;EINSCHALTDAUER: LINE
0E3F
0E3F OUTPUT             =$E97A
0E3F PHXY               =$EB9E
0E3F PLXY               =$EBAC
0E3F READ               =$E93C
0E3F OUTDIS             =$EF05
0E3F
0E3F UOUT               =$10A
0E3F BUOUT              =$DFDC          ;STANDARD UOUT DER BASIC-
0E3F                                    ;ERWEITERUNG (SYSTEM
0E3F                                    ;ABHAENGIG)
0E3F DILINK             =$A406
0E3F OUTFLG             =$A413
0E3F LINFO              =$E7            ;LINE INFORMATION
0E3F TINFO              =$E8            ;TIME INFORMATION
0E3F
0E3F ORB                =$A000          ;USER VIA 6522
0E3F ORRA               =$A001
0E3F DDRB               =$A002
0E3F DDRA               =$A003
0E3F T2                 =$A008
0E3F ACR                =$A00B
0E3F PCR                =$A00C
0E3F IFR                =$A00D
0E3F IER                =$A00E
0E3F
0E3F          ;+++++++)))) KB-SCHALTER AUF II  !!!
0E3F
0E3F          ;    ***  INITIALISIERUNG  ***
0E3F
0E3F          ;OBON UND OBOFF SIND VON BASIC AUS MIT SYS
0E3F          ;AUFZURUFEN. IST DIESES NICHT MOEGLICH, DANN
0E3F          ;INITIA EINMAL MIT X=USR(Y) ANSPRECHEN, DA-
0E3F          ;NACH KANN MIT POKE OUTFLG UMGESCHALTET WERDEN
0E3F
0E3F
0E3F          ;-----) VON BASIC MIT SYS AUSSCHALTEN:
0E3F OBOFF    A90D    LDA ß$0D
0E41          8D13A4  STA OUTFLG      ;RUECKSETZEN VON OUTFLG
0E44          A9DC    LDA ßkBUOUT     ;UND UOUT-VEKTOR
0E46          8D0A01  STA UOUT        ;AUF STANDARDWERT.
0E49          A9DF    LDA ßgBUOUT     ;ACHTUNG: UOUT STANDARD
0E4B          8D0B01  STA UOUT+1      ;AUF EIGENES SYSTEM
0E4E          60      RTS             ;ANPASSEN
```

```
0E4F            ;-----) VON BASIC MIT SYS EINSCHALTEN:
0E4F OBON       20890E JSR INITIA
0E52            A955   LDA #'U'        ;SOFORT UMSCHALTEN
0E54            8D13A4 STA OUTFLG
0E57
0E57            ;-----)EINSTIEG UM UOUT EINZURICHTEN:
0E57 IUOUT      A987   LDA #kANF
0E59            8D0A01 STA UOUT
0E5C            A90E   LDA #gANF
0E5E            8D0B01 STA UOUT+1
0E61            60     RTS
0E62
0E62            ;-----) EINSTIEG BEI AUFRUF MIT JSR ....
0E62 JSCALL     48     PHA              ;MIT CHR IN A
0E63            98     TYA
0E64            48     PHA              ;Y SICHERN
0E65            20890E JSR INITIA       ;JEDESMAL INITIALISIEREN
0E68            68     PLA
0E69            A8     TAY
0E6A            68     PLA
0E6B            48     PHA
0E6C            20B90E JSR PRI01        ;PRINT CHR IN A
0E6F            68     PLA
0E70            60     RTS
0E71            ;ANDERE MOEGLICHKEIT BEI AUFRUF MIT JSR.... :
0E71            ;EINMAL :    JSR INITIA
0E71            ;DANN   :    JSR PRI01      MIT CHR IN  A
0E71
0E71            ;-----) EINSTIEG UM DILINK EINZURICHTEN
0E71 DILCAL     A97E   LDA #kDILCA0     ;DILINK AUF
0E73            8D06A4 STA DILINK       ;DILCA0 AUSRICHTEN
0E76            A90E   LDA #gDILCA0
0E78            8D07A4 STA DILINK+1
0E7B            4C890E JMP INITIA
0E7E
0E7E            ;-----)DILINK EINSTIEG
0E7E DILCA0     48     PHA
0E7F            2005EF JSR OUTDIS       ;DIREKT AUF DISPLAY
0E82            20BC0E JSR PRI02        ;WEIL DILINK JA SCHON
0E85            68     PLA              ;HIERHER ZEIGT
0E86            60     RTS
0E87
0E87            ;-----) UOUT EINSTIEG
0E87 ANF        B02F   BCS PRINT        ;USER OUTPUT EINGANG
0E89
0E89 INITIA     AD03A0 LDA DDRA         ;INITALISIERUNG
0E8C            29FC   AND #$FC
0E8E            0988   ORA #$88
0E90            8D03A0 STA DDRA         ;BIT 3 UND 7 ALS OUTPUT
0E93            A9FF   LDA #$FF
0E95            8D02A0 STA DDRB         ;ALLE OUTPUT
0E98            AD0BA0 LDA ACR
0E9B            29DF   AND #$DF
0E9D            8D0BA0 STA ACR          ;T2 ONE-SHOT MODE
0EA0            AD0CA0 LDA PCR
0EA3            29EF   AND #$EF
0EA5            8D0CA0 STA PCR          ;CB1 IFR=1: HIGH TO LOW
```

```
0EA8          AD0EA0 LDA IER
0EAB          29CF   AND #$CF
0EAD          8D0EA0 STA IER          ;KEIN INTERRUPT
0EB0          A961   LDA #gTIME1
0EB2          A0A8   LDY #kTIME1
0EB4          20640F JSR TIMER        ;UM T2-IFR FLAG ZU SETZEN
0EB7          60     RTS              ;SONST WARTET PRI06 BEIM
0EB8                                  ;ERSTEN MAL VERGEBENS
0EB8
0EB8          ;    ***  AUSGABE EINES ZEICHENS  ***
0EB8
0EB8 PRINT    68     PLA
0EB9 PRI01    207AE9 JSR OUTPUT       ;KONTROLLAUSDRUCK UEBER
0EBC                                  ;DILINK
0EBC PRI02    C98D   CMP #$8D         ;TABELLE GEHT NUR BIS $8C
0EBE          9002   BCC PRI03
0EC0          297F   AND #$7F         ;WENN GROESSER, BIT 7 =0
0EC2                                  ;SETZEN
0EC2 PRI03    C940   CMP #$40         ;SEITENTRENNER ?
0EC4          D004   BNE PRI04        ;ASCII $40
0EC6          203CE9 JSR READ         ;JA, DANN DRUCK STOPPEN,
0EC9                                  ;WARTE AUF TASTENDRUCK
0EC9          60     RTS              ;IN AUFRUFENDES PROGRAMM
0ECA
0ECA PRI04    209EEB JSR PHXY
0ECD          AA     TAX
0ECE          BD720F LDA TAB,X        ;DRUCKER-CODE-TABELLE
0ED1          48     PHA
0ED2          AA     TAX              ;IN X AUFHEBEN
0ED3
0ED3          ;ZUERST SHIFT UND KEYBOARD SETZEN, BIT 3, 7
0ED3          8E01A0 STX ORRA         ;SHIFT DOWN WENN BIT3=LOW
0ED6                                  ;KEYBOARD I WENN BIT7=LOW
0ED6
0ED6          ;AUFBEREITEN DER LINE INFORMATION, BIT 2, 1, 0
0ED6          2907   AND #$07         ;INFO ABTRENNEN
0ED8          A8     TAY
0ED9          C8     INY              ;Y ALS COUNTER
0EDA          A9FF   LDA #$FF
0EDC          18     CLC              ;LOW VON CLC WIRD
0EDD                                  ;ENTSPRECHEND
0EDD PRI05    2A     ROL A            ;LINE-INFO VON BIT 0 NACH
0EDE          88     DEY              ;BIT 7 GESCHOBEN
0EDF          D0FC   BNE PRI05        ;FERTIG?
0EE1          85E7   STA LINFO
0EE3
0EE3          ;AUFBEREITEN DER TIME INFORMATION, BIT 6, 5, 4
0EE3          8A     TXA
0EE4          4A     LSR A            ;IN POS 2, 1, 0 SCHIEBEN
0EE5          4A     LSR A
0EE6          4A     LSR A
0EE7          4A     LSR A
0EE8          2907   AND #$07         ;INFO ABTRENNEN
0EEA          85E8   STA TINFO

0EEC          ;FUNKTIONSZEIT ABGELAUFEN ?
0EEC          ;DAUERTE DIE CHR. BEREITSTELLUNG LAENGER, SO
0EEC          ;MUSS NUN NICHT MEHR GEWARTET WERDEN
```

```
0EEC          A920    LDA ß$20        ;FLAG VON T2
0EEE PRI06    2C0DA0  BIT IFR         ;FUNKT TIME OUT ?
0EF1          F0FB    BEQ PRI06       ;NEIN
0EF3
0EF3          ;BUCHSTABE AN DRUCKER GEBEN
0EF3          A906    LDA ß6
0EF5 PRI07    48      PHA
0EF6          20330F  JSR LINE        ; 6 MAL LINE EINSCHALTEN
0EF9          68      PLA
0EFA          AA      TAX             ;X ALS ZAEHLER
0EFB          CA      DEX
0EFC          8A      TXA
0EFD          D0F6    BNE PRI07
0EFF
0EFF          ;BEREIT FUER NAECHSTEN BUCHSTABEN ?
0EFF          ;BEI BUCHSTABEN AUF AUSLOESEIMPULS DES
0EFF          ;DRUCKER-HAMMERS WARTEN,
0EFF          ;BEI FUNKIONSTASTEN,SPACE UND BS WIRD DER
0EFF          ;DRUCKER-HAMMER NICHT BENUTZT, DANN TIMER
0EFF          ;STARTEN UND RTS
0EFF
0EFF          68      PLA             ;WAR ES EINE FUNKTIONS-
0F00          C96F    CMP ß$6F        ;TASTE, DANN KOMMT KEIN
0F02          F01F    BEQ SPACE       ;READY-IMPULS
0F04          C96E    CMP ß$6E
0F06          F01B    BEQ LFEED
0F08          C978    CMP ß$78
0F0A          F017    BEQ BS          ;BACSPACE
0F0C          2970    AND ß$70        ;WENN TIME=0,
0F0E          F013    BEQ FUNKT
0F10          C970    CMP ß$70        ;ODER TIME=7 DANN WAR ES
0F12          F00F    BEQ FUNKT       ;EINE FUNKTIONSTASTE
0F14
0F14          A901    LDA ß$01        ;WENN NICHT, DANN
0F16 PRI08    2C01A0  BIT ORRA        ;WARTE BIS DRUCKMAGNET...
0F19          F0FB    BEQ PRI08       ; ...ON?
0F1B PRI09    2C01A0  BIT ORRA
0F1E          D0FB    BNE PRI09       ; ...WIEDER OFF?
0F20          4C2D0F  JMP RETURN
0F23
0F23
0F23          ;BEI FUNKTIONSTASTE TIME1 WARTEN
0F23 SPACE                            ;TIME1 ENTSPRICHT TIME
0F23 LFEED                            ;FUER SPACE, WIRD ABER
0F23 BS                               ;FUER ALLE FUNKTIONS-
0F23 FUNKT    A0A8    LDY ßkTIME1     ;TASTEN BENUTZT
0F25          A961    LDA ßgTIME1
0F27          8C08A0  STY T2
0F2A          8D09A0  STA T2+1        ;USER TIMER 2 STARTEN

0F2D RETURN   A920    LDA ß$20        ;$20 WIRKT BEI RUECKKEHR
0F2F                                  ;IN BASIC UND ASSEMBLER
0F2F                                  ;NEUTRAL
0F2F          20ACEB  JSR PLXY
0F32          60      RTS             ;RETURN IN AUFRUFENDES
0F33                                  ;PROGRAMM
0F33
0F33          ;    ***  SUBROUTINES  ***
```

```
0F33          ;LINE IN ABHAENGIKEIT VON LINE- UND
0F33          ; TIME-INFO EINSCHALTEN
0F33
0F33          ;WARTE AUF STARTIMPULS VON LINE 0 (TRIGGERUNG)
0F33 LINE     A9FF    LDA ß$FF        ;CLEAR INTERRUPT FLAG
0F35          8D00A0  STA ORB
0F38          A910    LDA ß$10
0F3A LIN1     2C0DA0  BIT IFR
0F3D          F0FB    BEQ LIN1        ;HIGH-LOW UEBERGANG AN
0F3F                                  ;LINE 0 ?
0F3F
0F3F          ;JA, WARTE NUN ENTSPRECHEND TIME-INFO IMPULSE DER
0F3F          ;TAKTLEITUNG AB
0F3F TIME     A5E8    LDA TINFO
0F41          AA      TAX             ;X ALS COUNTER
0F42          E8      INX
0F43 TIM1     A902    LDA ß$02
0F45 TIM2     2C01A0  BIT ORRA
0F48          F0FB    BEQ TIM2        ; ...HIGH?
0F4A TIM3     2C01A0  BIT ORRA
0F4D          D0FB    BNE TIM3        ; ...WIEDER LOW?
0F4F          CA      DEX
0F50          D0F1    BNE TIM1        ;GENUEGEND IMPULSE AB-
0F52                                  ;GEWARTET?
0F52
0F52          ;JA, JETZT LINE FUER ZEIT "TIME2" EINSCHALTEN
0F52          A5E7    LDA LINFO
0F54          8D00A0  STA ORB         ;LINE   ** ON **
0F57          A903    LDA ßgTIME2
0F59          A0E8    LDY ßkTIME2
0F5B          20640F  JSR TIMER
0F5E          A9FF    LDA ß$FF
0F60          8D00A0  STA ORB         ;LINE   ** OFF **
0F63          60      RTS             ;ENDE EINES DER SECHS
0F64                                  ;EINSCHALT AUFRUFE
0F64
0F64          ;WARTE ENTSPRECHEND TIME-INFO
0F64 TIMER    8C08A0  STY T2
0F67          8D09A0  STA T2+1        ;START USER TIMER 2
0F6A          A920    LDA ß$20
0F6C TIME0    2C0DA0  BIT IFR
0F6F          F0FB    BEQ TIME0       ;TIME OUT?
0F71          60      RTS
0F72
0F72          ;TABELLE MIT DRUCKER-CODE
0F72 TAB

                      .BYT $6F,$6F,$6F,$6F,$7F,$6F,$0E,$6F ;00
                      .BYT $65,$78,$7E,$6F,$6F,$6E,$6F,$6F ;08
                      .BYT $6F,$6F,$6F,$6F,$7D,$6F,$A2,$56 ;10
                      .BYT $6F,$76,$6F,$6F,$6F,$6F,$6F,$6F ;18
                      .BYT $6F,$22,$20,$2E,$C0,$40,$60,$12 ;20
                      .BYT $50,$63,$C6,$3E,$3A,$1A,$2A,$36 ;28
                      .BYT $BE,$18,$28,$38,$48,$68,$58,$6B ;30
                      .BYT $5E,$4E,$26,$10,$49,$30,$5F,$32 ;38
                      .BYT $6F,$14,$62,$37,$34,$33,$44,$57 ;40
```

```
.BYT $61,$45,$51,$41,$31,$42,$52,$35 ;48
.BYT $25,$13,$43,$24,$53,$55,$47,$23 ;50
.BYT $27,$17,$54,$11,$46,$21,$15,$56 ;58
.BYT $6F,$1C,$6A,$3F,$3C,$3B,$4C,$5F ;60
.BYT $69,$4D,$59,$49,$39,$4A,$5A,$3D ;68
.BYT $2D,$1B,$4B,$2C,$5B,$5D,$4F,$2B ;70
.BYT $2F,$1F,$5C,$19,$A2,$29,$1D,$6D ;78
.BYT $80,$71,$72,$73,$74,$A0,$B6,$A6 ;80
.BYT $02,$01,$76,$16,$1E             ;88
.OPT LIS                                #
```

Bernhard Kokula, 6700 Ludwigshafen

Prozeßtechnik mit Microcomputern (3)

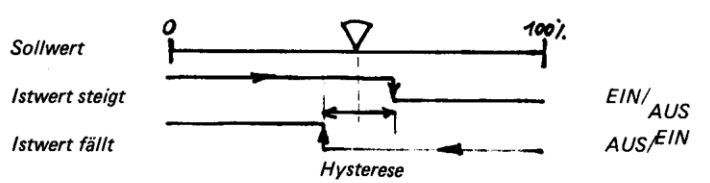
Der Zweipunktregler

Mit den früher veröffentlichten Modulen können wir: 1 bis 8 Analogwerte digital in die 'HEXREG, X' holen, mit dem 'FILTER' diese Meßwerte glätten, mit 'HEXDEZ' in Dezimalzahlen (BCD) wandeln und mit 'ANZEIG' diese Meßwerte wieder anwählen und anzeigen. Nunmehr folgen drei Module mit Reglern, und zwar:

Zweipunktregler	für	Pumpen-Ansteuerung Magnetventile Brenner, Kühlaggregate oder als Grenzwertmelder für 1 Melderichtung
Dreipunktregler	für	Stellantriebe mit 2 Drehrichtungen Heizen - Kühlen oder als Grenzwertmelder für 2 Melderichtungen
Puls-Proportionalregler	für	Mischventile Verhältnisregeleungen Regelungen mit großen Zeitkonstanten

In dieser Folge beginnen wir mit dem einfachsten Regler, dem Zweipunktregler, hier mit ZPR abgekürzt. Doch zunächst die Frage: Was ist ein Regler? Dazu die DIN 19266, Regelungs- und Steuerungstechnik, Begriffe und Benennungen. Dort heiß es: Die Regelung ist ein Vorgang, bei dem die zu regelnde Größe laufend erfaßt (Istwert x) und mit der Führungsgröße (Sollwert w) verglichen wird. Die Stellgröße (Ausgang y) wird in Abhängigkeit vom Ergebnis dieses Vergleichs im Sinne einer Angleichung von Istwert an Sollwert beeinflußt.

Damit sei der Theorie genug. Die Arbeitskennlinie beschreibt das Arbeiten des Zweipunktreglers anschaulicher:



Ein Zweipunktregler setzt sich aus dem Analogteil oder einer Arithmetikgruppe und dem Schaltverstärker zusammen. Das gilt sowohl für den einfachen mechanischen wie auch für den vom Mikroprozessor gesteuerten Regler. Nehmen wir den einfachsten und am häufigsten benutzten Bimetallregler, wie er in jedem Bügeleisen oder Toaster zu finden ist:

Vorspannung = Sollwert

Durchbiegung = Istwert

Schaltverstärker (Mikroschalter)

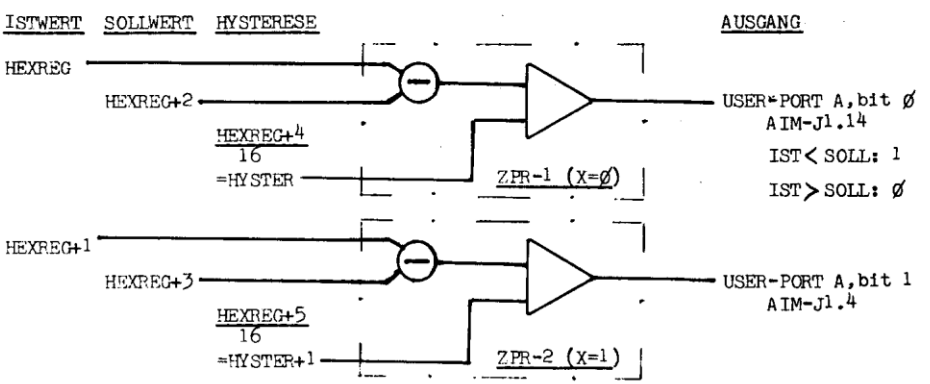
Analogteil (Bimetall)

Die Einstellschraube erzeugt hier eine Vorspannung, den Sollwert. Die Temperatur im Bimetallstreifen bewirkt durch Verbiegen des Streifens eine Biegespannung, den Istwert. Die Differenz zwischen beiden, nämlich Sollwert minus Istwert ist die Regelabweichung. Ist sie groß genug, so betätigt sie den Mikroschalter, der hier der Schaltverstärker ist. Die zur Stabilität benötigte Hysterese liegt im Mikroschalter.

Der Zweipunktregler in Software folgt dem selben Prinzip. Im Arithmetikteil werden Sollwert und Istwert verglichen, mal mit positiver, mal mit negativer Hysterese. Liegt das Ergebnis innerhalb des Hysterese-Fensters, wird nichts unternommen, im anderen Falle EIN- oder AUS-geschaltet. Ohne die Hysterese würde der Regler zu oft, nämlich bei jeder geringsten Änderung des Istwertes jedesmal schalten.

Für den Experimentierfall sind hier alle 3 Variablen Werte vom A/D-Wandler. Sie sind somit leicht zu ändern. In der Praxis wird man die Hysterese als Konstante ausführen. Da sie nur kleine Werte annimmt (1-8), wird zusätzlich der vom A/D-Wandler kommende Wert durch 16 dividiert, so läßt er sich besser handhaben. Dieser Teiler ist bei der Hysterese wieder zu entfernen. Wer nur zwei A/D-Eingänge aufbereitet, kann gleich die Konstante einsetzen.

Für das Programm-Modul Zweipunktregler gilt folgendes Block-Schaltbild:



Autor und Herausgeber bitten Leser, die zur 'Prozeßtechnik' eigene Module entworfen haben, um deren Zusendung zum Zwecke der Veröffentlichung.

```
0003            ;<ZPR.3> AUS ZPR.2 /19.3.82
0003            ;VORSPANN DAMIT MODUL ASSEMBLIERFAEHIG IST
0003                  * =$B0            ;ZEROPAGE-LINK
00B0 ZYKLUS           =6                ;MINDESTENS FUER 2 ZPR.3
00B0 HEXREG           *=*+ZYKLUS
00B6 XPOINT           *=*+1
00B7 VARADW           =*
00B7
```

```
00B7 PROADW          =$0200          ;PROGRAMM-LINK
00B7           ;ADW IST NEUE BEZEICHNUNG VOM MUX555-MODUL,
00B7           ;UND BEDEUTET ANALOG/DIGITAL-WANDLER
00B7
00B7           ----------------------
00B7           MODUL 2-PUNKT-REGLER
00B7
00B7           ;ZUM REGELN EINER ODER MEHRERER PROZESSGROESSEN.
00B7           ;REGLER WIRD MIT <X>-POINTER ANGEWAEHLT, UND
00B7           ;JEDER REGLER HAT EIGENE IST- UND SOLLWERTE
00B7           ;SOWIE HYSTERESE, KONSTANT ODER VARIABEL.
00B7           ;WIRKRICHTUNG: EINSCHALTEN BEI ISTWERT < SOLLWERT
00B7           ;ZUM UMKEHREN DIE BRANCHES ZU EIN/AUSMASKEN TAUSCHEN
00B7           ;JEDER REGELKREIS BRAUCHT EIGENE EIN/AUSSCHALTMASKEN
00B7           ;FUER SEINEN ZUGEORDNETEN AUSGANG.
00B7           ;DIE MASKEN SIND HIER FUER 2 REGLER VORGESEHEN.
00B7
00B7           ;#### ZPR.3 VARIABLEN
00B7                  *=VARADW
00B7 ISTREG           =HEXREG
00B7 SOLREG           =HEXREG+2
00B7
00B7 ZPR              =2              ;2 REGLER
00B7 HYSTER           *=*+ZPR         ;ANZAHL DER ZPR-REGLER
00B9           ; KANN KONSTANTE SEIN, SIEHE BEISPIEL HYSTEX
00B9 VARZPR           =*
00B9
00B9 ZPPORT           =UDRA           ;USER-6522,PORT A
00B9
00B9                  *=PROADW
0200
0200           ;----- ZPREGLER-PORT INITIALISIEREN
0200 INIZPR    A903   LDA #%000011    ;BIT 1 UND 0=1: OUT
0202           0D03A0 ORA ZPPORT+2    ;ZPPORT-DIREKTION
0205           8D03A0 STA ZPPORT+2
0208           60     RTS
0209
0209           ;++++++++++ PROGRAMM ZPR
0209
0209 ZPREGL    A6B6   LDX XPOINT      ;WELCHER REGLER IST DRAN?
020B           203702 JSR ZP16        ;HYSTERESE-16-TEILER
020E           ; NUR WENN HYSTERESE VOM A/D-WANDLER KOMMT!
020E
020E           ;----- IST ISTWERT > SOLLWERT+HYSTERESE ?
020E           B5B2   LDA SOLREG,X    ;SOLLWERT
0210           75B7   ADC HYSTER,X    ;PLUS HYSTERESE
0212           D5B0   CMP ISTREG,X    ;MIT ISTWERT VERGLEICHEN
0214           9011   BCC AUS         ;JA, AUSSCHALTEN
0216
0216           ;----- IST ISTWERT < SOLLWERT-HYSTERESE ?
0216           B5B2   LDA SOLREG,X    ;SOLLWERT
0218           F5B7   SBC HYSTER,X    ;MINUS HYSTERESE
021A           D5B0   CMP ISTREG,X    ;MIT ISTWERT VERGLEICHEN
021C           B001   BCS EIN         ;JA, EINSCHALTEN
021E
021E           ;----- KEINE VERAENDERUNGEN
021E           60     RTS
021F
```

```
021F            ;----- EINSCHALTEN DA ISTWERT < SOLLWERT
021F EIN        AD01A0 LDA ZPPORT
0222            1D3102 ORA ZPEIN,X
0225            D006   BNE AU1          ;IMMER
0227
0227            ;----- AUSSCHALTEN DA ISTWERT > SOLLWERT
0227 AUS        AD01A0 LDA ZPPORT
022A            3D3302 AND ZPAUS,X      ;AUSSCHALTMASKE
022D AU1        8D01A0 STA ZPPORT
0230            60     RTS
0231
0231            ;----- MASKEN ZUM EIN- UND AUSSCHALTEN
0231            ; FUER 2 REGLER HIER VORGESEHEN.
0231 ZPEIN      02     .BYT %00000010   ;BIT 6, REGLER 1
0232            01     .BYT %00000001   ;BIT 7, REGLER 2
0233 ZPAUS      FD     .BYT %11111101   ;BIT 6
0234            FE     .BYT %11111110   ;BIT 7
0235
0235            ;----- BEISPIEL FUER HYSTERESE ALS KONSTANTE
0235 HYSTEX     01     .BYT 01          ;REGLER 1
0236            03     .BYT 03          ;REGLER 2
0237
0237            ;----- HYSTERESE-16-TEILER, ALS KONSTANTE NI;.HT NOETIG.
0237 ZP16       A003   LDY #3           ;4 MAL RUM
0239            B5B4   LDA HEXREG+4,X   ;HOLE ROH-DATE
023B ZP         4A     LSR A            ;TEILE DURCH 2
023C            88     DEY
023D            10FC   BPL ZP
023F            95B7   STA HYSTER,X     ;SCHAFFE WEG
0241            60     RTS
0242 PROZPR             =*
0242
0242 UDRA               =$A001           ;USER-VIA PORT A
0242
0242                    .END

001B            .PAG 'TESTPROGRAMM ZPR.3 2-PUNKT-REGLER'
001B            ;
001B            WORK
001B            JSR INIADW ;INIT A/D-WANDLER
001B            JSR INIZPR ;INIT 2-PUNKT-REGLER
001B            WO1
001B            JSR ADW ;BEARBEITE A/D-WANDLER
001B            LDX #1
001B            STX XPOINT ;SETZE XPOINTER
001B            WO2
001B            JSR ZP16 ;HYSTERESE-16-TEILER
001B            JSR ZPREGL ;BEARBEITE REGLER 1 & 2
001B            DEC XPOINT
001B            BPL WO2
001B            BMI WO1
001B
001B            .EN
```

Ein Hinweis: Neben dem neuen Modul ZPR. muß das A/D-Wandlermodul vorhanden sein, alter Name MUX555, neuer Name ADW. (Teil 1).

Aus der Branche / Produkte

MICRO-TREFF '82 am 22. und 23. Mai 1982, jeweils 10 - 18 Uhr in Ludwigshafen-Friesenheim, Willi-Graf-Haus, Leuschnerstr. 151. Die Arbeitsgemeinschaft Microcomputer im DARC (Deutscher Amateur Radio Club) veranstaltet wiederum ihr traditionelles Mai-Treffen mit einer Hobby-Computer-Ausstellung, Systemberatung und zum persönlichen Kennenlernen.

Für die 'großen' elektronischen Typenrad-Schreibmaschinen von Olivetti ETxxx, also insbesondere ET 201/221 bietet die Fa. Hard + Soft in Bayreuth ein IEEE-Bus-Interface an, das insbesondere auf das Zusammenspiel der Commodore-Rechner mit den Schreibmaschinen als flotte Schönschreibdrucker ausgelegt ist. In dieser Kombination wird das professionell gefertigte Interface seit Monaten beim Herausgeber in einwandfreier Funktion betrieben. Mit etwas Geschick ist die Montage in einer guten Stunde erledigt. Die über Flachkabel im Kabelkanal herausgeführte IEEE-Buchse wird an der Rückseite der Schreibmaschine in einer dafür bereits vorbereiteten Mulde befestigt, so daß das übliche Geräteverbindungskabel einen festen Anschlußpunkt hat (keine Gehäusearbeiten). Nach dem enttäuschenden Versuch mit dem Interface einer Mannheimer Firma ist festzustellen, daß das hier besprochene der IEEE-Norm entspricht und darüber hinaus einige sehr nützliche Eigenschaften aufweist: Der Zeichensatz kann per Befehl wahlweise auf ASCII und auf die CBM-Darstellung der Zeichen dirigiert werden. Unter ASCII entspricht die Belegung der deutschen Umlaute und des '?' der DIN. Weiterhin: Per Software kann das Interface auch auf eine andere Geräteadresse als '4' eingestellt werden, auf die es nach dem Einschalten hört. So ist es z.B. möglich, einen Rechner auch mit einem Matrixdrucker (Gerät 4) und einer Olivetti-Schreibmaschine zusammen zu betreiben, z.B. zum Fakturieren, wo das eine Gerät das Jounal und das andere die Rechnung schreibt. - Besagte Olivetti-Maschinen zeichen sich durch eine Vielzahl von Funktionen aus, z.B. 10, 12, 15 Zeichen Schreibbreite je Zoll, Proportionalschrift, Unterstreichung, Fettdruck, Blocksatz, Tabulation, Speichern von Druckformaten und Standard-Texten. Alle diese Funktionen können vom Rechner angesteuert werden, z.B. vertikale und horizontale Formularformate übermitteln und speichern. Bei der Ausrüstung des Interface mit einem 1K Pufferspeicher ist ein sehr flottes, überlappendes Arbeiten möglich. - Daneben gibt es ein **Universal-Druckerinterface IEEE auf Parallelschnittstelle** (Z.B. Centronix, Epson, NEC, Qume usw.), bei dem der Zeichensatz per Software nicht nur von ASCII auf CBM, sondern auch auf andere Zeichensätze umgeschaltet werden kann. Im weiteren Angebot finden sich Echtzeituhr, ROM-Box und der bewährte, beliebte NEWTIM (vielseitiger Monitor) sowie die 8K BASIC System Help. Info: Hard + Soft R. S. GmbH, Gagernstr.4, 8580 Bayreuth, Tel.: 09 21 - 6 88 77.

Die GWK-Elektronik in Herzogenrath hat für den AIM 65/40 bereits jetzt ihr 12K Extended BASIC fertiggestellt, noch ehe das normale BASIC von Rockwell vorliegt. Auch die reiche Palette von Expansionskarten, u.a. Floppy Controller, Video, Eprom-Programmer usw., konnte auf den neuen Computer adaptiert werden, so daß von dort aus bereits vollständige Systeme geliefert werden können. Diese Palette, auch die für den normalen AIM und für den 6809 mit Betriebssystem FLEX bzw. OS9 Level 2, wird auf der Hannover-Messe 82 in Halle 12 gezeigt werden. Eine erste Vorstellung erfolgte hier in Ahrensburg bei einem Seminar des Herausgebers für Landesstudienleiter an Gewerbeschulen in Schleswig-Holstein, insbesondere auch mit dem hochauflösenden Grafik-Prozessor, der jetzt auf 6809 und 6502 läuft. - Dem Vernehmen nach wird in Hannover ein komfortables Adressenverwaltungssystem für 6809 gezeigt werden und man wird sich dort bemühen, Vertriebspartner im In- und Ausland zu finden.

Für Commodore-Rechner bietet die Fa. Roßmöller in Bonn zahlreiche nützliche Erweiterungen an. Getestet wurde hier zunächst das COMBASIC (für BASIC 3 und 4), ein Softwarepaket für kommerzielle Programme, das eine Reihe zusätzlicher Befehle zur Verfügung stellt. Für Stringverarbeitung finden wir vor allem den INSTRING-Befehl, der eine Variable oder auch ein Array (auch ab Zeichenstelle Nr.) auf Übereinstimmung mit dem Suchstring absucht, den Alpha-Sort eines Arrays auf ... Stellen. Ein INSERT in Strings ab Stelle Nr. ist möglich und man kann Strings in geschützte Bereiche bringen und sie z.B. nach einem NEW von dort wieder laden. Für die Ein- und Ausgabe hat man erweiterte GET- und INPUT-Befehle, ein PRINTUSING, Hardcopy, Cursor-Setzen, ein MOVE, Interuptbeeinflussung, ein ON ERROR GOTO, Stackbereinigung nach Schleifen, RESTORE Datenzeiger auf Zeile Nr., Umrechnung HEX/DEZ usw.. In der Summe handelt es sich um einfache und leicht zu handhabende Befehle mit immer wieder nützlichen und den Programmierer entlastenden Funktionen. - Aus Zeitmangel konnten weitere eingetroffene Hardware-Komponenten nicht mehr getestet werden, ein 4K CMOS RAM, das mit Flachkabel einem beliebigen ROM-Steckplatz zugeordnet wird, sowie eine 16-fach Eprom-Umschaltplatine für 12 Stück 2 KB oder 4 KB Roms und 4 Zeichengeneratoren, die auch eine hochauflösende Grafik möglich macht und die per POKE geschaltet wird. - Ebenso noch nicht das Eprom-Programmiergerät nebst Betriebsprogramm für die Typen 2716, 2516 und 2532. Info: Fa. Martin Roßmöller, Kaiserstr. 34, 53 Bonn, Tel. 0221 - 76 019 31.

AIM 65 mit CPU 6809: *Mit dem Aufruf 'Packen wir's an!' regt Herr Kokula aus Ludwigshafen die Bildung eines Arbeitskreises an, der das Monitor-Programm des AIM 65 in vergleichbarer Form für die CPU 6809 umsetzt. Bekanntlich gibt es bei der Fa. Dohmann in Gütersloh bereits eine Aufsteckplatine mit 6809 statt 6502-CPU. Dort, wie auch bei anderen Firmen, die ähnliches planen, fehlt offensichtlich die Zeit, auch die Firmware dazu zu entwickeln. Herr Kokula schlägt daher vor, daß die Teilnehmer eines solchen Arbeitskreises jeweils Module zur Ausprogrammierung übernehmen, die im Editor des AIM (Tape) symbolisch programmiert, später zusammengetragen, assembliert und ihnen als Festwertprogramm wieder zur Verfügung gestellt werden. Bei einer solchen Aktion sollten in ASCII formulierte Messages, die Namen der ausführenden Hauptroutinen und die Hantierungen zweckmäßigerweise möglichst gleich bleiben. Der Platz sollte für einige Leistungserweiterungen ausreichen. Ein Cross-Assembler, z.B. auch unter FORTH geschrieben, scheint nicht unbedingt nötig, obwohl er wie auch die eigentliche Aktion eine lehrreiche Aufgabe wäre. Hier und anderswo stünden 6809-Maschinen mit Assembler zur Verfügung, in denen man die auf dem AIM geschriebenen Quelltexte umsetzen könnte.*

An einer Mitarbeit interessierte Leser sollten ihre Bereitschaft möglichst bald an den Herausgeber zur weiteren Organisation melden. Nach weiterem Abstimmen der Gedanken und Ziele könnte dann z.B. zum 5. Juni, Samstag ein Arbeitstreffen in möglichst geografischer Mitte der Beteiligten einberufen werden. - Eine ähnliche Aktion wäre nebenbei auch für CBM interessant, hier würde es innerhalb eines umschaltbaren Systems darauf ankommen, mit einem Urlader ein entsprechendes Betriebssystem hereinzuziehen.

Rockwell International sandte jüngst verschiedene Pressemitteilungen, die wie folgt zusammengefaßt werden: Für den AIM 65 soll ab März 1982 ein Gleitkomma-Mathematikpaket lieferbar sein (Adressen $D000-DFFF), das auch komplexe mathematische Problemstellungen auf Maschinenebene zu lösen gestattet. Es ist von Assemblerprogrammen oder auch FORTH her aufrufbar und bietet alle gebräuchlichen Tischrechnerfunktionen der Trigonometrie, der Logarithmen, Multiplikation, Division und Polynomberechnungen. Rechengenauigkeit 9 Stellen mit Abrundung aus der zehnten.

Die Rockwell-Organisation wird weiterhin ein Gehäuse für den AIM 65 mit oder ohne Stromversorgung liefern.

Für die Einarbeitung in die Telekommunikation steht das 'R24 Integral Modem user's manual' zur Verfügung, ein 152seitiges Anwenderhandbuch für die Anwendung und das Design der R24 1200/2400 Baud-Modembaugruppen. Es enthält Informationen über Modemwahl, Datenübertragungszeiten, Signale, Schaltungen usw.

Bei Mikroprozessorschaltkreisen wurde der Leistungsbereich weiter abgerundet. Jetzt sind die 2-MHz-Versionen folgender Bausteine ab Lager lieferbar: CPU 6502, 6512 und die Interfacebausteine PIA, VIA, RIOT und ACIA.

Ab Ende 1982 sollen CMOS-Bausteine als Prototypen lieferbar sein für RC6502 (CPU) und RC6520 (PIA). Die CPU soll dabei eine Taktgeschwindigkeit von 4 MHz haben, den Durchsatz also erheblich beschleunigen. - Angekündigt wurden ferner die CPU 6511 mit RAM und I/O, jedoch ohne ROM. Daneben kommen drei neue Einchipper mit CPU, ROM RAM, USART-Kanal und 2 Timern.

EUROCOM II V7 mit CPU 6809 der Firma ELTEC in Mainz jetzt lieferbar. Das neue Computerboard in der Größe einer Doppel-Europa-Karte gestattet es, ein vollständiges Floppy-Disk-System mit bis zu vier 5"- oder 8"-Floppy Disks aufzubauen. Der Controler WD 1794 läßt Double Side und bei 5" auch Double Density zu. Bestückung der Karte bis max. 64 KB RAM und 8KB ROM. Durch den Vollgrafik-Videocontroller mit einer Auflösung von 512H x 256V Bildpunkten sind Text und Grafik beliebig mischbar, ebenso Gestalt und Größe eines jeden Zeichens per Software frei definierbar. Zwei auf der Platine enthaltene PIAs 6821 bieten dem Benutzer 40 E/A-Pins. Mit dem 6850 ACIA ist eine V24-Schnittstelle aufgebaut, Baudraten von 50 bis 19200 Bits/s. Die Adreßdekodierung on board ist durch PALs kundenspezifischen Anforderungen anpaßbar.

Das System ist durch verschiedene Karten erweiterbar. Software. Für das Floppy Disk-System wird das FLEX Betriebssystem angeboten. Zur Verfügung steht PASCAL und FORTH. Hinzu kommen ein 'C' Compiler, Extended BASIC, Editor, Assembler.

Info: ELTEC Elektronik GmbH, Postfach 18 47, 6500 Mainz 1, Tel 061 31 - 50 031.

#

Bücher

Schneider, Wolfgang: 'BASIC für Fortgeschrittene' in der Serie 'Programmierung von Microcomputern (3)', 189 S., Vieweg-Verlag, Braunschweig/Wiesbaden 1982, ISBN 3-528-04197-8. Nach seiner empfehlenswerten 'Einführung in BASIC' im selben Verlag legt der Autor (Professor an der FHS in Wolfenbüttel) wiederum ein sorgfältig durchgearbeitetes Buch vor, das auf den ersten 121 Seiten folgende Kapitel abhandelt: BASIC-Sprachelemente, Textverarbeitung, Verarbeitung von logischen (Boolschen) Größen, Arbeiten mit Zufallszahlen, Unterprogrammtechnik. Es folgen im zweiten Teil vollständig ausprogrammierte exemplarische Beipiele: Ver- und Entschlüsselung von Texten, Formatierung von Zahlen, zwei Spiele und Lottozahlen, Bearbeitung logischer Fragestellungen, Simulation eines Problems der Wahrscheinlichkeitslehre, Schreiben von Rechnungen und alphabetisches Ordnen von Texten. In allen Abschnitten finden wir ausreichende Erklärungen, druckmäßige Hervorhebung der wichtigen Aussagen, sauber abgegrenzte Tabellen, Programmbeispiele und Übungen. Die ausprogrammierten Beispiele des zweiten Teils sind daneben mit Programmablaufplänen versehen, so daß die Belehrung durch dieses verständlich geschriebene Buch eine Freude ist.

PASCAL Handbuch für AIM 65, deutsch, ca. 96 Seiten, Bremen 1982. Dieses von der Firma Reco unter H. J. Regge herausgegebene und zusammen mit S. Jellinghaus ins Deutsche übertragene Handbuch für das INSTANT PASCAL des AIM 65 ist eine von Rockwell autorisierte Übersetzung des amerikanischen Anwenderhandbuches. Nachdem seit März 82 die PASCAL-ROMs lieferbar sind (auch von Reco, Preis unverb. ca. 327,- DM inkl. MwSt), wird diese sorgfältig aufgemachte deutsche Ausgabe sicher eine schnelle Verbreitung finden. Bezug/Anschrift: Reco, Fesenfeld 57, 2800 Bremen 1.

Hofacker, W. 'Programme für VC20, Spiele, Utilities, Erweiterungen', Hofacker-Verlag, Holzkirchen 1982, ca. 168 Textseiten, DM 29,80. Der Erscheinungszeitpunkt für dieses Buch ist gut gewählt, es begleitet den eben erst lieferbaren VC 20 von Commodore von Anfang an. Wir finden in ihm zahlreiche BASIC-Programme, einen in BASIC geschriebenen Monitor und zahlreiche Vorschläge für die Speichererweiterung mit Platinen-Layout sowie Experimente und Programme für das Interfacing. Während diese Abschnitte vor allem den Anfänger und den Bastler (für sie wird ja der Volkscomputer angeboten) interessieren und anregen werden, so wird der erfahrenere Betreiber dieses Buch wegen der in ihm enthaltenen Speicherbelegungspläne und wegen der fünfseitigen Liste der BASIC-Systemvariablen im Bereich der Adressen 0000 bis 1019 (dezimal) für nützlich finden.

Krizan, Peter und Kaufmann, Klaus-Dieter: 'Spaß mit BASIC', Idea-Verlag, 2. Aufl., Pucheim 1981 224 S. ISBN 3-98-00371-7. Der Untertitel dieses munter aufgemachten und mit vielen Zeichnungen und Programmbeispielen und Flußdiagrammen versehenen Buches lautet: Ein heiterer Computer-Sprachlehrgang von der Pike auf. Mit einer guten Didaktik gelingt es den Autoren tatsächlich, auch den Anfänger Schritt um Schritt in den Gebrauch der Programmiersprache einzuführen und ihn vor allem auch interessiert zu halten. Ein empfehlenswertes Buch für den Anfänger - und damit auch als Geschenk für unsere Junioren geeignet. #

Editorial

Die Leserumfrage im März erbrachte einige hundert aufschlußreiche Antworten. Viele Leser sandten zusätzliche briefliche Stellungnahmen. Der Herausgeber dankt allen Beteiligten herzlich und bedauert, nicht immer individuell antworten zu können. Daher folgender Bericht:

Etwa die Hälfte aller Einsender betreibt nur einen Mikro-Computer. Oder anders: die andere Hälfte betreibt zwei und mehr, vor allem den AIM 65/PC 100 oder Rechner von Commodore. Nach den bisher behandelten Themen und auch nach der Modellverbreitung treten APPLE, JUNIOR und 6809-Systeme erwartungsgemäß dahinter zurück. Überraschenderweise gibt es auch Leser, die Systeme mit einer anderen CPU betreiben die also offensichtlich mehr die aufgezeigten allgemeinen Lösungswege beobachten.

Die vorliegenden Antworten bedeuten für diese Zeitschrift eine Bestärkung der Linie, möglichst viele Darstellungen für eine eng umgrenzte Computerfamilie zu bringen, nämlich die des 6502, zunehmend auch des 6809 und im weiteren Verlauf ggfs. des 68000. Neben der Assembler- und Interfaceprogrammierung für diese wird der Gebrauch der neueren Hochsprachen FORTH und PASCAL wichtig. Wenn man einmal von den vielen individuellen Leserwünschen absieht, so finden die größeren abgedruckten Dienstprogramme eine dankbare Aufnahme.

Daneben wird häufig der Wunsch geäußert, Lösungswege allgemein algorithmisch aufzuzeigen, nämlich in der Form von Fluß- oder Blockdiagrammen, kommentierendem Text und ggfs. kurzen Programmsequenzen, denn Programme werden häufig nicht in der veröffentlichten Form über-

nommen, sondern in das Anwendersystem eingebunden. Hier liegt ein wichtiger Hinweis für künftige Autoren, der an anderer Stelle noch besonders ausgeführt ist. Man sollte Programme, mehr noch als bisher, auch für Besitzer eines anderen Systems als das des Autors durchsichtig machen, indem man die spezifischen Adressen des eigenen Systems (Monitor-RAM, Pointer in der Zeropage, benutzte Interfaces) sowie die Leistungen der benutzten Systemroutinen erklärt, so daß man diese Leistungen ggfs. nachcodieren kann.

In diesem Zusammenhang ist der häufig geäußerte Wunsch nach einem übergreifenden 6502-Betriebssystem zu erwähnen, etwa ähnlich CP/M oder FLEX usw.. - Wir haben es nicht, dafür eine Vielzahl von Maschinen mit sehr unterschiedlicher RAM-Ausstattung und mit Video- und sonstigen Interfaces, die etwa bei $8000 oder $A000 hardwaremäßig mitten im Adreßbereich liegen. Es tritt damit die Frage auf, ob man von der Betreiberseite (und nicht von der eines marktbedeutenden Anbieters) her noch zu einer Vereinheitlichung wenigstens in gewissen Bereichen kommen kann. Voraussetzung wäre in jedem Fall zunächst die Formulierung der Anforderungen und ihre Diskussion brieflich, telefonisch und in der Form von Artikeln für diese Zeitschrift. Man sollte sich dabei auch vor Augen halten, daß der Gebrauch vorhandener Sprach-ROMs behindert werden könnte, wenn man nicht umschaltbare RAM-ROM-Dekodierungen der Adreßbereiche hat bzw. sich dazu entschließt, Sprachen grundsätzlich in ein entsprechendes RAM zu laden. Hinsichtlich eines möglichen relokativen Ladeformates finden sich Grundüberlegungen im Artikel RALOAD des Herausgebers in Heft 1, Seite 3, bzw. im 'Dsa Buch 1-6 des 65xx MICRO MAG' auf Seite133.

Betreiber eines AIM 65/PC 100 äußern häufig den Wunsch, auf ihrem Gerät auch eine 6809-CPU betreiben zu können. Eine entsprechende Aufsteckplatine für die CPU ist schon lieferbar, allein es fehlt noch ein vergleichbares Betriebssystem in Festwertspeichern. Daher ist an anderer Stelle ein entsprechender Aufruf zu einer Mitarbeit an Firmware-Modulen enthalten. Dieses Projekt ist zweifellos einfacher als ein übergreifendes Betriebssystem zu verwirklichen.

Hinsichtlich der höheren Sprachen war es eine Überraschung, besonders bei AIM-Betreibern eine bereits erhebliche Verbreitung des FORTH und des PL/65 zu sehen, die sich noch nicht in entsprechenden Autoren-Beiträgen niedergeschlagen hat. Noch größer ist bei allen Systembetreibern die Bereitschaft, FORTH und PASCAL (spez. AIM) ggfs. anzuschaffen, PL/65 tritt dahinter zurück. Autoren-Beiträge vor allem zu den beiden erstgenannten Sprachen sind daher von großem Interesse - und natürlich weiterhin in Assembler.

Das 65xx MICRO MAG wird weiterhin die Vielfalt der Betätigungen bei Lesern und Autoren wiederspiegeln und dabei immer einmal Aufsätze für Minderheiten abdrucken, die die auf ein allgemeineres Publikum ausgerichteten großen Zeitschriften nicht abdrucken können. Die künftige redaktionelle Arbeit kann sich dabei an den Ergebnissen der Leserumfrage orientieren.

Im Juni 1982 beginnt der 5. Jahrgang des 65xx MICRO MAG ! #

Hinweise für Autoren

Beiträge zu dieser Zeitschrift sollten zweckmäßig unter folgenden Gesichtspunkten vorbereitet werden: Themenwahl, Aufbau des Artikels und Bereitstellen der Druckvorlage.

Die Themenwahl

Das gewählte Thema sollte möglichst viele der ernsthaft mit Computern befaßten Leser ansprechen. Im Zweifel, und um Überschneidungen mit evtl. bereits vorliegenden Arbeiten zu vermeiden, stimme man sich im Vorwege kurz ab, ehe man die Mühe der Niederschrift auf sich nimmt.

Neben der in der Vergangenheit gepflegten maschinennahen und Interfaceprogrammierung werden künftig auch der Gebrauch des FORTH, PASCAL und des PL/65 interessieren, daneben Betriebssysteme, Mehrbenutzersysteme, Prozessorverbund, Anwenderberichte über neue Computer und ihre Peripherie sowie über Softwarepakete usw.. Von Nutzen sind ferner die vielen kleinen Kniffs und Hinweise auf Besonderheiten/Fehler der Betriebssysteme. - Es ist den Autoren unbenommen, auf bereits auch anderswo veröffentlichte Quellen zurückzugreifen, wenn dabei die Quelle klar genannt wird und wenn die eigene Arbeit ihr gegenüber wesentliche Veränderungen/Erweiterungen bedeutet.

Aufbau des Artikels

Besondere Hinweise sind eigentlich nur für den Fall zu geben, daß Programme enthalten sind. In der Einleitung sollte man das Ziel und den Grundgedanken der Lösung ansprechen. Im weiteren Verlauf interessieren die evtl. benutzte besondere Hardware, ihre Verbindungen und ggfs. das Pin-out/die Funktion seltenerer Bausteine, denn Programmierung ist von solchen Gegebenheiten abhängig.

Zu Programmen erwartet man neben ihrer Erklärung und ausreichenden Kommentierung zunehmend begleitende Fluß- oder Blockdiagramme. Wenn deren Aufstellung zuviel Mühe macht, so ist zumindest in der Programm-Liste eine klare Gliederung der Funktionsblöcke wünschenswert. Der Leser möchte ferner erkennen können, welchen Bereichen der Zero Page, der Seite 1 und des Monitor-System-RAM welche Funktionen zugeordnet sind, welches Interface-Adressen sind und welche Dienste mitbenutzte Systemroutinen erbringen. Zum Programm und zu Interfaces werden im allgemeinen auch Hinweise für deren richtige Bedienung, für das Hantieren gehören.

Druckvorlagen

Alle Begleittexte werden auf die hier benutzte Setzmaschine gegeben, so daß von ihnen nur gute Lesbarkeit zu fordern ist, nicht jedoch eine besondere Art der Ausfertigung.

Anders sieht es bei Programmlisten aus: Hier kommt es auf hohen Kontrast im nachfolgenden Offset-Druck an. Hinzu kommt, daß das Reproverfahren für die zumeist benutzten blauen Farbbänder etwas blind ist, so daß der Kontrast abfällt. Daher die Empfehlung, möglichst tief und schwarz färbende Bänder zu benutzen. Im Zweifel sende man lieber auch einen Datenträger ein, von dem hier ausgelistet werden kann: Cassette für AIM und CBM, auch für TSR-Color Computer, 2040/4040-Diskette für AIM und CBM bzw. 5''-Diskette im IBM kompatiblen Format.

Zeilen in Programmlisten sollten wegen des Druckspiegels möglichst nicht breiter als 165 mm sein. Bei Zeichnungen halte man sich vor Augen, daß wegen der Verkleinerung der Vorlagen von DIN A4 auf DIN A5 im Reproverfahren unscharfe Details verloren gehen können.

Die vorstehenden Ausführungen wollen als ein Leitfaden für den Zeitraum verstanden werden, in dem man die Entstehung eines Manuskriptes und einer Arbeit noch beeinflussen kann, und nicht als eine Abschreckung. Bisher gab es eigentlich immer noch Wege, vom Manuskript zur fertigen Druckseite zu gelangen. Hauptsache: Der Leser soll viele Erklärungen und Hinweise finden! ⫲

Ein besonderer Hinweis: Verschiedene Manuskripte, die im Februar und März 1982 vorgelegt wurden, und für die der Herausgeber den Autoren dankt, können erst im nächsten Heft abgedruckt werden.

Kleinanzeigen

CBM-2/3/4/8001 Speicherbeleg., 630 Adr. 20 DM * Beschreibung von ROM-Routinen für reelle Arithm., Tape- & IEEE-Bus-I/O, Parameterübergaben an Maschinenprogramme etc. 25 DM * Zusammen 80 Seiten A4 für 35 DM * Katalog kostenl. * H.J. Koch, Liegnitzer Str. 8, 3008 Garbsen 8

Controler-/Treibersoftware für 8''-Laufwerke (auch Unterlagen/Pläne/Listings) gesucht. Kurt Peter, Kölnerstr. 6, 6053 Obertshausen. Tel. (Wochenende): 061 04 - 71 789.

Rockwell, System 65, stat. RAM-Karten á DM 1.400,- zu verkaufen. 6502-Karten lt. Titelbild in MC 2/82 DM 73,-- (unbestückt). Regge, Bremen, Tel.: 04 21 - 71 114 (ab 18 Uhr).

IBM Composer 82 (Setzmaschine): Schaltpläne/Vorschläge für den Anschluß an Mikroprozessor gesucht. Roland Löhr, Hansdorfer Str. 4, 2070 Ahrensburg, Tel.: 04 102 - 55 816.

Wegen Systemumstellung: Original BASIC ROMs mit Handbuch für AIM 65 zu verkaufen. VB DM 140,-. R. Murswiek, Tel.: 06 21 - 54 217.

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

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

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

KIM-USER NOTES 1-12 gesucht. Klaus Schuenemann (0221) 20 041 40, (0221) 76 019 31.

Interface Olivetti ETxxx/

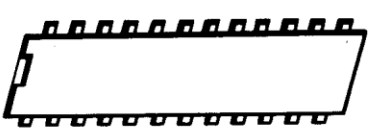
COMMODORE-Computer IEEE - IEC

Leistungsmerkmale

Interface für ET 121, 201, 221,231 lieferbar
Komplett montierte Maschinen lieferbar
Pufferspeicher 1 K-Byte zusätzlich lieferbar
Umschaltbar von ASCII auf Commodore-Standard per Software
Maschine schreibt mit Maximalgeschwindigkeit (25 z/s)
Jede Funktion über Interface ansprechbar
Geräteadresse softwaremäßig umschaltbar
Programmlisting automatisch mit markierten Cursorzeichen
Keyboard 1 und 2 automatisch und im Aufruf ansprechbar
Proportionalschrift im Blocksatz möglich
Intelligentes, Microprozessor-gesteuertes Interface
Befehlsketten werden zu einem Befehl zusammengefaßt
Dezimaltabulator mit automatischer Umwandlung Punkt (CBM) in Komma
Zeitoptimierung durch variablen, dynamisch verwalteten Pufferspeicher
Space-Optimierung: Spaceketten belegen keinen Speicherplatz
Spaces am Ende der Zeile werden nicht gedruckt

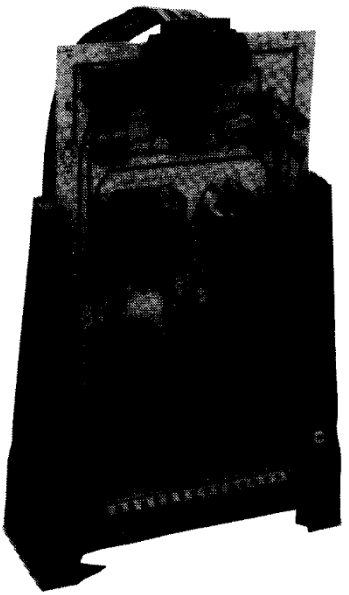
Sofort lieferbar

HARD + SOFT



Hard + Soft R. S. Microcomputer GmbH
Gagernstraße 4
8580 Bayreuth
Telefon: 09 21 - 6 88 77

Relaisplatine E-941



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

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

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

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

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

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

STECKER
INGENIEURBÜRO

Postfach 60 07 66 5000 Köln 60 Tel.: (0221) 74 51 12

AIM 65 PC 100

*** FLOPPY CONTROLLER MIT µPD 765

für Double Density und Double Sided 5" Laufwerke. Bei 35 Tracks ergibt sich eine formatierte Kapazität von 286k Bytes; mit 40 Tracks ergeben sich 327k Bytes/Laufwerk. Es können 4 Laufwerke angeschlossen werden. DOS auf der Karte. Die Sektoren werden durch das DOS dynamisch verwaltet. Das DOS beinhaltet die Befehle:

I - formatieren einer Floppy
K - File löschen
N - File umbenennen
C - gelöschte Files aus der Directory entfernen
D - Directory ausgeben
R - gelöschtes File aktivieren
T - Files übertragen
@ - File laden und starten

Das DOS arbeitet mit dem ASSEMBLER, BASIC und FORTH zusammen. Mit einer Software Option können BASIC Programme nach dem Netzeinschalten automatisch geladen und gestartet werden. Auf der Karte befindet sich eine Centronics-Schnittstelle für einen Drucker.

*** PROGRAMM-RESTART-KARTE

- 4K CMOS RAM mit Batteriepufferung
- 8K EPROM
- 2 VIA's 6522
- UART 6551
- Hardwareuhr

Hiermit ist es möglich, bei einem Ausfall der Netzspannung den Rechnerzustand in ein CMOS-RAM zu retten. Nach Wiederkehr der Spannung wird der Rechnerzustand wieder hergestellt und das BASIC- oder ASSEMBLER-PROGRAMM an der Abbruchstelle weitergeführt. Alle BASIC-VARIABLEN bleiben erhalten und müssen nicht neu eingegeben werden.

*** GRAFIC CONTROLLER MIT µPD 7220 (in Vorbereitung)

- 512 x 512 Bildpunkte durch 32k Byte unabhängiges DISPLAY-RAM
- 2 unabhängig scrollbare Bildschirmbereiche
- Grafic und alphanumerische Zeichendarstellung in unabhängigen Bereichen
- Hardcopy auf einem Drucker
- BASIC Erweiterung durch Grafic Statements
- Lichtgriffel

*** KOMPLETTE SYSTEMLÖSUNGEN

- Hardware und Software
- weitere Karten: RAM, EPROM, seriell und parallel I/O, IEEE, EPROM-Programmer, Buserweiterung
- Entwicklung und Aufbau von speziellen Karten, z.B. ADC (12 bit, 2 µs) mit unabhängigem Speicher, DAC

*** INTERFACES FÜR COMMODORE

- V24 voll duplex Schnittstelle, durch BASIC konfigurierbar, interruptgetrieben. Der Rechner wird bei der Ausgabe nicht aufgehalten und kann, während das Programm arbeitet, Zeichen empfangen.
- parallele Datenübernahme mit Handshake.

DIA-LOG GESELLSCHAFT FÜR DIGITAL-ANALOGE DATENTECHNIK MBH
Harffstraße 34, 4000 Düsseldorf 13, Telefon (0211) 723088

Computer-Baugruppen im Europaformat

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

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

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

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

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

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

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

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

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

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

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

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

DIPL.-ING. HORST NEUDECKER

INGENIEURBÜRO FÜR MESSTECHNIK

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

65xx MICRO MAG

COMPUTING · SOFTWARE · HOBBY

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

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

Bezugsbedingungen: Abonnement ab laufender Ausgabe für 6 Hefte DM 49,-- (Inlandsendpreis). Ausland/foreign via surface mail DM 54,--, USA air 26 Dollar. Abonnements laufen bis auf Widerruf mit Kündigungsmöglichkeit bis zu zwei Wochen vor deren Ablauf.

Nachliefermöglichkeiten: Hefte 1-13 sollten als Buch nachbezogen werden (s. Anzeige unten). Solange Vorrat reicht, können auch noch einzelne Hefte der Nos. 7-13 geliefert werden. Hefte 14-23 sind unbeschränkt nachlieferbar.

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

Leser-Service des Herausgebers

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

Thermokopf (Printerplatte) für AIM 65 und PC 100
mit Anleitung für den leichten Einbau. Auffrischung des Druckes — DM 25,--

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

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

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

Programming & Interfacing the 6502 with Experiments
von Marvin L. de Jong, ca. 410 S., engl., viele Schaltungsvorschläge und die Programme für den Betrieb der Interfaces dazu, didaktisch gegliedert — DM 59,--

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

Forth User's Guide
Handbuch der Firma Rockwell für ihr Fig-FORTH zum AIM 65, ca. 300 S., engl.. Erklärung des Befehlssatzes und der E/A, die im vorläufigen Handbuch noch fehlte. Geeignet auch für andere Betreiber eines Fig-FORTH — DM 24,--

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

AIM 65- Deutsches Anwenderhandbuch solange Vorrat reicht: nur — DM 12,--