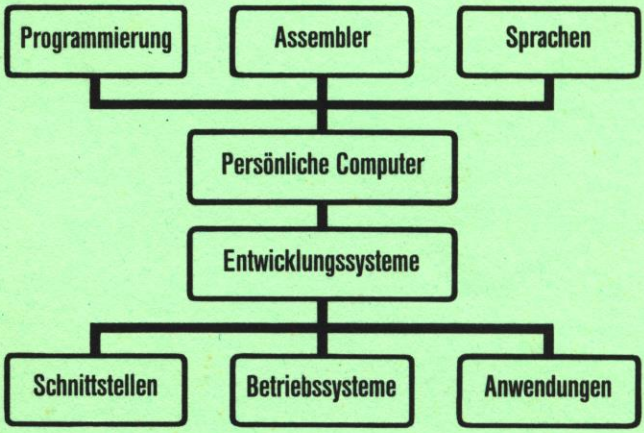
MICRO MAG

DM 9,50 Nr.38 August 1984



Inhaltsverzeichnis

AKTUELLE MIKRO-COMPUTER LITERATUR VON te-wi

Das 8086/8088 Buch
Rector/Alexy
560 Seiten, Softcover
ISBN 3-921803-11-X
DM 79,–

C 64/IEEE-488 Buch und Modul
ISBN 3-921803-29-2
DM 239,– (2. Q. 84)

M68000-Familie
Hilf/Nausch
Teil 1 – Grundlagen und Architektur
550 Seiten, Softcover
ISBN 3-921803-16-0, DM 79,–
Teil 2 – Anwendung und 68000-Bausteine (2. Q. 84)
350 Seiten, Softcover DM 59,–
ISBN 3-921803-30-6

Die 8085/8086 Interfaces
Klaus-Dieter Thies
656 Seiten, DIN A 4,
Softcover
ISBN 3-921803-23-3
DM 89,–

UNIX™ Anwenderhandbuch
Thomas/Yates
630 Seiten, Softcover
Viele Abbildungen
ISBN 3-921803-17-9
DM 79,–

Weitere Bücher:

Einführung in die Mikrocomputer-Technik
DM 66,–

MC-Bücher:
Apple II Anwenderhandbuch DM 56,–
Apple II PASCAL DM 59,–
Apple Maschinensprache DM 49,–
CBM Computerhandbuch DM 59,–
C64 Computer Handbuch (2./3. Q. 84) DM 56,–
Mein ATARI Computer DM 59,–
IBM-PC Handbuch (2. Q. 84) DM 59,–
CP/M und WordStar DM 29,80
VisiCalc, mit Diskette DM 79,–

Das 8085 Buch
K.-D. Thies
310 Seiten, A4, Softcover
Band 1, Grundlagen und Architektur,
ISBN 3-921803-25-X, DM 49,–
Band 2, Software,
ISBN 3-921803-26-8,
DM 49,–

Z8000 – Aufbau und Anwendungen
P. Stuhlmüller
464 Seiten, Hardcover
ISBN 3-912803-07-1
DM 79,–

6502 Programmieren in Assembler
Lance A. Leventhal
704 Seiten, Softcover
Viele Funktionsdiagramme und Abbildungen
ISBN 3-921803-10-1
DM 59,–

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

UNIX™ – Führer durch das System
Stanka/Lösch
ISBN 3-912803-27-6
ca. 250 Seiten,
Softcover
DM 59,–

Die C-Sprache
Stanka/Lösch
ca. 250 Seiten, Softcover
ISBN 3-912803-28-4
DM 59,–

Im 3680

Andreas Goppold, 2000 Hamburg 26

Trends in der Entwicklung der Programmiersysteme

1. Der sogenannte Anwendungsstau

Während die Fortschritte auf dem Hardwaresektor jährlich eine Verdoppelung irgendeines Systemfaktors aufweisen können, sei es Speicherdichte, Preis/Leistungsverhältnis oder Geschwindigkeit, stagniert die Leistungsfähigkeit der Programmierung auf einem Niveau, das etwa in der Mitte der 60er Jahre erreicht wurde. Folglich stöhnen alle Computerhersteller und Anwender unter dem 'Anwendungsstau' und alle klagen, daß sich die neuen Computersysteme viel besser verkaufen würden, wenn nur genügend qualitativ hochwertige Software vorhanden wäre.

Bei der Suche nach der Lösung des Softwareproblems werden immer mächtigere Programmiersprachen entwickelt, so z.B. ADA, ein Konstrukt, das alle Eigenschaften der sprichwörtlichen 'eierlegenden Wollmilchsau' aufweist. Doch trotz oder vielleicht gerade wegen solch ungeheuer komplizierter und leistungsfähiger Hilfsmittel scheint die Programmiertechnologie nicht viel schneller von der Stelle zu kommen. Bei der Suche nach Lösungen des Programmierproblems wird leicht vergessen, daß man das Problem als System ansehen muß mit vielen ineinander verschlungenen Faktoren, die sich nicht durch 'overkill' mit noch leistungsfähigeren Werkzeugen lösen lassen.

Die Faktoren, die die Programmierproduktivität erhöhen, sind nämlich andere, als die Syntaxeigenschaften einer Sprache oder ihre Mächtigkeit. Programmierproduktivität läßt sich auf zweierlei Weise erhöhen. Einerseits dadurch, daß man den Experten immer stärkere Hilfsmittel a la ADA in die Hand gibt, was zwar deren Produktivität erhöht, aber durch die immer weitere Steigerung dieser Komplexität andererseits auch den Kreis der Experten sehr klein hält, und damit die Gesamtproduktion ebenfalls sehr gering. Dies ist aber bisher der am meisten beschrittene Weg gewesen. Heutige Mikrocomputer-Ökonomien weisen aber in eine ganz andere Richtung. Es entstehen laufend neue Programmiersysteme, die es auch relativ untrainierten Kräften erlauben, arbeitsfähige Programme zu erstellen. Diese Lösungen versprechen auf lange Sicht die wirklich entscheidende Lösung des Produktivitätsproblems. Die Namen dieser Lösungen heißen: Interaktive Programmiersysteme und Objektprogrammierung.

Interaktive Programmiersysteme entwickelten sich anfangs an Universitäten und Forschungslaboratorien unter Namen wie 'Smalltalk' und 'Interlisp'. BASIC war ein früher Vertreter eines interaktiven Programmiersystems, der sehr schnell weite Verbreitung fand. Heute dringen sie mit Computern wie dem Macintosh als MacBASIC oder MacPASCAL in die Bereiche der normalen Anwendung vor. Objektsprachen fingen mit Systemen wie DBASE II an, entwickelten sich dann über integrierte Umgebungen wie LOTUS oder OPEN ACCESS weiter und lassen für die Zukunft Baukastensysteme für Programmieranwendungen erwarten.

Die Problemlösung als System

Das Prozedursystem

Das Erstellen einer Problemlösung auf einem Computer ist ein komplexes System der Interaktion Mensch-Maschine. Die Betrachtung einer Problemlösung unter dem Aspekt einer Programmiersprache herkömmlicher Art, sei es COBOL, PASCAL oder ADA, beschränkt sich auf einen spezischen und heute immer mehr zurückgehenden Anteil an diesem großen Gesamtsystem. Eine Programmiersprache definiert die Prozedurinformation, die nötig ist, um ein Problem zu lösen. Die Syntaxdefinition einer Programmiersprache tut im wesentlichen nichts anderes, als den Spielraum einzuschränken, den ein Programmier hat, um ein Problem zu lösen.Die Prozedur stellt innerhalb des Gesamtsystems einer Problemlösung eine Untermenge oder ein Untersystem dar, das man auch das Prozedursystem nennen kann.

Das Objektsystem

Neben den prozeduralen Anteilen der Problemlösung existiert noch der große Bereich des Objektproblems (daher auch der oben verwendete Name Objektprogrammierung). Es ist nämlich so, daß die Objektsysteme immer eine gewisse und normalerweise sehr komplizierte Logik aufweisen, die mit Hilfe des Prozedursystems nachgebildet wird. Der Programmierer stößt bei der Lösung seiner

Aufgabe hier aber auf die Klippe, daß er als Spezialist für das Prozedursystem normalerweise wenig Ahnung von dem Objektsystem hat. Will man in der speziellen Logik des Prozedursystems nun eine Lösung für das Objektproblem schreiben, dann stellt man sehr schnell fest, daß die logischen Probleme des Prozedursystems sich mit denen des Objektsystems multiplizieren. Es existiert eine sehr große Barriere zwischen Anwendern, die alles von ihrem Problem verstehen, aber fast nichts von der Art, wie es zu lösen ist, und den EDV-Kräften, die viel von Prozeduren verstehen, aber wenig von dem Problem, das zu lösen ist. Das Ergebnis ist, wie überall leicht festzustellen aber meist schamhaft verschwiegen, oft weniger als berauschend. Heutige Ansätze für Objektsysteme gehen in die Richtung, die zur Verfügung gestellten Prozeduren in eine Logik zu bringen, die der Logik des Objektsystems angepaßt ist. Dadurch wird das Prozedursystem 'transparent', und es wird den Fachleuten der Anwendung möglich, ihre Problemlösungen selber zu spezifizieren oder sogar zu erstellen.

Das Subjektsystem

Die herkömmliche Entwicklung der Programmiersprachen hat sich lange einseitig auf das Prozedursystem konzentriert und die Objektbezogenheit einer Problemlösung außer Acht gelassen, in dem Bestreben, möglichst gute Prozedursysteme zu entwickeln. Ebenso wurde der Aspekt des Problemlösers, des Menschen außer Acht gelassen. Dieser Faktor findet heute seinen Eingang als 'Software-Ergonomie'. Es ist ein entscheidender Faktor in der Produktivität, wie einfach oder schwierig die Interaktion des Menschen mit dem Computer/Softwaresystem ist.

Interaktivität und Ergonomie

Es wurde schon angesprochen, daß 'mächtige' Programmiertools zwar denen, die sie beherrrschen, eine bessere Produktivität erlauben. Aber sie sind äußerst schwer zu beherrschen, so daß paradoxerweise die Gesamtproduktivität sinkt, je 'mächtiger' die Softwaretools werden. So z.B. kann man mit der strukturierten Sprache PASCAL zwar komplexe Probleme leichter lösen, aber man muß diese Sprache erst beherrschen. Und das erzeugt einen Auslese-Effekt, der die Zahl der PASCAL-Programmierer stark einschränkt.

Auf der anderen Seite ist BASIC. Es kann sehr leicht gelernt werden, aber es ist schwer, in dieser Sprache komplexe Programme zu schreiben, und noch viel schwerer, sie auch zu verstehen. Das Problem, daß ein langes BASIC-Programm unleserlich ist, wird im Volksmund auch der 'Spahetti-Effekt' genannt, eine sehr treffende Beschreibung. GOTOs und GOSUBs winden sich wie Laokoon-Schlangen über Seiten und Seiten, und der geplagte Leser des Programms findet sich vor allem als hochbezahlter Seitenumblätterer bei dem Versuch, die wahre Logik des Programms zu entschlingen. Das optische Merkmal eines Spaghettigerichtes ist, daß alles ein ziemlich einheitliches Gewühl von ineinander verschlungenen Fäden ist. Ein BASIC-Programm steht diesem in nichts nach.

Der Interpreter: Eine interaktive Programmierumgebung

Trotz dieser offensichtlichen Mängel ist BASIC aber die populärste Sprache. Es stellt sich heraus, daß das gar nicht an der Sprache (also an der Syntax) liegt, sondern an seiner Interaktivität. Es ist eine interpretierende Sprache. Das heißt, der Benutzer braucht nur eine Zeile einzutippen und schon bekommt er die Antwort vom Computer. Er bekommt ein direktes feedback. Wie bei allen Dingen, die man lernen kann, geht es besser, wenn man etwas sofort ausprobieren kann. Dadurch hat sich BASIC seinen ersten Platz unter den Computersprachen geschaffen.

Interaktivitätshindernis Nr. 1: Der ECLG-Zyklus

PASCAL brachte zwar den Fortschritt der logischen Strukturierung in die Programmierung, aber es brachte gegenüber BASIC gleichzeitig einen Rückschritt. Alle bekannten PASCALs sind compiliert. Es gibt eine große Schranke zwischen dem Programmierer und seinem Programm. Diese Schranke heißt der ECLG-Zyklus. Dieses Akronym steht für die Anfangsbuchstaben von Edit-Compile-Link-and-Go. Ein Programm wird zuerst editiert, dann compiliert, dann muß man es linken und endlich darf man es starten. Nur um zu erleben, daß da noch ein paar Fehler waren. Und man muß dann wieder zurück zum Editor, Änderungen vornehmen, wieder compilieren, wobei meist noch ein paar Fehler auftreten usw. bis zur Vergasung.

Hindernis Nr. 2: Die babylonische Sprachverwirrung

Jeder dieser Arbeitsschritte aus Edit, Compile, Link und Go wird in einer anderen Umgebung gemacht. Der Editor benötigt bestimmte Kommandos (seine eigene Syntax), der Compiler wieder andere und das Betriebssystem, in dem das alles passiert, noch einmal andere. Alles in allem muß der PASCAL-Programmierer nicht nur PASCAL können, sondern noch drei bis sechs andere Sprachen, die Spezialsprachen seines Editors, seines Betriebssystems etc. Wie ist das zu vergleichen mit einem APPLE-Computer, wo das Applesoft-BASIC im Moment des Einschaltens verfügbar ist, und wo man sich nie in einer anderen Umgebung befindet, als in dieser?

Hindernis Nr. 3: Die Zeitschranke

Es ist also nicht allein die spezielle Logik der Programmiersprache, die Probleme macht. Es ist ebenso der Umstand, daß es viele und höchst verschiedene Umgebungen sind, in denen sich der Programmierer bewegen muß, und die er alle beherrschen muß. Und dann gibt es noch das Zeitproblem. Um den Editor aufzurufen, muß man warten. Das Editieren dauert seine Zeit. Das Compilieren dauert bei den üblichen Mikrocomputern meist lange genug, um einen Kaffee zu kochen, ihn zu trinken und meist noch, um die Tasse abzuwaschen und das Geschirr zu trocknen. Inzwischen ist das Problem, an dem man gearbeitet hat, scho ein wenig in den Hintergrund gerückt. Menschen, die lieber direkt mit dem Werkstück arbeiten, werden von einer solchen Arbeitsweise frustriert. Sie werden durch die lange Zeitspanne zwischen dem Entwerfen eines Programms und seinem letzendlichen Lauf in ihrer Kreativität gestört. Es ist nicht so, daß solche Menschen nicht die Begabung zum Programmieren hätten. Ihre Kreativität ist nur viel direkter, als etwa die eines mathematischen Typus, der sein Problem sowieso schon in voller Gänze im Kopf oder auf dem Papier gelöst hat, und der den Computer in Wirklichkeit eigentlich gar nicht mehr braucht.

Hindernis Nr.4: Komplexitätsprobleme

Und mit dem Zeitproblem schnappt noch eine große Falle zu: Da es ja ziemlich lange dauert, um eine Compilation zu machen oder um den Editor zu laden, macht man gerne viel auf einmal. Es ist nun eine alte Erfahrungstatsache: Wer viel arbeitet, macht viele Fehler. Leider ist bei einem Programm noch schwerwiegender als bei anderen Dingen des Lebens. Da hat man die Dinge fein säuberlich voneinander getrennt, hier das Werkzeug, da das Werkstück. In einem Programm aber wirken alle Bestandteile aufeinander ein, sie interagieren, und ihre Interaktion ist komplex. Und alle Programmteile haben ihren Zweck, es gibt keine Redundanz, die irgendwelche Fehler ausgleichen würde. Es ist eine Eigenschaft eines Systems von vielen Einzelteilen, daß die Komplexität nicht linear mit der Zahl der Einzelteile wächst, sondern exponentiell. Hat man ein Teil, ist die Komplexität auch eins. Bei zwei Teilen ist sie vier, bei drei Teilen neun. Man hat also vielfache Möglichkeiten, etwas falsch zu machen.

Modularität

Deshalb ist es beim Programmieren sehr gut, wenn man alle Einzelteile des Programms einzeln schreibt und austestet. Man nennt das auch Modularisierung. Einige noch neuere Programmiersprachen als PASCAL (zum Teil von demselben Autor) konzentrieren sich ja bekanntermaßen auf dieses Konzept. Programm-Module sollten am besten separat ausgetestet werden und danach in Kombinationen, aber möglichst nicht alle zusammen. Nur ist eine solche goldene Regel nicht viel wert, wenn es jedesmal zehn Minuten dauert, um einen solchen Durchgang des ECLG-Zyklus zu machen. Da schreibt man lieber eine ganze Menge auf einmal, compiliert sie zusammen und vergißt dann meist, daß das Debuggen dieses Komplexes Stunden und Tage dauert, auf jeden Fall mehr, als wenn man das Programm säuberlich in Teile zerlegt und sie einzeln getestet hätte.

So gesehen, liegen die Produktivitätshindernisse beim heutigen Programmieren ganz woanders als erwartet. Und Abhilfe werden wir auf jeden Fall nicht durch 'noch besser' strukturierte und 'noch mächtigere' Sprachen bekommen, wie ADA. Um es mit Weizenbaum zu sagen: 'Es wurde noch keinem, der im Begriff war, in ein Loch zu fallen, dadurch geholfen, daß er dies zehnmal schneller und effektiver tun konnte' (Weizenbaum, Computer, Power und Human Reason).

2. Die Neuen Programmiersprachen

Inzwischen sind einige Firmen dabei, etwas ganz Neues zu bieten. Diese Neuheiten heißen Mac-

PASCAL, MAC-BASIC, True BASIC, Better BASIC, Professional BASIC. Richtig, BASIC ist stark vertreten und sogar wieder gewaltig auf dem Vormarsch. Diese alte, längst von den Experten in die Abfalltonne verbannte Sprache erweist sich als die Triebfeder des Fortschrittes in der Programmiertechnologie. Wie kann das kommen?

BASICs Wiedergeburt

Eigentlich haben diese neuen BASICs mit den ältern auf Commodore, Tandy und APPLE noch soviel zu tun, wie eine Dampfwalze mit einem Sportwagen. Sie haben auf einmal viel Ähnlichkeiten mit PASCAL, aber sie haben die alte Haupteigenschaft von BASIC, die Nähe zum Benutzer, nicht verloren, und sogar noch mehr dazubekommen. Die neuen Eigenschaften dieser Sprachen heißen: Strukturierte Programmierung, inkrementelle Compilation, Grafik, Fenster, lange Namen, echte Prozeduren, dynamische Syntaxkorrektur und als großes Letztes: Die konsistente Programmierumgebung. - Dies ist nun ein ganzer Haufen von Technologie-Schlagworten, von denen nur einige bis jetzt überhaupt bekannt sind und auch nur in einem anderen Zusammenhang. Was bedeuten sie und was hat sich da auf dem Gebiet der Programmiersprachen getan?

Strukturierte Programmierung

Dieser Begriff ist inzwischen nicht mehr so ganz taufrisch. PASCAL ist ja die sprichwörtliche strukturierte Sprache. Neu ist, daß heute die modernen BASICs die strukturierten Konstrukte erlauben, als wären sie PASCAL. Den obligaten und sehr hinderlichen Zeilennummern von BASIC sind auch die Zähne gezogen worden: Wer will, darf sie immer noch verwenden, aber man muß nicht mehr. Und als Sprungadressen können Namen verwendet werden. Das gibt einem wesentlich mehr Freiheiten, z.B. mit Einrücken. Und vorbei sind die Tage, wo man mangels RENUMBER neu hinzugefügte Codestücke in irgendwelche obskuren Ecken des Programms drücken mußte, weil sie nicht mehr dahin paßten, wo sie eigentlich hingehörten, was den Spaghetti-Effekt noch verstärkte. Die Unterschiede zu PASCAL haben sich in der Tat gewaltig verringert und man kann sagen, die neuen BASICs sind eigentlich BASCAL mit der Struktur von PASCAL und der Interaktivität von BASIC.

Interaktivität

Und diese Interaktivität hat sich im Vergleich zu den früheren BASIC-Versionen noch erhöht. Es war ja der enorme Vorteil von BASIC, daß man nur eine Kommandozeile einzugeben brauchte, und schon gab der Computer einem das Ergebnis, sei es nun richtig oder falsch. Man wußte sofort, was die Maschine auf genau die Eingabe tut, die man ihr gab. Man mußte nicht in dicken Handbüchern herumblättern, um mühselig herauszufinden, was es wohl mit diesem Befehl auf sich Man brauchte es nur auszuprobieren. Bei all der Wissenschaftlichkeit, die man der PASCAL-Programmierung anhängte, wurde doch vergessen, daß alle Naturwissenschaft unserer Zivilisation damit anfing, daß sich jemand hinsetzte und es ausprobierte.

Empirisches Programmieren

Unsere heutige Zivilisation ist auf die empirische Wissenschaft aufgebaut, wobei 'empirisch' für das Ausprobieren steht. Demgegenüber war das Mittelalter die scholastische Zeit, wo man auf alles spekulierte, weil es keine Möglichkeit gab, es durch Augenschein zu beweisen. So hat die PASCAL-Programmiermethodik einen entscheidenden Nachteil, nämlich daß man nicht sofort sehen kann, was ein Kommando bewirkt. Klar, im Prinzip kann man es, indem man ein einzeiliges Programm entwirft, es editiert, compiliert, linkt und dann laufen läßt. Aber welcher vernünftige Mensch macht das schon? Und dann noch mit vielen verschiedenen Kommandos. Dazu hat man normalerweise keine Zeit. PASCAL ist scholastisch, nicht empirisch. BASIC war empirisch, und es hat heute nur noch dazugewonnen.

Inkrementelle Compilation

BASIC war interpretiert, d.h. jedes Kommando wurde vom Computer ausgeführt, so wie es war. Das dauerte natürlich wesentlich länger als mit einer compilierten Sprache wie PASCAL, und auch deshalb war BASIC für größere Programme nicht geeignet. Heute hat man einen intelligenten Weg gefunden, wie man diese Klippe umschifft: Die inkrementelle Compilation. Das heißt nichts anderes, als das man jede einzelne Zeile, wenn man sie eingetippt hat, schon compiliert, also in ein

optimiertes Maschinenformat übersetzt. (Es handelt sich dabei um einen Zwischencode, ähnlich dem, den ein PASCAL-Compiler erzeugt, auch p-code genannt. Dieser ist zwar nicht so schnell, wie ein Maschinencode, aber doch etwa 10mal schneller als interpretierendes BASIC.) Dadurch gibt es noch eine ganze Menge anderer Vorteile.

Sofortige Syntaxprüfung

Der Compiler macht gleichzeitig bei der Eingabe die Syntaxprüfung. Also keine Fehler mehr mit falsch geschriebenen Kommandos, die der Compiler früher immer nur schön nach der Reihe finden konnte, und man nur so, um die Syntaxfehler herauszukriegen, etwa 10mal den ECLG-Zyklus durchgehen mußte. Das Professional BASIC ist sogar so hilfreich, daß es bei einem Fehler obendrein noch eine Liste aller in dieser Situation korrekten Möglichkeiten gibt. Was will man mehr?

Variablen-Anzeige, Cross-Reference

Als Hilfsmittel beim Programm-Debugging sind Cross-Reference-Programme gebräuchlich. Es sind Programme, die das Vorkommen aller Variablen auf allen Seiten des Programm-Listings auffinden und als sortierte Liste ausgeben. Cross-Reference oder ihr Fehlen machten schon den Unterschied, ob man mit einem Programm von 10 Seiten Länge etwas anfangen konnte oder nicht. Wenn irgendeine Änderung im Programm gemacht werden mußte, dann war es unbedingt wichtig, daß man wußte, auf welche Variablen sich diese Änderung erstreckte. Somit war Cross-Reference ein unabdingbares Hilfsmittel beim Programmieren.

Die heutigen BASICs haben nicht nur ein Cross-Reference-Programm, sondern sie haben sie gleich eingebaut (Professional BASIC). Man braucht nur das Programm FIND name zu geben, und schon zeigt der Computer alle Programmzeilen, in denen die Variable name auftritt.

TRACE

Es war ein besonderer Vorzug von einigen früheren BASIC-Versionen, daß man ein TRON-Kommando geben konnte. Damit zeigte das Programm an, welche Zeilennummern es durchlief. Man konnte damit zumindest feststellen, wie der Kontrollfluß des Programms verlief. Allzu komfortabel war TRON nicht, denn man konnte damit noch nicht die wirklich wichtigeren Werte der Variablen erfahren, nach denen das Programm ja seinen Weg nahm.

Heute kann man ein Programm mit Hilfe der TRACE-Möglichkeit in ein paar Minuten testen. Man kann es nicht nur komplett mit all seinen Variablen darstellen, so daß man immer genau weiß, wo das Programm steht und aus welchem Grunde es genau das tut, was es tut, sondern man kann es noch anhalten, selber per Hand die Variablen verändern, und es dann wieder starten. Professional BASIC kann sogar einen Programmablauf rückwärts machen.

Fenster, Prozeduren, Programmierumgebung

Als wenn all diese Neuigkeiten nicht schon genug für eine kleine Revolution des Programmiergeschäfts wären, es kommt aber noch mehr! Man brauchte ja nur das Konzept der LISA weiter umzusetzen und alles, was sich da in Richtung Programmieren verwerten ließ, auch einzusetzen. LISA hat Fenster. Wie kann man Fenster nutzbringend in der Programmierung einsetzen?

Fenster

Ohne ausdauerndes Blättern im Programm-Listing ist heute Programmierung nicht zu machen. Wenn man irgendein Stück Code schreibt, dann bezieht sich dieses Stück Code erfahrungsgemäß immer auf ein paar andere Stücke, die 10 oder 20 Seiten voneinander entfernt im ganzen Listing verstreut sein mögen. Ebenso erfahrungsgemäß hat man gerade kein aktuelles Listing zur Hand, so daß man genötigt ist, erst einmal wieder aus dem Editor herauszugehen, den Drucker anzustellen, die zehn oder zwanzig Minuten zu warten, bis er sich durch die Liste hindurchgequält hat, und dann darf man weiterprogrammieren, nur um nach etwa einer halben Stunde wieder vor demselben Problem zu stehen, denn jetzt hat man so viel verändert, daß das alte Listing nichts mehr taugt.

Mit Fenstern geht es viel einfacher. Da holt man sich die Stücke Code, die man im Augenblick gerade braucht, dazu und schreibt die Routine fertig, an der man gerade arbeitet. Das geht natürlich nur dann einfach, wenn man das oben erwähnte Suchkommando hat, das einem auch schnell

alle Vorkommnisse einer bestimmten Variablen auffindet. Denn sonst wäre das Auffinden aller relevanten Textstellen am Bildschirm allein auch sehr mühselig.

Fenster sind ungeheuer wichtig, wenn man die oben erwähnten Möglichkeiten des TRACE und andere ausnutzen will. Da ja solche Zusatzinformation nicht zum eigentlichen Programmlisting gehört, ist es optisch viel besser, wenn man diese Information abgesetzt präsentiert, eben in einem Fenster.

Prozeduren und Informationsverdichtung

Fenster sind allerdings nicht in allen Fällen eine sofortige Erleichterung für das Programmieren. Wer sich ein BASIC- oder PASCAL-Listing auf dem Bildschirm ansieht, der stellt fest, daß man kaum irgendwelche relevanten Zusammenhänge eines Programms auf einem Schirm darstellen kann. Programmiersprachen stellen gewöhnlich immer nur ein Kommando pro Zeile dar. Bei den heutigen Bildschirmen mit 24 Zeilen kommt da nicht sehr viel heraus. Zumindest ein BASIC löst aber auch dieses Problem: Better BASIC durch echte Prozeduren.

Wenn man nämlich in einer echten Prozedur etwa 20 Zeilen Kommandos auf einen einzigen Namen abkürzen kann (und dieser Name kann 10 oder 20 Zeichen haben, also wirklich aussagen, was diese 20 Zeilen tun), dann kann man auf einmal einen Extrakt von 10 Seiten Programmlisting auf einen Bildschirminhalt bekommen und damit ist man in der Lage, auch sehr komplexe und umfangreiche Programme souverän zu handhaben. Better BASIC erlaubt nicht nur Prozeduren, sondern auch Module. Damit hat man eine gewisse Form eines variablen Compilers, ein Konzept, zu dem später noch etwas gesagt werden soll.

Weitere Eigenschaften

Andere Möglichkeiten der neuen Sprachen sind mächtige Graphik-Befehle, in denen z.B. Kommandos für den Bildaufbau gegeben werden können, wie weiter oben erwähnt wurde, Hilfsfunktionen an jeder Ecke und Kante des Systems, automatische optische Formatierung (pretty printing), integrierter Editor, Datentypen. Die PASCALs der modernen Fertigung sind bis jetzt noch nicht erwähnt worden. Das Mac-PASCAL nutzt alle Benutzer-Eigenschaften des Macintosh aus und ist damit als Programmiersystem ebenso leistungsfähig wie das BASIC, wenn es auch scheint, daß es nicht wesentlich besser ist.

Die neuen Programmiersysteme und das Lernen der Programmierung

Der wirklich einschneidende neue Effekt, der mit den neuen Programmiersystemen zu erwarten ist, ist seine Auswirkung auf die Rate, mit der Menschen Programmieren lernen. Auf einmal hat man die Vorzüge von PASCAL und BASIC zusammen. Sowohl ungeahnte Möglichkeiten der 'Ausprobierens' und ungeheuer verbesserte Bedingungen der Strukturierung eines Programms, dazu noch Hilfsmittel, deren Tragweite noch gar nicht abgeschätzt werden können. Wieviele Menschen würden Fahrradfahren lernen, wenn es erst einen zweimonatigen Trockenkurs bedingen würde, in dem man alles lernen müßte, von der Physik der Kreisel, von der Mechanik der Übersetzung in der Gangschaltung bis hin zu den Kompliziertheiten eines Dreipunkt-Kräftesystems, das der Fahrer und die beiden Räder bilden, - und nur wenn man all diese Lehrstoffe absolviert hat, dann darf man auf ein wirkliches Fahrrad steigen? Dann würde es vielleicht tausend Menschen geben, die jemals Fahrrad fahren lernen.

Dabei ist es ja so einfach. Man lernt alle Gesetzmäßigkeiten der Kreisel, indem man sich für fünf Minuten auf ein rollendes Fahrrad setzt und merkt, daß man nur dann nicht umkippt, wenn man schnell genug rollt. Das Zusammenwirken dieses Dreipunktsystems erfährt man sofort, wenn man das erste Mal um eine Kurve fahren will und instinktiv lernt, daß man sich dabei in die Kurve legen muß, oder man kippt um. Das System als wirkliches Objekt lehrt dem Lernwilligen alles über sich in wenigen Minuten, wozu es in einer akademischen Ausbildung Monate bräuchte, mit einer Chance von eins zu zehn, daß er bei all dem trockenen und uninteressanten Stoff jemals an das Ende seines Kurses kommt. Das ist die Lehre, die die heutigen Computersysteme allen akademischen und universitären Lehransätzen heute geben.

Programmierumgebungen und generische Programmiersysteme

Der Trend geht allgemein weg von den spezifischen Eigenschaften einer bestimmten Programmiersprache, hin zu den Unterstützungsmöglichkeiten, die geboten werden. Und diese haben mit der aktuellen Syntax einer Sprache soviel zu tun, wie die Farbe eines Autos mit seiner PS-Zahl, nämlich gar nichts. BASIC nähert sich an PASCAL an, aber es bietet viel, viel mehr als PASCAL, es übernimmt nur ein paar gute Eigenschaften von dort und schafft etwas völlig anderes. Vielleicht sollte man sich deshalb auch eine neue Namensgebung überlegen, es ist jetzt ein höchst komplexes System, eine Programmierumgebung.

Generische Programmier-Systeme

Die besprochenen Systeme sind alle ganz neu und zum Teil noch nicht auf dem Markt. Das Mac-PASCAL und Mac-BASIC kommen erst in den nächsten Monaten. Deshalb kann man hier noch ein paar Worte über die Zukunftsentwicklung machen. Die Entwicklung wird nun schnell weitergehen. Mit Systemen dieser Art wird sich die Programmierproduktion mindestens verdoppeln, wenn nicht noch wesentlich stärker zunehmen. In ein paar Jahren wird es keine Anwendungsprogrammierer mehr geben, nur noch Systemprogrammierer und Anwender, die sich ihre Anwendungen mit hochleistungsfähigen Generischen Systemen selber schreiben.

Was sind solche generischen Systeme? Die heutige Entwicklung weist schon den Weg. Die aktuelle Syntax einer Sprache verliert mehr und mehr an Gewicht. BASIC und PASCAL haben sich so angenähert, daß es kaum mehr lohnt, eine Unterscheidung zu machen. Spätestens dann wird jemand auf die Idee kommen, daß es überhaupt nicht mehr wichtig ist, welche Syntax ein Programmiersystem hat. Schon heute sind die Unterstützungsmechanismen für diese Systeme ja schon viel größer, als der eigentliche Compiler. Man wird bald darauf kommen, daß man die Syntax vollständig von dem Compiler trennen kann und nur noch als Overlay oder als Datei einziehen kann.

Objektsysteme

Dann haben wir ein System, nennen wir es das 'Generische Computersystem', welches alle Eigenschaften wie oben beschrieben hat und vielleicht noch ein paar mehr. Wollen wir ein Programm in BASIC schreiben, dann sagen wir BASIC. Das System holt uns dann die Syntax dieser Sprache. Sinngemäß für andere Sprachen. Die Wahrscheinlichkeit ist aber mehr, daß man weniger in BASIC oder in PASCAL programmieren wird. Stattdessen wird man einfach die Syntax für 'Lagerhaltung' aufrufen, und man bekommt einen Satz von sehr mächtigen Kommandos, mit dem man als Benutzer allerlei Arten von Programmen für 'Lagerhaltung' aufbauen kann. Oder man ruft analog die Syntax für 'Buchhaltung' u.ä. auf. So wird es Systeme geben, die jeweils Sätze von Kommandos bereithalten, die für einen bestimmten Objektbereich geeignet sind. Wie eine solche Syntax aussehen kann, zeigen heute schon so hochintegrierte Systeme wie DBASE II oder neuere, wie LOTUS usw.

3. FORTH, ein unbekannter Prototyp des Generischen Systems

All dies ist auch in Wirklichkeit keine Zukunftsmusik, sondern existiert schon jetzt. Die Programmiersprache FORTH, heute nur einigen wenigen Spezialisten bekannt, erlaubt das heute schon. Es existieren einige PASCALs und BASICs, die in FORTH implementiert worden sind. Das ist aber eine ganz andere Situation, als wenn heute alles Mögliche in C implementiert ist. C ist ein Compiler, und das fertige Produkt hat nichts mehr mit C zu tun. Ein FORTH-Compiler, mit dem man ein PASCAL implementiert hat, ist aber immer noch vorhanden, und der Benutzer kann, wenn er will, aus dem PASCAL heraus in FORTH weiterprogrammieren.

Die Wege in die Zukunft

Viele Konzepte, die in FORTH existieren, sind durch die neuen Systeme auch in den Allgemeingebrauch gekommen. FORTH verwendet schon seit seiner Entwicklung vor 15 Jahren ein Block-Konzept, das ein Vorläufer der Fenster ist, und das TRACE mit der Darstellung aller Programmelemente ist in FORTH schon immer gebräuchlich. Inkrementelle Kompilation ist in FORTH die Grund-Arbeitsweise, und der Programmierer bewegt sich dort in einer kompletten Programmier-

umgebung bestehend aus Editor, Compiler und Massenspeicherverwaltung. Einige Konzepte aus FORTH warten aber noch auf ihre Entdeckung in der allgemeinen Welt der Programmierung, und die sind:

Hierachische Strukturierung. Die Strukturierung in PASCAL ist nur ein Anfang, den man in Richtung auf logische Gliederung seines Programms machen kann. Konzepte der mathematischen Logik, wie die Russel'sche Theorie der Typen, können weitreichend und profitabel eingesetzt werden. Diese Möglichkeit ist aber nur mit einer mächtigen und leicht handhabbaren Arbeitsweise von Prozeduren gegeben. Das ist bisher nur in FORTH möglich. In FORTH kann man auch neue Operatoren definieren. So wie man Datenstrukturen aufbauen kann, so kann man Operatorenstrukturen aufbauen. Durch diese Konstrukte erreicht man eine zehnfache Informationsdichte des Programms. Es ist normalerweise nicht möglich, in einer Sprache wie PASCAL oder BASIC einen relevanten Teil der Programmablaufsteuerung auf einen Computerscreen zu bekommen. Die Code-Teile, die auf einen Screen passen, sind rein lokal, aber die Übersicht über das ganze Programm ist nicht auf einem Schirm darstellbar. Dies geht nur, wenn man hochkomplexe Zusammenfassungen wie in FORTH machen kann (/1/, /2/). Ein Programm von 40 Seiten Programmlisting in BASIC oder PASCAL paßt typischerweise auf vier Seiten FORTH-Listing.

Intelligenter Assembler. Dieses Problem steht im Zusammenhang mit dem weiter unten erwähnten Problem der Transparenz. Wie gut man ein Problem in einer high-level Sprache auch strukturieren kann, es nützt nichts, wenn man auf eine Schranke von Zeit und Computerspeicher stößt und wenn das Programm um eine Größenordnung optimiert werden muß (zehnmal schneller oder zehnmal kleiner, wie es z.B. der Fall mit der LISA-Systemsoftware ist, die dort zunächst 512 K groß war und nach der Optimierung für den Macintosh in 64 K paßte und zudem noch schneller war. Der Erfolg des Macintosh hing sehr davon ab, ob die Leistungen der LISA, die extrem maschinen- und speicheraufwendig waren und ebenso teuer, auf die billigere Maschine gebracht werden konnten). Die kann bisher nur durch Hand-Assemblierung gemacht werden, und das ist wieder 'schmutziges Geschäft'. FORTH bietet hier, weil es Strukturierung auf ein wesentlich niedrigeres Maschinenniveau bringt, die Möglichkeit, sich den Produktivitätsvorteil der Strukturierung auch in solchen Anwendungen zu bewahren. Die Situation ist heute, daß man versucht, hochkomplexe intelligente Programme auf die Basis von 'dummen' Assemblern zu bauen. Das ist eine höchst fragliche Strategie. Die erste Aufgabe der Systemprogrammierung ist ein 'intelligenter' Assembler. FORTH stellt ein Modell für einen intelligenten Assembler. Irgendetwas anderes als Basis für komplexe Systementwicklungen zu nehmen, heißt eine Zeitbombe in das System einzubauen und zu hoffen, daß das System veraltet ist, bevor man auf die Idee kommen könnte, es zu erweitern oder zu verbessern. Wer sieht heute, daß die Microcomputerindustrie ihre Programme wesentlich verbessert? DBASE ist über vier Jahre gleich geblieben und hat höchstens kosmetische Änderungen erfahren. WORDSTAR hat von Release zu Release kaum irgendwelche Verbesserungen dazubekommen, obwohl es der Firma inzwischen etwa 50 Mio. Dollar eingebracht hat. Kann das vielleicht daran liegen, daß niemand mehr bei Micro Pro auch nur weiß, wie das Programm funktioniert?

Iterative Compilation. Das ist etwas anderes als die oben erwähnte inkrementelle Compilation. Ein Compiler wandelt die vom Menschen verstehbaren Kommandos einer Programmiersprache in maschinenverwertbare Anweisungen um. Nun kann der Compiler entweder auf Geschwindigkeit oder Zeit-Effizienz optimiert sein. Meistens auf Zeit. Es ist aber so, daß ein Programm, das zeiteffizient ist, meist mehr Platz bracuht, als ein anderes, das nicht speziell zeitoptimiert wurde. Andererseits verursachen in einem Programm meist nur 10% des Codes 90% der Laufzeit. Vor allem heute in den Benutzer-interaktiven Programmen kommt es daher kaum auf Zeiteffizienz an. Heutige Programme sind sehr groß, eben weil viel Benutzerunterstützung gegeben wird, und der meiste Platz ist verschwendet, weil eben normal compiliert wird. In FORTH ist es möglich, ein Programm ersteinmal auf die platzsparendste Weise zu compilieren und dann bestimmte Routinen auf Geschwindigkeit zu optimieren. Dadurch kann man etwa 50% Platz in einem Programm sparen, und Computerspeicher ist eine Ressource, die, obwohl immer billiger, nicht unbegrenzt zur Verfügung steht. Die nächste Generation der Compiler wird Optionen geben, ob man lieber etwas auf Zeit oder auf Platz optimiert compilieren will.

Erweiterbarkeit. In FORTH ist man seit 15 Jahren an ein Konzept gewöhnt, das sich in anderen Programmiersprachen nur sehr rudimentär findet. Was tut man, wenn man ein bestimmtes Subprogramm immer wieder braucht, so daß man es eigentlich nicht mehr immer wieder mit einer library dazulinken will? Man würde es gerne in seinen Compiler aufnehmen. Das geht aber nur in FORTH. Dort kann man seinen Compiler beliebig ausbauen. Eine neue Sprachsyntax, wie oben erwähnt, wäre in FORTH durch eine Spezialerweiterung, ein neues Vokabular zu bewerkstelligen.

Transparenz. In FORTH ist der Compiler bis ins letzte dokumentiert. Jeder der will, ist in der Lage, bis in die tiefsten Einzelheiten von FORTH vorzudringen. Es wird ihm leichtgemacht, weil FORTH in FORTH geschrieben worden ist. Wenn man das Sourcelisting eines anderen Compilers hätte aber keinen Assembler versteht, ist man dann wenig glücklich dran. Aber meist bekommt man das Sourcelisting nicht. Die meisten Anwender wollen ja auch nichts damit zu tun haben, sie wollen ja nur programmieren. Allerdings sind sie dann öfter einmal darauf angewiesen, sich in teuren Ferngesprächen mit dem Hersteller des Compilers zu unterhalten, weil das gute Ding eine Macke aufweist, über deren Vorhandensein dem Handbuch nichts zu entnehmen ist. Auch heute noch ist die technische Dokumentation der Programme gerade so schlecht, wie es der Hersteller sich vor seiner Kundschaft noch erlauben kann. Und da es ja zum Berufsethos eines Informatikers gehört, sich durch unleserliche, hiero- und akro-glyphische Dokumentationen durchzuquälen, kann er sich in dieser Hinsicht viel erlauben.

Insofern scheint sich die Anwendungsprogrammierung über die nächsten Jahre auf sehr einschneidende Veränderungen einstellen zu dürfen. Die Systemprogrammierung wird davon nicht verschont bleiben. Da man ja in der Systemprogrammierung Systeme haben muß, die eine Größenordnung schneller und effizienter sind, als die der Anwendung, hat man da natürlich nicht mehr soviel Spielraum mit Hilfsmitteln. Es ist aber zu erwarten, daß auch hier interpretierte Systeme vordringen werden und Ansätze wie der von FORTH weitere Verbreitung finden werden.

Literatur:

/1/ Goppold, A.: Was ist FORTH?, FORTH Gesellschaft, Hamburg, Juni 1984
/2/ Goppold, A.:Das Paradigma der interaktiven Programmierung, MICRO MAG Nr. 37, 1984

□○□

Roland Löhr

'Ein-Chippern' mit R6511Q

1. Übersicht

Einchip-Mikroprozessoren gewinnen immer größere Bedeutung. Sie vereinigen in einem einzigen Gehäuse mit typisch 28, 40 oder 64 Pins alle Einheiten, die einen vollständigen Mikrocomputer ausmachen: Die Zentraleinheit, Takterzeugung, Arbeitsspeicher (RAM), Timer/Counter und Ports zur Abfrage/Steuerung der Außenwelt. In manchen Fällen treten Wandler für A/D und D/A sowie Portleitungen und Register für den seriellen Datenverkehr hinzu. In Heft 27 von Oktober 1982 stellten wir bereits die Motorola-Typen 6805/68705 vor.

Einchipper führen zu einem sehr platzsparenden Aufbau: Auf der Platine sind nur sehr wenige Leiterbahnen aus der CPU herauszuführen, was auch Einfluß auf die Aufbaukosten hat. - Bei den Einchippern gibt es verschiedene Philosophien für den Entwickler: Er kann z.B. in einer Emulation wie in einem Zielsystem arbeiten, das später mit einer kundenspezifischen ROM-Version on chip in größerer Stückzahl versehen sein soll. Oder er kann ein Zielsystem entwickeln, das mit externen Festwertspeichern (EPROM) arbeitet, die entweder über den Bus/mit Portpins gemultiplextem Bus oder mit Huckepack-EPROM's zusammenarbeiten. Ein gemultiplexter Bus oder ein ziemlich voller Bus sehen dann auch mehr RAM und Interfaces vor, als man sie 'on chip' findet. Die erwähnten 68705 haben häufig ein Ur-Einbrenn-ROM und daneben EPROM, in das das kundenspezifische Festwertprogramm eingebrannt wird. - Die Möglichkeiten sind heute bereits so vielfältig, daß es nicht Wunder nimmt, wenn die hier gegebenen Möglichkeiten intensiv genutzt werden, was zu einer Materialverknappung führt.

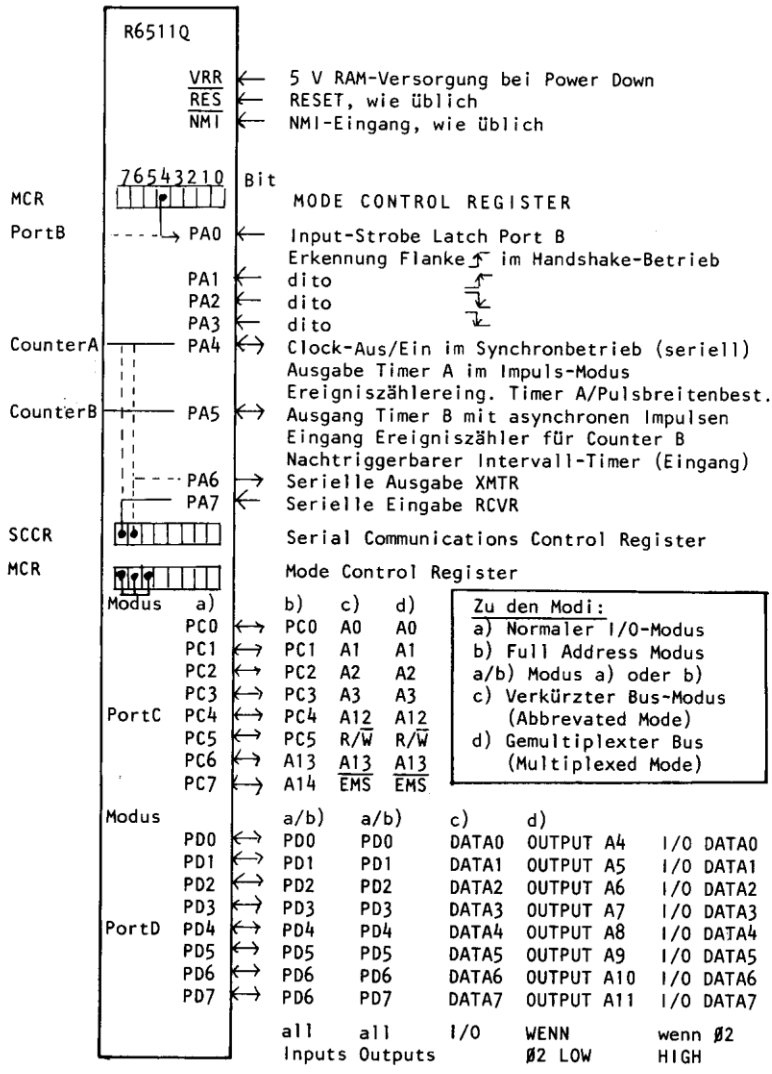


Abb. 1: Pins des R6511Q mit Besonderheiten

Wir besprechen hier die Einchipper von Rockwell R6501Q und R6511Q. Ersterer hat pull up-Widerstände an den Ports A, B und C, der zweite hat dort open collector. Die Aussagen für diese Typen lassen sich auch auf den FORTH-Prozessor R65F12 übertragen, der offensichtlich beim Reset sich nur in der Initialisierung des seriellen Ports unterscheidet. Dann gibt es noch den gleichartigen R6500/13, der ein kundenspezifisches ROM von 256 Byte enthalten kann.

Diese Prozessoren haben alle einen gegen die CPU 6502 erweiterten Befehlssatz mit den Bit-spezifischen Befehlen für die Zero Page, nämlich 'Setze Memory Bit' (SMB), 'Lösche Memory Bit'

(RMB), 'Branch on Bit SET/Clear' (BBS bzw. BBR). Nicht enthalten sind die weiteren Befehle der CMOS-CPU R65C02 für JMP, BIT, TSB, STZ, PHX, PHY, PLX, PLY usw.. Der Befehlssatz der Einchipper ist daher typisch für die Möglichkeiten der Steuerung von Interfaces und ihrer Register erweitert, die in der Zero Page adressiert werden. Dort liegt nebenbei auch der Hardware-Stack.

2. Architektur und Besonderheiten des R6511Q

Im Bereiche der CPU finden wir die gleichen Register für den Anwender wie beim 6502, nämlich Akku, die Indexregister X und Y, den Stackpointer, das Statusregister und den 16 Bit breiten Programmzähler. Der Stackpointer zeigt auf Adressen in der Zero Page und muß vom Benutzer nach einem Reset per Programm initialisiert werden.

Das Gehäuse ist vom Typ 64 Pin Quip. An ihm finden wir folgende wichtige Pin-Gruppen: Datenbus, Adreßbus mit A0-A12 und A15 und die vier Ports A, B, C und D. Auf dem Chip haben wir ferner zwei 16 Bit breite Counter/Timer A und B sowie 192 Byte RAM, die in der Zero Page adressiert sind (dort liegen auch die Interfaces und Steuerregister). Hinzu kommen Pins und Aggregate für die Takterzeugung und Buskontrolle.

Abbildung 1 weist auf Pins und ihre Signale hin, die Besonderheiten aufweisen. Zunächst ist zu sagen, daß die Ports A, B und C im 'Normal Mode' in herkömmlicher Art gefahren werden können. Dabei kann jeder einzelne Pin individuell als Ein- oder Ausgang programmiert werden. Port D kann dann mit allen 8 Pins entweder nur Eingang oder nur Ausgang sein.

Für Port B gilt, daß er immer als reiner E/A-Port gefahren wird. Sofern er Inputs enthält, ist eine kontrollierende Eintaktung (Strobe) der Signale möglich, wenn das Mode Control Register (MCR) in Bit 4 zu '1' gesetzt wird. Der Strobe-Impuls muß dann auf den Pin PA0 geleitet werden. Kontrolle dieser Betriebsart durch das MCR.Bit 4. Das sind die einzigen Besonderheiten hier.

Die Pins des Port A können abweichend vom Portbetrieb Sonderfunktionen im Zusammenhang mit der Impulsflanken-Erkennung im Handshake-Betrieb übernehmen (PA0 bis PA3), Pin PA4 kann mit mit dem Counter/Timer A zusammenarbeiten und auch mit der seriellen Synchronen (getakteten) Datenübertragung (in die auch Timer A eingeschaltet ist, ferner einer oder beide Kanäle der seriellen Datenübermittlung an PA6 und PA7), Pin PA5 kann mit dem Counter/Timer B zusammenarbeiten (Ereigniszählung/asynchrone Impulserzeugung).

Abweichend vom Portbetrieb können die Pins PA6 und PA7 als serielle Ausgabe/Eingabe benutzt werden, wenn das Serial Communication Control Register entsprechend gesetzt ist. Dieser Artikel geht mit wenigen Beispielen in den weiteren Abschnitten auf die Programmierung ein. Ein weiterer Artikel in diesem Heft versucht, die Programmierung mit Interface-Worten unter Forth zu erklären, von denen auf die entsprechende Assembler-Programmierung zurückgeschlossen werden kann.

Nachfolgend soll besonders auf die Betriebsarten im Überblick eingegangen werden.

Der bisher erwähnte 'Normal Mode' erlaubt es, bis zu 16 KB externen Speicher/Interfaceadressen anzusprechen, weil 'normal' nur die Adreßleitungen A0-A12 und A15 eine feste Pinbelegung haben, auf der die Signale ausgesendet werden. Es sind jedoch noch drei weitere 'Bus Modes' möglich.

Sollen 64 KB Speicher angesprochen werden, dann geht es um den 'Full Bus Mode'. Er wird im Kontrollregister 'Mode Control Register' (MCR) eingerichtet. In diesem Fall stehen die Portpins PC6 und PC7 nicht für die Ein- und Ausgabe zur Verfügung, sie werden stattdessen umdirigiert und senden die Signale A13 und A14 aus, so daß ein 'voller Bus' zur Verfügung steht.

Im 'verkürzten Busmodus' (Abbrevated Bus Modus) kann man bis zu 4 Interfacebausteine an definierten Adressen ansprechen (sie dürfen je bis zu 16 Register/Port haben, wie z.B. die VIA 6522). Außerdem können nötigenfalls bis zu 32 KB externes RAM angesprochen werden. In diesem Fall übernimmt Port D den bidirektionalen Datenbus und an Port C werden Adreß- und Steuersignale ausgesandt, so daß auch hier eine Umsteuerung der Pinfunktionen eintritt.

Schließlich gibt es noch den gemultiplexten Bus-Modus (Multiplexed Bus Mode). Auch hier ist Port D in Sonderfunktion. Während der Takt Phi2 LOW ist, sendet er Adreßbusssignale. Auch die Pins, die 'normalerweise' für Port C bereitstehen, senden dann Adreß- und Steuersignale. Insgesamt können in diesem Modus einmal bis zu 16 KB und dann außerdem bis zu 32 KB externe Adressen in Speichern oder Interfaces angesprochen werden.

Die vier Konfigurationsmöglichkeiten (Bus Modes) mögen auf den ersten Blick etwas verwirrend sein. Hier ist es Aufgabe des Designers, das Datenblatt sorgfältig zu studieren. Die Pins haben also durchaus unterschiedliche Aufgaben (sie sind nicht nur einfach 'Port' oder 'Adreßbus'. Ihre Funktion wird wesentlich durch das 'Mode Control Register' und durch das 'Serial Control Register' durch das firmwaremäßige Einschreiben in dieselben bestimmt (Initialisierungsroutine). Jedoch: Die Zuweisung von Funktionen an die Pins beeinflußt auch die notwendige Verdrahtung auf der Platine. Man wird wohl sagen dürfen, daß die Designer dieses Einchippers zwischen 'sparsam' und 'opulent' einen sehr wandlungsfähigen Systemaufbau ermöglicht haben. Man muß sich als Anwender jedoch auch vor Augen halten, daß die verschiedenen 'Bus Modes' zu einer jeweils verschiedenen Adressierung der externen Speicher- und I/O-Bereiche führen. Ein Tausch der Pferde mitten im Ritt wird nach den anfänglichen Festlegungen begrenzt möglich sein.

Zu weiteren Besonderheiten: Die Signale RESET und NMI (aktiv LOW) arbeiten wie üblich. Wir finden keinen Eingabepin für IRQ. Solche Interrupte können nur von den Baugruppen des Einchippers ausgehen, es sei denn, man würde einen der Pins PA2 oder PA3 interruptfähig machen und ein IRQ-Signal von draußen auf diese zuleiten.

3. Ports, Sonderregister und RAM in der Zero Page

Die Ports und Register sind im Bereich hex 00 bis 1F dekodiert. Dafür die nebenstehende Übersicht. Daran schließt sich ein nicht belegter Adressenbereich an. Von hex 40 bis FF liegt dann das interne RAM des Einchippers. Hier befindet sich auch der vom Anwender zu initialisierende Hardware-Stack, bei den FORTH-Einchippern ferner der Datenstack ab Adresse C2 abwärts.

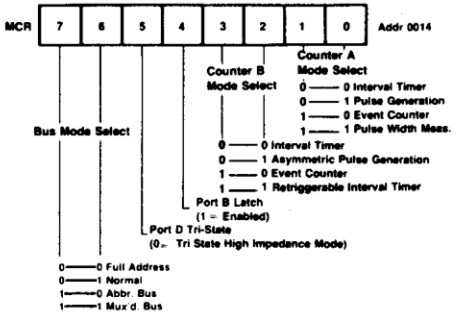
Adresse	READ	WRITE-Funktion
001F	--	--
001E	Lower Counter B	Upper Latch B a) b)
001D	Upper Counter B	Upper Latch B
001C	Lower Counter B b)	Lower Latch B
001B	--	--
001A	Lower Counter A	Upper Latch A a) b)
0019	Upper Counter A	Upper Latch A
0018	Lower Counter A b)	Lower Latch A
0017	Ser. Receive Data Register	Ser. Transmit Data Register
0016	Serial Status Register	Serial Status Register c)
0015	Serial Control Register	Serial Control Register
0014	Mode Control Register	Mode Control Register
0013	--	--
0012	Interrupt Enable Register	Interrupt Enable Register
0011	Interrupt Flag Register	--
0010	Read FF	Clear Interrupt Flag d)
000F bis 0007	frei für Anwender	
0006 bis 0004	reserviert für die 3 zusätzlichen Ports des R6500/12	
0003	I/O Port D	dito
0002	I/O Port C	dito
0001	I/O Port B	dito
0000	I/O Port A	dito

a) Lade und Starte Counter
b) Clear Flag-Adresse
c) Nur Bits 4 und 5
d) Nur Bits 0-3

4. Funktionskontrolle durch das Mode Control Register (MCR)

Das Mode Control Register beherrscht den Einsatz der Timer A und B sowie den Einsatz der ihnen ggfs. zuzuteilenden Pins PA4 und PA5 in Sonderfunktion, ferner die Pins der Ports C und D in ihrer Rolle als Portpins oder als erweitertes Bussystem (auch gemischt). Es beherrscht weiterhin das Input-Latching an Port B mit Strobe an Pin PA0. Es ist damit das wichtigste Register des Prozessors. Eine untergeordnete steuernde Wirkung haben daneben nur noch das Serial Communications Control Register (wenn serieller Datenverkehr gewünscht wird) und das Interrupt Enable Register (wenn Interrupte zugelassen werden sollen).

Bei einem Reset wird das Interrupt Flag im Status gesetzt. Das Mode Control Register wird zu 00 zurückgesetzt, der Hardware-Stackpointer ist undefiniert. Es liegt also in der Sorgfalt des Programmierers, das MCR zu initialisieren, ferner den Stackpointer durch LDX #$FF und TSX. Daran wird sich die Initialisierung anderer Register anschließen. Diese Sequenz ist mit CLI abzuschließen, von wo es in die Hauptprogrammschleife weitergeht.



Mode Control Register

5. Die Interruptsteuerung des IRQ

Zum IRQ-Interrupt können nur Quellen auf dem Chip zugelassen werden, und zwar die Pins PA0 bis PA3 in ihrer Funktion zur Erkennung von Impulsflanken, ferner die beiden Counter/Timer bei Unterlauf sowie jeder der Kanäle Receiver und Transmitter für die serielle Datenübertragung, letztere in Abhängigkeit von Bedingungen im Serial Status Register, z.B. Datenregister leer oder Paritätsfehler, Overrun.

Drei Register steuern den IRQ. Das Interrupt Enable Register (IER) ist ein Schreib-/Leseregister. Ein hier zu '1' gesetztes Bit läßt von korrespondierender Quelle und Bedingung den Eintritt in die Interruptroutine zu. Das ist wie beim IER der VIA 6522. Dieses Register kann nicht nur mit der Sequenz LDA ... STA IER initialisiert werden, es kann auch mit den Befehlen Set oder Reset Memory Bit gearbeitet werden, natürlich auch im Programmverlauf, z.B.:

```
IER=$12  ; Interrupt Enable Register
SMB0 IER ; PA0 Interrupt Enable bei aufsteigender Impulsflanke
...
RMB0 IER ; Disable Interrupt von PA0
```

Nach einem IRQ-Interrupt wird man ein Polling durchführen müssen, wenn es mehrere zugelassene Interruptquellen gibt. In der Interruptroutine wird man die Verzweigung in der Reihenfolge der zeitlichen Dringlichkeit durchführen. Ein Interrupt setzt im Interrupt Flag Register (IFR, Adresse 11) ein korrespondierendes Bit zu '1'.

Wir gehen einmal davon aus, daß der serielle Receiver immer als erster auf Interrupt abgefragt werden soll, dann eine mögliche Impulsflanke an PA0 usw.. Unter Anwendung der Befehle Branch on Bit Set wäre dann in der Interrupt-Routine z.B. wie folgt zu programmieren:

```
IFR=$12

        IFR=$12
INTERR  PHA          ;RETTE AKKU
        BBS6 INTRCV ;WENN RECEIVER
        BBSØ


        IFR=$12
INTERR  PHA                 ;RETTE AKKU
        BBS6 IFR INTRCV     ;VERZWEIGE, WENN RECEIVER-BIT IN IFR GESETZT
        BBSØ IFR INTPAØ     ;DITO, WENN PAØ-BIT DORT GESETZT IST
        ...
INTRCV  ...                 ;SERVICEROUTINE
        RMBX SCSR           ;ENTFERNE INTERRUPT FLAG
        ...                 ;JMP ODER BRANCH NACH INTOUT
INTPAØ  ...                 ;SERVICEROUTINE
        ...
INTOUT  PLA                 ;RESTORE
        RTI                 ;VERLASSE INTERRUPTROUTINE
```

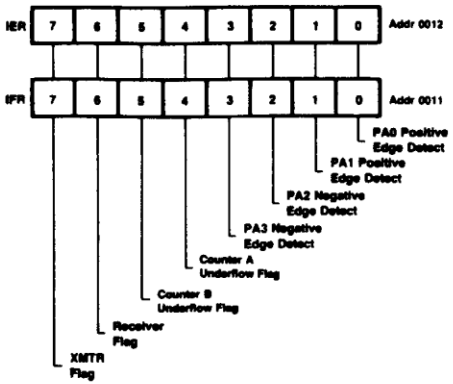


FIGURE 3-6. Interrupt Enable and Flag Registers

Wie wird nun ein Interrupt-Flag nach der Interruptbehandlung gelöscht? Die anzuwendenden Befehle sind unterschiedlich. Für die Interruptbits 0-3 der Portpins PA0 bis PA3 gilt, daß man auf das Write Only-Register bei Adresse 0010 einen Befehl Reset Memory Bit anwenden muß, also RMB0 bis RMB3. Diese Befehle transportieren in die Adresse 11 (das IFR) und löschen dort das entsprechende Bit. - Das Löschen der für die Counter/Timer gesetzten Interruptbits geht aus der Tabelle in Abschnitt 3 hervor: Lesen der unteren Counter-Hälfte oder Schreiben in das obere Latch. Bei Interrupts von den seriellen Kanälen werden die Flags gelöscht, indem man im Serial Communications Status Register (SCSR) in den Bits 0-3 oder 6-7 löscht, was dann in das IFR durchkopiert.

Der Einchipper hat keinen Eingangspin für IRQ. Sofern weitere externe Interfaces zum Interrupt gelangen sollen, kann man deren IRQ-Ausgang auf einen der Pins PA0 bis PA3 in entsprechender Beschaltung zuleiten.

6. Die Bedienung der Ports

Die in den vorstehenden Abschnitten erwähnte mögliche Umsteuerung von Pinfunktionen an Port C und D betrifft in erster Linie den Designer einer Platine. Der Anwender braucht eigentlich nur zu wissen, welche Portpins ihm zur Verfügung stehen, und natürlich auch, wo sich die Speicheradressen befinden.

Die Bedienung der Ports ist anders als z.B. bei der VIA 6522 oder der PIA 6520. Es gibt keine Datenrichtungsregister. Das was man in den Port hineinschreibt, das wird gesendet. Bei Lesevorgängen wird der tatsächliche Pegel High oder Low am Port gelesen. Um einen Port insgesamt zum Imput zu machen, beschreibt man ihn mit hex FF. Eine externe Signalquelle hat jetzt die Möglichkeit, einzelne Pins auf LOW zu ziehen, was dann am Port gelesen werden kann. Gleiches gilt, wenn man nur einzelne Pins/Bits zu Eingängen machen will. Diese Aussage betrifft die Ports A, B und C. Für Port D gilt im 'normal Mode', daß er mit allen 8 Pins entweder nur Eingang ist oder nur Ausgang. Das wird durch das Mode Control Register (MCR) in Bit 5 gesteuert. Ist es 0, so ist der Port Input, mit 1 ist er Output.

Im Nachgang und zur Ergänzung zu Abschnitt 2 mit den Sonderfunktionen der Pins: Bei seriellem Datenverkehr sind an Port A entweder die Pins PA6 oder PA7 oder beide belegt, je nach beabsichtigtem Modus. Bei synchroner (getakteter) Datenübermittlung ist außerdem PA4 als Ein- oder Ausgang für den Takt belegt, in allen Fällen auch der Counter/Timer A.

Pin PA4 ist weiterhin für den Counter/Timer A in verschiedenen Sonderfunktionen belegt, auf die bei den Timern eingegangen wird.

Der Pin PA0 kann Hilfsfunktionen für den Port B übernehmen, wenn er den Strobe für die an Port B ankommenden Daten übernimmt. Sie werden dann gelatcht (zwischengespeichert wie zum Zeitpunkt des Strobe entgegengenommen). Das kann Überwachungsroutinen für Ereignisse entbehrlich machen und eine spätere Auswertung des Zustandes an Port B ermöglichen.

Die Pins PA0 und PA1 können von Low zu High aufsteigende Impulsflanken erkennen, die Pins PA2 und PA3 eine abfallende. Sie sind damit für den Handshake-Betrieb vorgesehen. Der Pin PA5 kann wahlweise mit dem Timer/Counter B zusammenarbeiten.

In der Summe läßt sich damit sagen, daß die Pins des Port A besonders für Steuerfunktionen im Handshakebetrieb, bei der Ereigniszählung oder bei der Impulsausgabe vorgesehen sind, natürlich auch für den seriellen Datenverkehr.

Port B ist immer in reiner Portfunktion einsetzbar, wobei jeder Pin individuell als Eingabe oder Ausgabe benutzt werden kann.

Je nach Konfiguration des Systems im Mode Control Register sind die Pins des Ports C entweder Portpins oder sie übernehmen Funktionen des Adreß- und Steuerbusses, auch gemischt.

Port D wird mit allen Pins entweder als Eingabe- oder als Ausgabeport gefahren. Steuerung im MCR, Bit 5. Je nach Systemkonfiguration im MCR kann Port D aber auch Funktionen eines Datenbusses übernehmen oder gemischt auch Datenbus und Ausgabeport.

Das Interruptverhalten der Pins PA0 bis PA3 wurde schon im 5. Abschnitt kurz angesprochen: Eine entsprechende Flanke an den Pins setzt ein korrespondierendes Bit im Interrupt Flag Register. Hier kann es auch im Vordergrundbetrieb abgefragt werden, ohne daß das Ereignis zum Interrupt zugelassen wurde. Typisches Beispiel z.B. bei einer Centronics-Schnittstelle mit der Abfrage im Hauptprogramm, ob der Drucker das Signal ACK zur Datenübernahme abgegeben hat.

7. Der Betrieb der Counter/Timer A und B

Der R6511Q und seine Verwandten haben zwei 16 Bit Breite Counter/Timer mit vorgelegtem Latch (Vorspeicher) in gleicher Breite. Der Counter B hat sogar zwei vorgelegte Latches für die aperiodische Impulserzeugung. Davon später. Der Betrieb weicht von dem der Timer auf der VIA 6522 etwas ab.

Der Counter/Timer A ist fest zugeordnet (und damit 'verbraten'), wenn serieller Datenverkehr eingerichtet ist. Dieser Fall wird in einem späteren Abschnitt angesprochen. Wenn diese Datenka-

näle nicht eingerichtet werden, dann ist der Betrieb als reiner Timer (Interval Timer), als Generator von symmetrischen Impulsen auf Pin PA4, als Ereigniszähler an PA4 und als Pulsweitenmesser ebenfalls an PA4 möglich. Die entsprechende Einrichtung erfolgt in den Bits 0 und 1 des Mode Control Register (MCR). Man beachte gem. Abschnitt 3, daß das Starten des Timers/Counters durch einen Speicherbefehl in Adresse 001A bewirkt wird.

Eine Erprobung dieser Möglichkeiten ist nicht möglich, solange man den Computer am seriellen Kanal hat, um ihm Befehle zu übermitteln. Man würde Timer A hinsichtlich der eingestellten Baudrate umwerfen und auch die Transmit-/Receive-Funktion ausschalten. Es gibt jedoch die Möglichkeit, die Maschine mit einem Maschinenprogramm umzustellen, aus dem sie hinterher nur noch mit einem Reset zurückgeholt werden kann. Dieser Vorschlag nur zur Erprobung, wobei man sich ggfs. die Register hilfsweise und vorübergehend auf einen Port zusätzlich ausschreiben könnte:

```
CODE TEST       ; Assembler hereinrufen für das Wort TEST
0 # LDA,
15 STA,         ; hex 15= Serial Communications Control Register, Disable Serial Comm.
1 # LDA,        ; Richte Impulsausgabe an PB4 ein (symmetrisch)
MCR STA,        ; in das Mode Control Register
FF # LDA,       ; Schreibe in das Latch Low und High des Timers
18 STA,         ; Lower Latch
1A STA,         ; Upper Latch und Start Timer
END-CODE
```

Nach dem Aufruf des Wortes TEST sind wir dann zwar von weiteren seriellen Datenverkehr ausgeschlossen, können aber auf dem Oszilloskop an PA4 den symmetrischen Ausgangsimpuls beobachten.

Der Counter/Timer B kann in folgende Betriebsarten gerufen werden, wenn man das MCR entsprechend in den Bits 2 und 3 setzt: Reiner Intervall-Timer, Assymmetrischer Pulserzeuger am Pin PA5, Ereigniszähler an PA5 oder Überwachungs-Timer für PA5 als Eingang (Retriggerable Interval Timer). Im letzten Fall wird der Counter/Timer immer wieder nachgeladen, wenn am PA5 ein Impuls rechtzeitig eintritt. Wird die Zeit überschritten, so wird das Interrupt Flag gesetzt.

Mit der möglichen asymmetrischen Impulserzeugung und den beiden Vorspeichern des Timers B hat es folgende Bewandnis: Das sog. Latch C hat keine eigenen Adressen. Das Laden geschieht durch die Reihenfolge der Beschickung. Zunächst schreibt man die Pulsbreite Low und High in die Adressen 1C und 1D (Latch B). Danach lädt man den zeitlichen Abstand zwischen den Impulsen mit dem niederwertigen Teil nach Adresse 1C und mit dem höherwertigen nach 1E, womit das 'Durchladen' geschieht. Wenn man jetzt Bit 2 im MCR zu '1' setzt, dann hat man am Pin PA5 den Ausgang einer asymmetrischen Rechteckschwingung, die sich ohne weiteres Zutun der CPU selbständig wiederholt.

8. Der serielle Datenverkehr

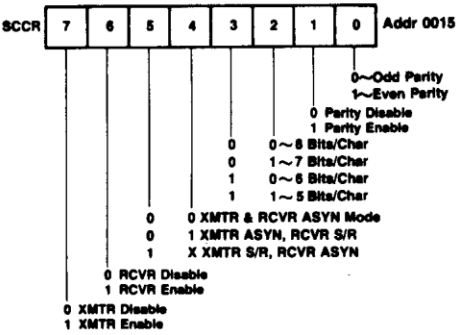
Die Ausgabe serieller Daten an Pin PA6 und der Empfang an PA7 wird im Serial Communications Control Register in den Bits 7 und 6 eingerichtet (zu '1' setzen). Hier wird auch verankert, ob die Übermittlung beidseitig asynchron oder einseitig asynchron und auf dem anderen Kanal synchron erfolgen soll. Bei asynchronem Verkehr wird ferner verankert, wieviele Bits pro Zeichen gesendet werden, ob Paritätsprüfung stattfindet und wenn ja, auf gerade oder ungerade. Bei der synchronen Datenübermittlung werden immer 8 Bits gesendet/empfangen. Pin PA4 ist dann als Clock-Ausgang oder Eingang fest zugeordnet. Bei 1 MHz Taktfrequenz ist die maximale Übertragungsrate dann 62,5 KBits/Sekunde.

Bei seriellem Datenverkehr ist Timer A fest zugeordnet.Er ist entsprechend der gewünschten Baudrate zu initialisieren. Das Datenblatt gibt dazu Auskunft. Dem Datenverkehr dient ferner das Serial Communications Status Register in Adresse 16. Wenn hier ein volles Datenregister beim Empfangen oder ein leeres beim Senden festgestellt wird, so wird das hier angezeigt, zugleich mit dem Setzen des Interrupt-Flags im IFR. Ebenso bei Ausnahmezuständen. Die empfangenen Daten

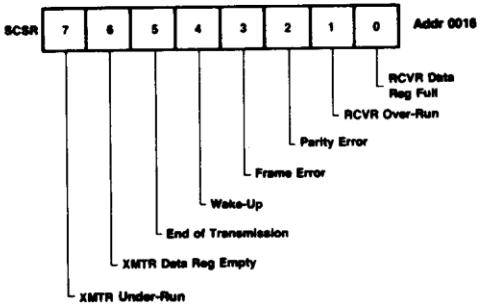
werden in Adresse 17 Serial Receiver Data Register gelesen. Zu sendende Zeichen werden in die gleiche Adresse geschrieben, die dann als Serial Transmitter Data Register dient.

9. Zusammenfassung

Mit den vorstehenden Ausführungen konnten nur die wichtigsten Eigenschaften der Einchipper angesprochen werden. Es war dabei die Absicht, die Zusammenhänge aufzuzeigen, die bei den einzelnen Betriebsarten bestehen. Es sollte klar geworden sein, daß diese Prozessoren für Betreiber, die sich in 65xx und 68xx auskennen, keine besonderen gedanklichen Schwierigkeiten bereiten. Die meisten Funktionsweisen sind nach vertrauter Art eingerichtet. Damit ist der Anreiz gegeben, die Prozessoren für einen sehr kompakten Aufbau vor allem von Steuerungen heranzuziehen. Besonders interessant wird es, wenn man zu ihnen bereits auf ein E/A-System zurückgreifen kann, das die Hantierung und Erprobung erleichtert. Das Minimum wird damit im allgemeinen ein Monitor sein, der mit den seriellen Kanälen zusammenarbeitet. Noch bequemer ist es, wenn man einen FORTH-Kernel zur Verfügung hat, der zugleich auch das E/A-System enthält. Das ist der Fall bei der in Heft 37 besprochenen Entwicklungsplatine und bei der in diesem Heft auch besprochenen Minibee-Karte MB6511, die hier in Interface-Versuchen betrieben wird.



Serial Communication Control Register



SCSR Bit Allocations

Roland Löhr

Ein FORTH-Computer

und Interface-Worte in FORTH

1. Der Computer

Der Autor hatte Gelegenheit, den FORTH-Computer Minibee MB6511 der Brühl Elektronik in Nürnberg zu testen. Die Zentraleinheit auf der 1/2 Europa großen Platine ist ein R6511A (2 MHz) im 64poligen Quip-Gehäuse. Dieser Einchipper mit seinen vielen Möglichkeiten vor allem auch für das Interfacing ist an anderer Stelle in diesem Heft besprochen worden.

Neben der CPU finden wir zwei 28polige Stecksockel, auf die nach Bedarf 24- und 28polige Festwertspeicher und RAM's verschiedener Typen gesteckt werden können. Der Prozessor ist auf 'Full Address Mode' geschaltet, so daß insgesamt 64 KB adressiert werden können. Im konkreten Fall handelte es sich um die Bausteine D27128 EPROM mit 16 KB Festwerten, in die versuchsweise das Rockwell RSC-FORTH V1.6 eingebrannt ist und um ein 8 KB RAM vom Typ HM 6264LP-15. Der Computer meldet sich nach dem Reset sofort in FORTH. wobei zu sagen ist, daß der Lieferant noch um eine Lizenz von Rockwell bemüht ist oder anderenfalls kurzfristig mit einer eigenen FORTH-Implementierung herauskommen wird.

Auf der Platine befinden sich außer einer Reset-Taste nebst Beschaltung dann noch verschiedene Codierstecker/Steckbrücken für die einsetzbaren gängigen Speichertypen und Verstärker für den seriellen Datenverkehr. Das System wird als Entwicklungssystem von einem Terminal her mit 1200 Baud bedient, und zwar voll duplex, d.h. die Karte echot die Eingaben auf das Terminal. Die seriellen Signale haben TTL-Level, sind also nicht 20 mA oder V24. Der Betrieb hier erfolgte am Regge-Terminal (lt mc 1/83), sollte aber auch z.B. mit einem AIM 65 möglich sein.

An den Rändern der 1/2 Europakarte finden sich eine 64polige VG-Messerleiste (Reihen a und b belegt, Bauform B64) mit allen Interfaceports und zusätzlichen Signalen, ein 4poliger Stecker für die Stromversorung und die Signale RX und TX sowie eine Buchsenleiste für Ausgangswiderstände (Pullup) und Zusatzkarten.

In kompakter Bauform handelt es sich damit um einen kompletten Computer, der außer mit FORTH als Sprache und Betriebssystem wahlweise auch mit einem Monitorprogramm geliefert wird, das das I/O mit der Außenwelt ermöglicht. Es ist etwa auf dem Level des KIM-1 Betriebssystems. In beiden Ausstattungen ist der Computer sowohl als Entwicklungs- wie auch als Zielsystem benutzbar. - Wenn sich mit dem KIM-1 in der Erinnerung des Autors auch eine sehr lehrreiche Zeit der Einarbeitung verbindet, so möchte er heute allein schon aus Produktivitätsgründen den Komfort einer höheren Sprache wie FORTH nicht missen, zumal hier ja auch ein Assembler für den erweiterten Befehlssatz der R6511 integriert ist. Wenn man sich nicht für die FORTH-Version entscheidet, so sollte man zweckmäßigerweise ein Entwicklungssystem wie etwa den AIM 65 haben, um dort assemblierte Programme herunterladen zu können und um Quelltexte auf Massenspeichern sichern zu können.

Beim hier benutzten Minibee MB6511 findet man folgende Speichereinteilung: I/O, RAM und Stack on chip in der Zero Page. Externes RAM ab hex 100-1FFF. Der Bereich von 100-2FF ist offensichtlich nicht belegt. Systemvariable des FORTH liegen ab 300, der Terminal Input Buffer (TIB) beginnt bei 380. TASK als niedrigstes RAM-Wort liegt bei 040B.

Der Kernel der Sprache FORTH (mit headerless code) ist ab Adresse F400 eingebrannt. Der Bereich F000-F3FF ist unbelegt und könnte für feste Anwenderroutinen nachgebrannt werden, z.B. Centronics-Ausgabe oder Bedienung des IEEE 488 Busses. Der hier vorliegende Festwertspeicher enthält in den Adressen C000-DFFF ferner das FORTH Dictionary sowie die übergeordneten Routinen einschließlich Compiler und Assembler, die in einem Zielsystem entbehrlich sind. Es handelt sich damit um ein Development ROM, das auch einen 'headerless code' zu generieren gestattet.

Das RSC-FORTH ist, wie bereits in Heft 37 dargestellt, ein FIG-FORTH in einer Version, die z.B. gegenüber derjenigen des AIM 65 beachtlich erweitert ist.

Mit dieser Platine, die 28 für den Benutzer freie Portpins hat und die damit hervorragend für Steuerungen und Interfacing eignet, wurden hier Interface-Versuche angestellt, für die nachfolgend Anregungen in der Spache FORTH gegeben werden.

2. Instrumentarium des FORTH

Die Abfrage und Steuerung von Interfacebausteinen mit ihren Ports und Registern geschieht typisch in Bit-Operationen oder in ganzen Bytes (8 Bit). Bei nebeneinanderliegenden Registern, wie z.B. bei Timern mag auch wortweise gearbeitet werden (16 Bit).

Transporte und Bearbeitungen erfolgen in FORTH dagegen typisch wortweise, also in 16 Bit. Eine Ausnahme machen Befehle wie C@ (c-fetch) und C! (c-store), die ein Byte aus adressierter Speicherstelle holen bzw. ein Byte dorthin abspeichern. Daß auf dem Datenstack intern gleichwohl in 2 Byte gearbeitet wird, braucht den Anwender nicht zu stören, denn das höherwertige Byte wird beim Laden zu 00 aufgefüllt, beim Schreiben jedoch nicht bewegt.

Diese beiden Befehle sind für die Bedienung von Interface-Adressen zunächst einmal wichtig. In 16 Bit Breite werden die korrespondierenden Worte @ (fetch) und ! (store) auch herangezogen werden. Für Bitmanipulationen bieten sich ferner die logischen Befehle AND OR XOR und NOT an. Daneben wird man Vergleichsbefehle heranziehen und auch die strukturierten Statements zur Programmverzweigung oder zum Warten auf eine Bedinung (mit WHILE oder UNTIL). Für Zeitberechnungen, Protokollierungen etc. kommen weiterhin die arithmetischen Befehle und die der Ein- und Ausgabe in Betracht.

FORTH hat den großen Vorteil, daß man in kleinen Modulen von wenigen Zeilen programmieren kann, die in Sekundenschnelle compilieren und die dann sofort erprobt werden können. Und es gibt den weiteren Vorteil, daß man bei den meisten Implementierungen mit dem Wort CODE auch den Assembler hereinrufen kann, um zunächst in FORTH formulierte zeitkritische Programmabschnitte zu optimieren. Dabei ist auch eine Übergabe von Argumenten zwischen Assemblerprogrammen und FORTH in beiden Richtungen möglich.

Sofern der Assembler Befehle hat, wie setze/lösche einzelnes Bit oder verzweige, wenn Bit gesetzt/gelöscht, dann lassen sich in der Tat viele Aufgaben elegant auch in Maschinensprache lösen.

3. Vorbereitende Maßnahmen

Sofern man kompakten Code für eine Zielmaschine erzeugen will (headerless code, frei von Verwaltungsinformation), wird man sich Gedanken über die Speichereinteilung machen. Der Code soll dann EPROM-fähig sein. Für Befehlsworte und Konstante ist das zulässig, nicht jedoch für Variablen, die ja im RAM liegen müssen, damit sie unterschiedliche Werte annehmen können. In Hinsicht auf die AUTOSTART-Fähigkeit wird man die Ablage des zu brennenden Codes an eine 1 K-Speichergrenze legen. Nach einem Reset sucht FORTH das Muster hex 5A A5 an den 1 K-Grenzen. Wird es angetroffen, so wird aus den beiden nachfolgenden Zellen die Codefeldadresse des auszuführenden Hauptprogrammes nach FORTH geladen. Erzeugter Code sollte in diesen Fällen in den Adreßbereichen liegen, in denen später ein EPROM decodiert wird, also etwa im Bereich ab hex 1004 oder ab 1404. Endziffer '4' wegen der Autostart-Bytes. Dann muß man dafür sorgen, daß die Variablen in den Bereichen gehalten werden, in denen später RAM ist, bei der vorliegenden Maschine z.B. etwa von 100-2FF oder von 400 bis 4FF (je nach Menge und auch Größe von Arrays). Man muß weiter dafür sorgen, daß die Verwaltungsinformation im Entwicklungssystem in einer nicht überschneidenden Art gehalten wird.

Diese Vorüberlegungen führen zu etwa folgender Vorgehensweise: Variable sollen ab hex 200 liegen, ein Array von 400 bis 480. Dann können die Namensköpfe und Parameter ab 500 liegen und der Code ab 1000. Wir schreiben:

FORGET TASK	Das Kunstwort im RAM löschen
HEX 500 H/C	Namensköpfe ab 500
200 DP !	Dictionary Pointer auf 200 für Variablen

```
11 VARIABLE ALPHA      Variable als erste anlegen
23 VARIABLE BETA       Variablen liegen im RAM!
400 DP !               Dictionary Pointer jetzt auf 400
55 VARIABLE GAMMA      Reserviert 2 Bytes
7E ALLOT               Reserviere 126 weitere Bytes
1004 DP !              Ab hier PROM-fähiger Code
....                   Weitere Compilierungen
: MAIN ... ;           Das Hauptprogramm
1000 AUTOSTART MAIN    Legt Autostart-Muster und den Vektor auf MAIN ab
```

Wenn wir so programmiert haben, dann ist zunächst der erzeugte Code ab 1000 zu retten. Das gilt möglichst natürlich auch für den Quelltext. Sobald wir jetzt die Reset-Taste drücken, startet der Computer wie später auch das Zielsystem mit der Hauptprogrammschleife bei MAIN. Er kann jetzt mit einem weiteren Reset nicht zurückgerufen werden, weil er immer wieder auto-startet. Man muß ihn also ausschalten, um Veränderungen vornehmen zu können. - Variable gehören also dorthin, wo später RAM ist, das kann natürlich auch hinter dem Bereich liegen, in dem der Code der Befehle abgespeichert wird. Und Variable sollten aus Gründen der Bequemlichkeit und Übersichtlichkeit zu den frühesten Definitionen gehören.

4. Interface-Worte in FORTH

Das RSC-FORTH (Rockwell) kennt das Befehlswort HWORD, mit der der Code einer Definition unmittelbar nach der Generierung aus dem Zielcode herausgenommen und in den Bereich des Dictionary unter Berichtigung der Pointer transportiert werden kann. Diese Einrichtung dient vor allem Testworten, die den Zielcode nicht belasten sollen. Wenn auf dem Zielsystem später das Dictionary nicht mehr vorhanden ist, darf im Zielsystem auf solche Definitionen nicht Bezug genommen werden, es sei denn, man würde mit Compiler-Tricks arbeiten, z.B. mit LITERAL. Im RSC-FORTH kann man gleichwohl bequem Konstanten von 1 Byte Breite schaffen, und zwar mit dem Wort C,CON. Dabei werden 5 Bytes im Zielcode belegt.

Soweit das Entwicklungs-ROM nicht schon symbolische Namen enthält, kann man diese zur Bequemlichkeit des Programmierers compilieren, also z.B.

```
HEX
0 C,CON PORTA  ( normal 0 CONSTANT PORTA)
11 C,CON IFR       (usw.)
```

Für spätere Bit-Operationen mag ein weiteres Arsenal symbolischer Hilfswörter nützlich sein:

```
HEX
1  C,CON BIT0
2  C,CON BIT1
...
80 C,CON BIT7
```

Datenrichtung am Port, zunächst für Bausteine wie 6522 VIA oder 6520 PIA:

```
: EINGANG  (DDR-Adresse ---)
00 SWAP C! ;

: AUSGANG  (DDR-Adresse ---)
FF SWAP C! ;
```

Gebrauch der Worte z.B. DDRA EINGANG.

Bausteine vom Typ R6511 haben eine vereinfachte Portbedienung ohne Datenrichtungsregister (siehe in diesem Heft). Ein Port wird in ganzer Breite Eingang durch

```
: EINGANG  (Portadresse ---)
FF SWAP C! ;
```

Durch Beschreiben der Pins mit HIGH sind externe Signale jetzt in der Lage, Pins auf LOW zu ziehen. Beim Lesen des Ports werden die dort tatsächlich anliegenden Pegel gelesen, nicht die in sie hineingeschriebenen Signale.

In den Port schreiben und ändern: Ob man besondere Definitionen für das Schreiben in Ports und Register schafft, ist eine Frage des Geschmackes. Es lohnt sich eigentlich nur, wenn die Definition auch weitere Bearbeitungen enthält.

```
: NACH   ( PARAMETER REGISTERADRESSE)
  C! ;   ( SCHREIBE PARAMETER INS REGISTER)
```

Analoges gilt für das Holen aus dem Port mit Transport zum Datenstack:

```
: VON    ( REGISTERADRESSE --- PARAMETER)
  C@ ;   ( HOLE INHALT AUS REGISTER)
```

Etwas anders sieht es aus, wenn man etwas in einem Register/Port gezielt ändern oder prüfen will. Z.B. sollen ein oder mehrere Bits eingeschaltet (zu '1' gesetzt) werden. Dann kann man schreiben:

```
: EIN1        ( PARAMETER ADRESSE ---)
  DUP C@      ( HOLE AUS DEM PORT)
  ROT         ( BEIDE ARGUMENTE ZUOBERST)
  OR          ( LOGISCHES ODER)
  SWAP C! ;   ( ZURUECKSCHREIBEN)
```

Man kann für das Hinzusetzen einzelner Bits auch ein CODE-Wort definieren, das eine Assembler-Routine enthält. In Heft 26 dieser Zeitschrift haben wir bereits die Benutzung des FORTH-Assemblers und die Parameterübergabe beschrieben. Im nachstehenden Beispiel werden zunächst 2 Byte-Paare mit JSR SETUP vom Datenstack des FORTH in die N-Area gebracht. In N 2+ steht das Parameter-Byte, in N und N+1 steht als Pointer die zu verwendende Adresse. Mit indirekter Adressierung über Y kann über diesen Pointer die logische Operation ORA benutzt werden. Die im RSC-FORTH enthaltenen Befehle zum Setzen und Löschen von Bits können hier nicht in einer allgemeinen Form mit Parameterübergabe benutzt werden, weil auf sie nicht die indirekte Adressierungsart anwendbar ist.

```
CODE EIN2         ( PARAMETER ADRESSE ---)
 2 # LDA,         ( 2 ITEMS VOM DATENSTACK)
 SETUP JSR,       ( PARAMETERUEBERGABE)
 N 2+ LDA,        ( PARAMETER LADEN)
 0 # LDY,         ( FUER INDIREKTE ADRESSIERUNG)
 N )Y ORA,        ( ADRESSPOINTER IN N UND N+1)
 N )Y STA,        ( ZURUECKSCHREIBEN IN PORT)
 NEXT JMP,        ( ZURUECK ZU FORTH)
END-CODE          ( VERLASSEN DES ASSEMBLERS)
```

Abschalten eines Bits/Pins:

```
: AUS         ( PARAMETER ADRESSE ---)
  SWAP        ( PARAMETER ZUOBERST)
  FF XOR      ( INVERTIEREN)
  OVER C@     ( ADRESSE DUPLIZIEREN, HOLEN)
  AND         ( BITMUSTER D. PARAMETERS AUSBLENDEN)
  SWAP C! ;   ( ZURUECKSCHREIBEN)
```

In einem gleichwertigen CODE-Wort für AUS wäre gemäß dem Beispiel EIN2 der ORA-Befehl zu ändern in

```
FF # EOR,
N ) AND,
```

Für das gezielte Umschalten eines Bits oder eines Bytes benötigt man keine Definition. Man schreibt einfach

Adresse Parameter TOGGLE z.B. PB 18 TOGGLE

Dabei werden die Bytes 4 und 3 in Port B invertiert. Entsprechend wäre nach dem Muster des Befehlswortes EIN2 dort auf den EOR-Befehl abzuändern. Man beachte, daß TOGGLE eine vom üblichen abweichende Notierungsart in der Reihenfolge hat.

Prüfen, ob Bit (Bits) gesetzt/gelöscht. Bei der Abfrage und Steuerung von Interfaces kommt es für den Programmfluß immer darauf an, ob gewisse Ereignisse eingetreten sind, die sich in Signalen an den Portpins oder in den Bits anderer Register äußern. In Assemblersprache geht es um den BIT-Befehl, den man bei der CPU 65C02 auch 'immediate' adressieren kann. Beim Warten auf Ereignisse kann man dort ferner mit den Befehlen Branch on Bit Set/Clear arbeiten. Auch diese ohne mögliche Parameterübergabe von FORTH, es sei denn, man wolle höchst individuelle Manipulationen über das RAM oder dort befindliche Variable vornehmen. Dem soll hier nicht das Wort geführt werden.

Das nachfolgende Wort ?EIN hinterläßt ein Flag auf dem Parameterstack, das mit IF abgefragt werden kann. Wenn das Bit gesetzt ist, wird der IF-Zweig eines nachfolgenden Statements ausgeführt, sonst der des ELSE, es sei denn man gibt vor dem IF noch das NOT, was die logischen Zweige umkehrt.

```
: ?EIN        ( PARAMETER ADRESSE --- FLAG)
  C@          ( HOLE BYTE AUS ADRESSE)
  AND ;       ( ERZEUGT FLAG)
```

Zum Wort ?EIN hier eine Alternative mit schneller ausführendem CODE:

```
CODE ?EIN2          ( PARAMETER ADRESSE --- FLAG)
 2 # LDA,           ( 2 ITEMS VOM DATENSTACK NEHMEN)
 SETUP JSR,         ( TRANSPORT)
 0 # LDY,           ( FUER INDIREKTE ADRESSIERUNG)
 N 2+ LDA,          ( PARAMETER)
 N )Y AND,          ( GEGEN INHALT AUS ADRESSE)
 PUSHØA JMP,        ( UEBERGABE AKKU AN FORTH)
END-CODE            ( VERLASSE ASSEMBLER)
```

Bei ?EIN und ?EIN2 ist zu beachten, daß wenn immer im Parameter-Byte mehrere Bits gesetzt sind (logisch ODER), das True-Flag immer gesetzt wird, sobald auch nur eins der Bits im Port gesetzt ist.

Mit den vorstehenden Beispielen dürfte ausreichend erklärt sein, auf wie einfache Weise man in FORTH an Ports eine Datenrichtung einstellt, Ports/Register beschreibt/verändert oder wie man Bits setzt, löscht, umschaltet und auf ihren Zustand abfragt. - Zu den Abfragen hier noch ein Beispiel. Es soll darauf gewartet werden, daß an Pin PA0 eine aufsteigende Impulsflanke eintritt. Sie setzt im IFR (Interrupt Flag Register, Adresse hex 11) in Bit 0 das Flag zu '1'.

```
: ?PULS         ( --- )
  BEGIN         ( SCHLEIFENBEGINN)
  1 IFR C@ AND  ( HERSTELLEN DER BEDINGUNG)
  Ø= NOT        ( ABFRAGE)
  UNTIL ;       ( WARTEN BIS AUF 1)
```

Peter W. Arps, 2000 Hamburg 73

INPUT-Map

für Commodore C 64

Das im MICRO MAG Nr. 36 veröffentlichte Programm MAPPING INPUT für den Commodore 64 hat, wie sich bei der Anwendung herausstellte, doch einige Schwachstellen. Es wurde daher erheblich erweitert, um dem Vorbild IBM 3270 näher zu kommen. Folgende Funktionen sind nun hinzugekommen:

1. Es werden alle Zeichen einschließlich führender Leerstellen, Komma und Doppelpunkt in die Variablen übernommen. Beim INPUT-Befehl können deshalb nur noch String-Variable verwendet werden.
2. Leerstrings werden in jedem Fall korrekt behandelt.
3. Es können bis zu vier Maps gleichzeitig vorrätig gehalten werden. Durch entsprechende OPEN-Befehle wird die angeforderte Map geladen oder aus dem Speicher (hinter dem BASIC-ROM) geholt. Ein schneller Wechsel der Map ist somit möglich. Es gibt jetzt folgende Arten des OPEN-Befehls:

   a) Eröffnen File 3, Laden der Map von der Floppy und Anzeige der Map:
   OPEN 3,3,n,"map-name" wobei n = 1 ... 4

   b) Eröffnen File 3, Laden der Map aus dem RAM und Anzeige der Map:
   OPEN 3,3,n n = 129 ... 132

   c) Eröffnen File 3 und Anzeige der laufenden Map:
   OPEN 3,3,0

   Schritt c) kann nicht vor Schritt b) und Schritt b) nicht vor a) gemacht werden. INPUT-MAP enthält entsprechende Prüfroutinen. Die Masken müssen nicht bei jedem Programm-Durchlauf erneut geladen werden.

   Die Anzahl der residenten Maps kann bei Verwendung des BASIC-RAMs noch erweitert werden.

4. Die Map wird nur beim OPEN-Befehl auf den Bildschirm gebracht. Dadurch ist es dem aufrufenden Programm möglich, in die Maske Informationen zu schreiben, die bis zum nächsten OPEN 3,... erhalten bleiben.

Beispiel:

```
10 OPEN 3,3,1,"NAME1" :REM LADEN UND ANZEIGEN DER MAP
20 PRINT "wert" :REM DEFAULT-WERT IN ERSTES FELD
30 INPUT#3,A$ :REM UEBERNAHME EINGABE ODER DEFAULT
40 CLOSE 3
50 OPEN 3,3,0 :REM MAP NEU AUF BILDSCHIRM SCHREIBEN
60 .....
```

5. Bei der Maskenerstellung können numerische Eingabefelder definiert werden. Es ist dann nur die Eingabe von Zahlen sowie '.' und '–' möglich. Hierzu sind die entsprechenden Felder bei der Maskenerstellung mit dem Klammeraffen (@) zu definieren. Die übrige Maskenerstellung hat sich nicht geändert.
6. Durch die Return-Taste wird zum nächsten Eingabefeld gesprungen, auch wenn dies in der gleichen Zeile ist.
7. Durch die F1-Taste wird zum nächsten Eingabefeld in der folgenden Zeile gesprungen.
8. Durch die F7-Taste wird die Eingabe beendet.
9. Der Cursor kann nach links über den Feldanfang auf das nächste Feld springen.
10. Die Clear-Taste bewirkt nur ein Löschen aller Eingabefelder. Die restliche Maske und die in geschützte Felder geschriebenen Informationen bleiben unverändert.
11. Die Insert- bzw. Delete-Funktion wirkt nur innerhalb des jeweiligen Eingabefeldes.

```
100 REM 160: REM WIRD ZU GOTO 160
110 POKE 1029,137: REM TOKEN AENDERN
120 OPEN 3,3,1,"MAP1": CLOSE3: REM MAP'S LADEN
130 OPEN 3,3,4,"MAP4": CLOSE3
140 OPEN 3,3,2,"MAP2": CLOSE3
150 REM
160 PRINT"clr"
170 OPEN 3,3,130: REM MAP2 EROEFFNEN
180 PRINT"text in die map schreiben"
190 INPUT#3,A$
200 INPUT#3,B$,C$,D$

    weitere verarbeitung

290 CLOSE 3
300 OPEN 3,3,132: REM MAP4 EROEFFNEN

    weitere verarbeitung

380 CLOSE 3
390 OPEN 3,3,0: REM LAUFENDE MAP ERNEUT EROEFFNEN

    weitere verarbeitung

450 CLOSE 3
460 OPEN 3,3,129: REM MAP1 EROEFFNEN

    weitere verarbeitung

770 CLOSE 3
780 END
```

```
0030              .LS
0040 ; ************************
0050 ; *                      *
0060 ; *      MAPPING-INPUT   *
0070 ; *       VERSION 2.1    *
0080 ; *       VOM 12.7.84    *
0090 ; *                      *
0100 ; ************************
0110              ;
0120              .BA $C000
0130              ;
0140 R6510        .DE 1
0150 DEVICE       .DE $13
0160 VON          .DE $AC
0170 PNT          .DE VON
0180 NACH         .DE $AE
0190 SEC.ADR      .DE $B9
0200 ANZ.KEY      .DE $C6
0210 VAR.LNG      .DE $C8
0220 BLNK.SW      .DE $CC
0230 BLNK.FLG     .DE $CF
0240 CRS.CHR      .DE $CE
0250 SCR.ADR      .DE $D1
0260 COLUMN       .DE $D3
0270 QUOT.SW      .DE $D4
0280 LNG.ZEIL     .DE $D5
0290 CRS.ZEIL     .DE $D6
0300 FELD         .DE $D7
0310 INS.MODE     .DE $D8
0320 ADR          .DE $FB
0330 CR.ALL       .DE $FD
0340 CR.VAR       .DE $C9
0350 FLD.PNT      .DE $FE
0360 VAR.ADR      .DE $49
0370 CHRGOT       .DE $73
0380 T.BUFF       .DE $200
0390 VAR.MOVE     .DE $A9DA
0400 STR.TEST     .DE $ADBF
0410 KOMMA        .DE $AEFD
0420 VAR.SUCH     .DE $B08B
0430 LNG.ERM      .DE $B4B9
0440 BASIC        .DE $A000
0450 KERNAL       .DE $E000
0460 SET.CRS      .DE $E56C
0470 INS.ZEIL     .DE $E965
0480 PRINT        .DE $E716
0490 BLK.COL      .DE $EA13
0500 SCRL.DOWN    .DE $292
0510 BGR.COLR     .DE $287
0520 GET.CHRS     .DE $E5B4
0530 SET.LFS      .DE $FFBA
0540 PGM.ERR      .DE $A43B
0550 SCREEN       .DE $400
0560 COL.RAM      .DE $D800
0570 STOP.TEST    .DE $FFE1
0580 ADD.1        .DE $FCDB
0590 END.TEST     .DE $FCD1
0600 OS.LOAD      .DE $FFD5
0610 OS.OPEN      .DE $F34A
0620 OS.INPUT     .DE $F157
0630 PRT.RTN      .DE $E674
```

```
                    0640 MSB.ROUT   .DE $E6B6
                    0650 KEY.DEC    .DE $EB48
                    0660 KEY.VEC    .DE $28F
                    0670 OPN.VEC    .DE $31A
                    0680 INP.VEC    .DE $324
                    0690            ;
                    0700            ;
                    0710            ;-------------------------------
                    0720            ;-       VEKTOREN SETZEN       -
                    0730            ;-------------------------------
C000- A9 3B         0740 INIT       LDA #L,OPEN
C002- A2 C0         0750            LDX #H,OPEN
C004- 8D 1A 03      0760            STA OPN.VEC
C007- 8E 1B 03      0770            STX OPN.VEC+1
C00A- A9 F6         0780            LDA #L,INP.START
C00C- A2 C0         0790            LDX #H,INP.START
C00E- 8D 24 03      0800            STA INP.VEC
C011- 8E 25 03      0810            STX INP.VEC+1
C014- A0 00         0820            LDY #L,KERNAL          ;OS INS RAM
C016- 84 AC         0830            STY *PNT
C018- A2 E0         0840            LDX #H,KERNAL
C01A- 86 AD         0850 IN1        STX *PNT+1
C01C- B1 AC         0860 IN2        LDA (PNT),Y
C01E- 91 AC         0870            STA (PNT),Y
C020- C8            0880            INY
C021- D0 F9         0890            BNE IN2
C023- E8            0900            INX
C024- D0 F4         0910            BNE IN1
C026- A9 60         0920            LDA #$60               ;UNTERDRUECKEN
C028- 8D 65 E9      0930            STA INS.ZEIL           ;LEERZEILEN
C02B- A9 A9         0940            LDA #$A9               ;LEERZEILEN
C02D- 8D BB E6      0950            STA MSB.ROUT+5
C030- A9 27         0960            LDA #$27
C032- 8D BC E6      0970            STA MSB.ROUT+6
C035- 8D C2 E6      0980            STA MSB.ROUT+12
C038- 60            0990            RTS
                    1000            ;
                    1010            ;
                    1020            ;-------------------------------
                    1030            ;-  OPEN-ROUTINE / MAP'S LADEN  -
                    1040            ;-------------------------------
C039- 28            1050 RTN1       PLP
C03A- 60            1060            RTS
                    1070            ;
C03B- 20 4A F3      1080 OPEN       JSR OS.OPEN
C03E- 08            1090            PHP
C03F- C9 03         1100            CMP #3                 ;OPEN X,3,..?
C041- D0 F6         1110            BNE RTN1               ;NICHT INPUT-MAPPING
C043- A5 B9         1120            LDA *SEC.ADR           ;ART TESTEN
C045- 29 1F         1130            AND #31
C047- 85 FB         1140            STA *ADR               ;SAVE IT
C049- F0 36         1150            BEQ MAP.TESTO          ;OPEN X,3,0
C04B- C9 05         1160            CMP #5                 ;1...4
C04D- B0 37         1170            BCS MAP.ERR
C04F- A6 B9         1180            LDX *SEC.ADR           ;OPEN X,3,128+X?
C051- 30 37         1190            BMI MAP.TEST           ;JA
C053- AA            1200            TAX
C054- CA            1210            DEX
C055- 8A            1220            TXA
C056- 0A            1230            ASL A                  ;MAL 8
C057- 0A            1240            ASL A
C058- 0A            1250            ASL A                  ;OFFSET MAP.ADR
C059- 9D 71 C3      1260            STA MAP.MERK1,X        ;MERKER SETZEN
C05C- A9 08         1270            LDA #8                 ;OPEN3,3,0,"NAME"
C05E- AA            1280            TAX
C05F- A0 00         1290            LDY #0
C061- 20 BA FF      1300            JSR SET.LFS FILE-PARAMETER
```

```
C064- A4 FB     1310             LDY *ADR
C066- BE 70 C3  1320             LDX MAP.MERK1-1,Y     ;OFFSET
C069- BC 49 C3  1330             LDY MAP.ADR+1,X       ;HIGH ADR
C06C- A2 00     1340             LDX #0                ;LOW ADR
C06E- A9 00     1350             LDA #0                ;LOAD
C070- 20 D5 FF  1360             JSR OS.LOAD           ;SCREEN + COL.RAM
C073- 90 1B     1370             BCC INIT.MAP          ;GELADEN
C075- 48        1380             PHA
C076- A4 FB     1390             LDY *ADR
C078- A9 FF     1400             LDA #255
C07A- 99 70 C3  1410             STA MAP.MERK1-1,Y     ;RESET MERKER
C07D- 68        1420             PLA
C07E- 4C 3B A4  1430 ERROR       JMP PGM.ERR           ;FILE NOT FOUND
                1440             ;
C081- AD 70 C3  1450 MAP.TESTO   LDA CURR.MAP          ;MAP DA?
C084- 10 2C     1460             BPL INIT.WARM         ;JA
C086- A9 04     1470 MAP.ERR     LDA #4                ;FEHLER-NR.
C088- D0 F4     1480             BNE ERROR             ;IMMER
                1490             ;
C08A- AA        1500 MAP.TEST    TAX
C08B- BD 70 C3  1510             LDA MAP.MERK1-1,X
C08E- 30 F6     1520             BMI MAP.ERR           ;MAP NICHT DA
                1530             ;
                1540             ;
                1550             ;---------------------------------
                1560             ;-      MAPS INITIALISIEREN      -
                1570             ;---------------------------------
C090- A6 FB     1580 INIT.MAP    LDX *ADR
C092- CA        1590             DEX
C093- 8E 70 C3  1600             STX CURR.MAP
C096- 20 F7 C2  1610             JSR MOVE.MAP          ;MAP HOLEN
C099- A0 24     1620             LDY #ADR2-MAP.ADR
C09B- 20 3A C3  1630             JSR SET.ADR
C09E- A2 00     1640             LDX #0
C0A0- A0 00     1650 LOOP1       LDY #0
C0A2- B1 AC     1660             LDA (VON),Y
C0A4- F0 1B     1670             BEQ ERM.FLD           ;'$'
C0A6- C9 64     1680             CMP #$64              ;C= '$'
C0A8- F0 17     1690             BEQ ERM.FLD
C0AA- 20 DB FC  1700             JSR ADD.1
C0AD- 20 D1 FC  1710             JSR END.TEST
C0B0- 90 EE     1720             BCC LOOP1             ;WEITER
                1730             ;
C0B2- A9 00     1740 INIT.WARM   LDA #0                ;MASKE ANZEIGEN
C0B4- 85 FD     1750             STA *CR.ALL           ;INPUT
C0B6- A0 20     1760             LDY #ADR1-MAP.ADR
C0B8- 18        1770             CLC
C0B9- 20 0C C3  1780             JSR MOVE
C0BC- 20 01 C3  1790             JSR MOVE.COLOR        ;UND FARBE
C0BF- 28        1800             PLP
C0C0- 60        1810             RTS
                1820             ;
C0C1- A5 AC     1830 ERM.FLD     LDA *VON              ;STORE ADR
C0C3- 9D 78 C3  1840             STA INP.FLD+1,X
C0C6- A5 AD     1850             LDA *VON+1
C0C8- 9D 79 C3  1860             STA INP.FLD+2,X
C0CB- D0 08     1870             BNE ERM1              ;IMMER
C0CD- A9 00     1880 LOOP2       LDA #0
C0CF- 2C        1890             .BY $2C
C0D0- A9 80     1900 LOOP2A      LDA #128
C0D2- 91 AC     1910             STA (VON),Y
C0D4- C8        1920             INY
C0D5- B1 AC     1930 ERM1        LDA (VON),Y
C0D7- F0 F7     1940             BEQ LOOP2A            ;NUMERIC
C0D9- C9 64     1950             CMP #$64
C0DB- F0 F0     1960             BEQ LOOP2             ;ALPHA
C0DD- 98        1970             TYA
C0DE- 9D 77 C3  1980             STA INP.FLD,X         ;LAENGE
```

```
C0E1- 18          1990              CLC
C0E2- 8A          2000              TXA
C0E3- 69 03       2010              ADC #3                ;NEXT FIELD
C0E5- AA          2020              TAX
C0E6- A9 00       2030              LDA #0
C0E8- 9D 77 C3    2040              STA INP.FLD,X         ;END-MERKER
C0EB- 98          2050              TYA
C0EC- 65 AC       2060              ADC *VON
C0EE- 85 AC       2070              STA *VON
C0F0- 90 AE       2080              BCC LOOP1
C0F2- E6 AD       2090              INC *VON+1
C0F4- D0 AA       2100              BNE LOOP1             ;IMMER
                  2110              ;
                  2120              ;
                  2130              ;-----------------------------------
                  2140              ;-       MAPPING.INPUT             -
                  2150              ;-----------------------------------
C0F6- A5 13       2160 INP.START    LDA *DEVICE
C0F8- C9 03       2170              CMP #3
C0FA- F0 03       2180              BEQ DEVICE.3
C0FC- 4C 57 F1    2190              JMP OS.INPUT
                  2200              ;
C0FF- 98          2210 DEVICE.3     TYA
C100- 48          2220              PHA
C101- 8A          2230              TXA
C102- 48          2240              PHA
C103- A5 FD       2250              LDA *CR.ALL           ;ALLES UEBERGEBEN?
C105- F0 03       2260              BEQ INPUT0            ;JA
C107- 4C C8 C1    2270              JMP NEXT.VAR
                  2280              ;
C10A- AD 8F 02    2290 INPUT0       LDA KEY.VEC           ;SAVE VECTOR
C10D- AE 90 02    2300              LDX KEY.VEC+1         ;WEGEN EVT.
C110- 8D 75 C3    2310              STA SAV.KEYVEC        ;FUNKTIONSTASTEN-
C113- 8E 76 C3    2320              STX SAV.KEYVEC+1      ;BELEGUNG
C116- A9 48       2330              LDA #L,KEY.DEC        ;AUSSCHALTEN
C118- A2 EB       2340              LDX #H,KEY.DEC
C11A- 8D 8F 02    2350              STA KEY.VEC
C11D- 8E 90 02    2360              STX KEY.VEC+1
C120- A5 01       2370              LDA *R6510
C122- 29 FD       2380              AND #$FD              ;OS UMSCHALTEN
C124- 85 01       2390              STA *R6510
C126- D0 05       2400              BNE INPUT9
                  2410              ;
                  2420              ;-----------------------------------
                  2430              ;-       INPUT-ROUTINE             -
                  2440              ;-----------------------------------
C128- A9 13       2450 INP.HOME     LDA #19
C12A- 20 16 E7    2460 INPUT        JSR PRINT
C12D- A9 00       2470 INPUT9       LDA #0
C12F- 85 D4       2480              STA *QUOT.SW
C131- 85 D8       2490              STA *INS.MODE
C133- A5 D6       2500              LDA *CRS.ZEIL
C135- C9 18       2510              CMP #24               ;LETZTE ZEILE?
C137- B0 EF       2520              BCS INP.HOME          ;JA
C139- 20 5F C2    2530              JSR STATUS
C13C- F0 04       2540              BEQ WAIT.LOOP         ;EINGABE-FELD
C13E- A9 1D       2550              LDA #29               ;CRS RIGHT
C140- D0 E8       2560              BNE INPUT             ;IMMER
                  2570              ;
C142- A5 C6       2580 WAIT.LOOP    LDA *ANZ.KEY
C144- 85 CC       2590              STA *BLNK.SW
C146- 8D 92 02    2600              STA SCRL.DOWN
C149- F0 F7       2610              BEQ WAIT.LOOP
C14B- 78          2620              SEI
C14C- A5 CF       2630              LDA *BLNK.FLG
C14E- F0 0C       2640              BEQ INPUT1
C150- A5 CE       2650              LDA *CRS.CHR
C152- AE 87 02    2660              LDX BGR.COLR
```

```
C155- A0 00     2670             LDY #0
C157- 84 CF     2680             STY *BLNK.FLG
C159- 20 13 EA  2690             JSR BLK.COL
                2700             ;
C15C- 20 B4 E5  2710 INPUT1      JSR GET.CHRS
C15F- 85 D7     2720             STA *FELD
C161- 29 7F     2730             AND #$7F
C163- C9 21     2740             CMP #'!                ;$00 - $20?
C165- 90 13     2750             BCC INPUT1A            ;JA
C167- 20 5F C2  2760             JSR STATUS
C16A- 90 0E     2770             BCC INPUT1A            ;ALLES
C16C- A5 D7     2780             LDA *FELD
C16E- C9 3A     2790             CMP #':                ;BUCHSTABEN
C170- B0 BB     2800             BCS INPUT9             ;SKIP
C172- C9 2F     2810             CMP #'/
C174- F0 B7     2820             BEQ INPUT9             ;SKIP
C176- C9 2D     2830             CMP #'-                ;'"' - ','
C178- 90 B3     2840             BCC INPUT9             ;SKIP
                2850             ;
C17A- A5 D7     2860 INPUT1A     LDA *FELD
C17C- C9 14     2870             CMP #$14               ;DELETE
C17E- D0 03     2880             BNE INPUT2
C180- 4C BA C2  2890             JMP DELETE
                2900             ;
C183- C9 0D     2910 INPUT2      CMP #13                ;CR
C185- D0 03     2920             BNE INPUT3
C187- 4C 82 C2  2930             JMP TAB
                2940             ;
C18A- C9 93     2950 INPUT3      CMP #$93               ;CLR
C18C- D0 03     2960             BNE INPUT4
C18E- 4C 9F C2  2970             JMP CLEAR
                2980             ;
C191- C9 94     2990 INPUT4      CMP #$94               ;INSERT
C193- D0 03     3000             BNE INPUT5
C195- 4C E2 C2  3010             JMP INSERT
                3020             ;
C198- C9 9D     3030 INPUT5      CMP #$9D               ;CRS LEFT
C19A- D0 03     3040             BNE INPUT6
C19C- 4C 70 C2  3050             JMP CRS.LEFT
                3060             ;
C19F- C9 85     3070 INPUT6      CMP #$85               ;F1
C1A1- D0 04     3080             BNE INPUT7
C1A3- A9 0D     3090             LDA #13                ;NEXT LINE
C1A5- D0 83     3100             BNE INPUT              ;IMMER
                3110             ;
C1A7- C9 88     3120 INPUT7      CMP #$88               ;F7
C1A9- F0 03     3130             BEQ INP.END
C1AB- 4C 2A C1  3140             JMP INPUT
                3150             ;
C1AE- AD 75 C3  3160 INP.END     LDA SAV.KEYVEC         ;RESTORE VECTOR
C1B1- AE 76 C3  3170             LDX SAV.KEYVEC+1       ;BELEGUNG
C1B4- 8D 8F 02  3180             STA KEY.VEC
C1B7- 8E 90 02  3190             STX KEY.VEC+1
C1BA- A5 01     3200             LDA *R6510
C1BC- 09 03     3210             ORA #$03               ;ROM EIN
C1BE- 85 01     3220             STA *R6510
C1C0- A9 00     3230             LDA #0
C1C2- 85 C9     3240             STA *CR.VAR
C1C4- 85 FE     3250             STA *FLD.PNT
C1C6- E6 FD     3260             INC *CR.ALL
                3270             ;
                3280             ;-------------------------------
                3290             ;-     VARIABLEN-UEBERGABE     -
                3300             ;-------------------------------
C1C8- 68        3310 NEXT.VAR    PLA                    ;STACK BERICHTIGEN
C1C9- 68        3320             PLA
C1CA- 68        3330             PLA
C1CB- 68        3340             PLA
```

```
C1CC- 68          3350              PLA
C1CD- 68          3360              PLA
C1CE- 68          3370              PLA
C1CF- 68          3380              PLA
C1D0- 20 BB B0    3390 VAR.NEXT     JSR VAR.SUCH EN
C1D3- 85 49       3400              STA *VAR.ADR
C1D5- 84 4A       3410              STY *VAR.ADR+1
C1D7- 20 BF AD    3420              JSR STR.TEST          ;STRINGTEST
C1DA- A2 FF       3430              LDX #255
C1DC- E8          3440 CHR.HOL      INX
C1DD- 20 01 C2    3450              JSR HOL.CHR           ;ZEICHEN
C1E0- 9D 00 02    3460              STA T.BUFF,X          ;IN DEN BUFFER
C1E3- C9 0D       3470              CMP #13               ;ENDE?
C1E5- D0 F5       3480              BNE CHR.HOL           ;NEIN
C1E7- A9 00       3490              LDA #0                ;ABSCHLUSS-KENNUNG
C1E9- 9D 00 02    3500              STA T.BUFF,X
C1EC- AA          3510              TAX
C1ED- A0 02       3520              LDY #2
C1EF- 20 89 B4    3530              JSR LNG.ERM           ;LAENGE
C1F2- 20 DA A9    3540              JSR VAR.MOVE          ;VAR AUFBAUEN
C1F5- 20 79 00    3550              JSR CHRGOT+6          ;WEITER?
C1F8- F0 06       3560              BEQ VAR.END           ;NEIN
C1FA- 20 FD AE    3570              JSR KOMMA             ;KOMMA-TEST
C1FD- 4C D0 C1    3580              JMP VAR.NEXT          ;WEITER
                  3590              ;
C200- 60          3600 VAR.END      RTS
                  3610              ;
                  3620              ;-----------------------------------
                  3630              ;-      ZEICHEN-UEBERGABE          -
                  3640              ;-----------------------------------
C201- 98          3650 HOL.CHR      TYA
C202- 48          3660              PHA
C203- 8A          3670              TXA
C204- 48          3680              PHA
C205- A4 C9       3690              LDY *CR.VAR           ;NEUES FELD?
C207- D0 1F       3700              BNE HOL1              ;NEIN
C209- A6 FE       3710              LDX *FLD.PNT          ;NR FELD
C20B- BD 77 C3    3720              LDA INP.FLD,X         ;LAENGE
C20E- A8          3730              TAY
C20F- BD 78 C3    3740              LDA INP.FLD+1,X       ;ADL
C212- 85 FB       3750              STA *ADR
C214- BD 79 C3    3760              LDA INP.FLD+2,X       ;ADH
C217- 18          3770              CLC
C218- 69 3F       3780              ADC #H,SCREEN-MAP.FLD
C21A- 85 FC       3790              STA *ADR+1
C21C- 88          3800 LOOP4        DEY                   ;FELD-ENDE ERMITTELN
C21D- 30 26       3810              BMI HOL.NEU           ;LEER-FELD
C21F- B1 FB       3820              LDA (ADR),Y
C221- C9 20       3830              CMP #'
C223- F0 F7       3840              BEQ LOOP4
C225- C8          3850              INY
C226- 84 C8       3860              STY *VAR.LNG          ;SAVE ENDE
C228- A4 C9       3870 HOL1         LDY *CR.VAR           ;CURR. POS.
C22A- C4 C8       3880              CPY *VAR.LNG          ;FELD-ENDE?
C22C- B0 17       3890              BCS HOL.NEU           ;JA
C22E- E6 C9       3900              INC *CR.VAR
C230- B1 FB       3910              LDA (ADR),Y           ;HOL ZEICHEN
C232- 85 D7       3920              STA *FELD             ;IN ASCII
C234- 29 3F       3930              AND #$3F
C236- 06 D7       3940              ASL *FELD
C238- 24 D7       3950              BIT *FELD
C23A- 10 02       3960              BPL HOL2
C23C- 09 80       3970              ORA #128
C23E- 70 02       3980 HOL2         BVS HOL3
C240- 09 40       3990              ORA #64
C242- 4C 74 E6    4000 HOL3         JMP PRT.RTN           ;ZURUECK
                  4010              ;
C245- A9 00       4020 HOL.NEU      LDA #0
```

```
C247- 85 C9      4030          STA *CR.VAR          ;LOESCHEN
C249- 18         4040          CLC
C24A- A5 FE      4050          LDA *FLD.PNT         ;INDEX ERHOEHEN
C24C- 69 03      4060          ADC #3
C24E- 85 FE      4070          STA *FLD.PNT
C250- AA         4080          TAX
C251- BD 77 C3   4090          LDA INP.FLD,X        ;ENDE?
C254- D0 06      4100          BNE HOL4             ;NEIN
C256- A9 00      4110          LDA #0
C258- 85 FE      4120          STA *FLD.PNT
C25A- 85 FD      4130          STA *CR.ALL
C25C- 4C 72 E6   4140 HOL4     JMP PRT.RTN-2        ;RETURN MIT CR
                 4150          ;
                 4160          ;
                 4170          ;---------------------------------
                 4180          ;-      INPUT-UNTER-ROUTINEN      -
                 4190          ;---------------------------------
C25F- 18         4200 STATUS   CLC
C260- A5 D1      4210          LDA *SCR.ADR         ;KORRESPONDIERENDES
C262- 85 AC      4220          STA *PNT             ;FELD DER MAP LADEN
C264- A5 D2      4230          LDA *SCR.ADR+1
C266- 69 C1      4240          ADC #H,MAP.FLD-SCREEN
C268- 85 AD      4250          STA *PNT+1
C26A- A4 D3      4260          LDY *COLUMN
C26C- B1 AC      4270          LDA (PNT),Y
C26E- 0A         4280          ASL A
C26F- 60         4290          RTS
                 4300          ;
C270- 20 16 E7   4310 CRS.LEFT JSR PRINT
C273- 20 5F C2   4320          JSR STATUS
C276- F0 24      4330          BEQ RTN3
C278- A5 D6      4340          LDA *CRS.ZEIL
C27A- 05 D3      4350          ORA *COLUMN          ;OBEN LINKS?
C27C- F0 39      4360          BEQ RTN4             ;JA
C27E- A9 9D      4370          LDA #$9D
C280- D0 EE      4380          BNE CRS.LEFT         ;IMMER
                 4390          ;
C282- A9 1D      4400 TAB      LDA #29
C284- 20 16 E7   4410          JSR PRINT
C287- 20 5F C2   4420          JSR STATUS
C28A- F0 F6      4430          BEQ TAB
C28C- A5 D6      4440 TAB1     LDA *CRS.ZEIL
C28E- C9 18      4450          CMP #24              ;LETZTE ZEILE?
C290- B0 25      4460          BCS RTN4             ;JA
C292- A9 1D      4470          LDA #29
C294- 20 16 E7   4480          JSR PRINT
C297- 20 5F C2   4490          JSR STATUS
C29A- D0 F0      4500          BNE TAB1
C29C- 4C 2D C1   4510 RTN3     JMP INPUT9
                 4520          ;
C29F- A9 13      4530 CLEAR    LDA #19              ;HOME
C2A1- 20 16 E7   4540          JSR PRINT
C2A4- 20 5F C2   4550 CLR0     JSR STATUS
C2A7- F0 03      4560          BEQ CLR1
C2A9- A9 1D      4570          LDA #29
C2AB- 2C         4580          .BY $2C
C2AC- A9 20      4590 CLR1     LDA #'
C2AE- 20 16 E7   4600          JSR PRINT
C2B1- A5 D6      4610          LDA *CRS.ZEIL
C2B3- C9 18      4620          CMP #24              ;LETZTE ZEILE?
C2B5- 90 ED      4630          BCC CLR0             ;NEIN
C2B7- 4C 28 C1   4640 RTN4     JMP INP.HOME
                 4650          ;
C2BA- A9 9D      4660 DELETE   LDA #$9D             ;CRS LEFT
C2BC- 20 16 E7   4670          JSR PRINT
C2BF- 20 5F C2   4680          JSR STATUS
C2C2- D0 D8      4690          BNE RTN3             ;IST KANTE
C2C4- A9 1D      4700          LDA #$1D             ;CRS RIGHT
```

```
C2C6- 20 16 E7  4710            JSR PRINT
C2C9- 20 5F C2  4720            JSR STATUS
C2CC- 88        4730            DEY
C2CD- C8        4740 DEL1       INY
C2CE- B1 AC     4750            LDA (PNT),Y
C2D0- 29 7F     4760            AND #$7F
C2D2- F0 F9     4770            BEQ DEL1            ;NICHT FELDENDE
C2D4- A9 14     4780            LDA #20             ;DELETE
C2D6- 88        4790 DEL2       DEY                 ;KORREKTUR
C2D7- 84 D5     4800            STY *LNG.ZEIL       ;ZEILENLAENGE
C2D9- 20 16 E7  4810            JSR PRINT
C2DC- 20 6C E5  4820            JSR SET.CRS         ;KORREKTUR
C2DF- 4C 9C C2  4830            JMP RTN3            ;IMMER
                4840            ;
C2E2- 20 5F C2  4850 INSERT     JSR STATUS
C2E5- C8        4860            INY
C2E6- B1 AC     4870            LDA (PNT),Y
C2E8- 29 7F     4880            AND #$7F
C2EA- D0 B0     4890            BNE RTN3            ;FELDENDE
C2EC- C8        4900 INS1       INY
C2ED- B1 AC     4910            LDA (PNT),Y
C2EF- 29 7F     4920            AND #$7F
C2F1- F0 F9     4930            BEQ INS1            ;NICHT FELDENDE
C2F3- A9 94     4940            LDA #148            ;INSERT
C2F5- D0 DF     4950            BNE DEL2            ;IMMER
                4960            ;
                4970            ;
                4980            ;-----------------------------------
                4990            ;-      MAP-MOVE-ROUTINEN          -
                5000            ;-----------------------------------
C2F7- AE 70 C3  5010 MOVE.MAP   LDX CURR.MAP
C2FA- BC 71 C3  5020            LDY MAP.MERK1,X
C2FD- 38        5030            SEC
C2FE- 20 0C C3  5040            JSR MOVE            ;SCREEN
C301- AE 70 C3  5050 MOVE.COLOR LDX CURR.MAP
C304- BD 71 C3  5060            LDA MAP.MERK1,X
C307- 18        5070            CLC
C308- 69 04     5080            ADC #4              ;OFFSET+4
C30A- A8        5090            TAY
C30B- 38        5100            SEC                 ;COL-RAM
                5110            ;
C30C- 08        5120 MOVE       PHP                 ;BLOCK-MOVE
C30D- 20 3A C3  5130            JSR SET.ADR
C310- 28        5140            PLP
C311- 78        5150            SEI
C312- A5 01     5160            LDA *R6510
C314- 29 FE     5170            AND #$FE            ;ROM UMSCHALTEN
C316- 85 01     5180            STA *R6510
C318- A2 04     5190            LDX #4              ;CY=1 ALLES
C31A- A0 00     5200            LDY #0              ;CY=0 <>0
C31C- B1 AC     5210 MOVE0      LDA (VON),Y
C31E- B0 06     5220            BCS MOVE1           ;STORE BYTE
C320- 29 7F     5230            AND #$7F
C322- F0 04     5240            BEQ MOVE2           ;SKIP BYTE
C324- B1 AC     5250            LDA (VON),Y
C326- 91 AE     5260 MOVE1      STA (NACH),Y
C328- C8        5270 MOVE2      INY
C329- D0 F1     5280            BNE MOVE0
C32B- E6 AD     5290            INC *VON+1
C32D- E6 AF     5300            INC *NACH+1
C32F- CA        5310            DEX
C330- D0 EA     5320            BNE MOVE0
C332- A5 01     5330            LDA *R6510
C334- 09 03     5340            ORA #$03            ;ROM EIN
C336- 85 01     5350            STA *R6510
C338- 58        5360            CLI
C339- 60        5370            RTS
```

```
                    5380                ;
C33A- A2 00         5390 SET.ADR        LDX #0
C33C- B9 48 C3      5400 LOOP3          LDA MAP.ADR,Y
C33F- 95 AC         5410                STA *VON,X
C341- C8            5420                INY
C342- E8            5430                INX
C343- E0 04         5440                CPX #4
C345- D0 F5         5450                BNE LOOP3
C347- 60            5460                RTS
                    5470                ;
                    5480                ;
C348- 00 A0         5490 MAP.ADR        .SI BASIC
C34A- 00 C5         5500                .SI MAP.FLD
C34C- E8 A3         5510                .SI BASIC+1000
C34E- 00 D8         5520                .SI COL.RAM
                    5530                ;
C350- 00 A8         5540                .SI BASIC+$800
C352- 00 C5         5550                .SI MAP.FLD
C354- E8 AB         5560                .SI BASIC+$800+1000
C356- 00 D8         5570                .SI COL.RAM
                    5580                ;
C358- 00 B0         5590                .SI BASIC+$1000
C35A- 00 C5         5600                .SI MAP.FLD
C35C- E8 B3         5610                .SI BASIC+$1000+1000
C35E- 00 D8         5620                .SI COL.RAM
                    5630                ;
C360- 00 B8         5640                .SI BASIC+$1800
C362- 00 C5         5650                .SI MAP.FLD
C364- E8 BB         5660                .SI BASIC+$1800+1000
C366- 00 D8         5670                .SI COL.RAM
                    5680                ;
C368- 00 C5         5690 ADR1           .SI MAP.FLD
C36A- 00 04         5700                .SI SCREEN
C36C- 00 C5         5710 ADR2           .SI MAP.FLD
C36E- E8 C8         5720                .SI MAP.FLD+1000
                    5730                ;
C370- FF            5740 CURR.MAP       .BY 255
C371- FF            5750 MAP.MERK1      .BY 255
C372- FF            5760 MAP.MERK2      .BY 255
C373- FF            5770 MAP.MERK3      .BY 255
C374- FF            5780 MAP.MERK4      .BY 255
C375- 00 00         5790 SAV.KEYVEC     .SI 0
                    5800                ;
                    5810 MAP.FLD        .DE INIT+$500          ;CURRENT MAP
                    5820                ;
                    5830                ; JE FELD 3 BYTES
C377- 00            5840 INP.FLD        .BY 0                  ;LAENGE
C378- 00 00         5850                .SI 0                  ;ADRESSE
                    5860 ;              .EN

 END OF PASS
//0000,C37A,C37A
```

=□

Kleinanzeige

Commodore 710 zu verkaufen. Buchholz, Tel. 030 - 68 45 769

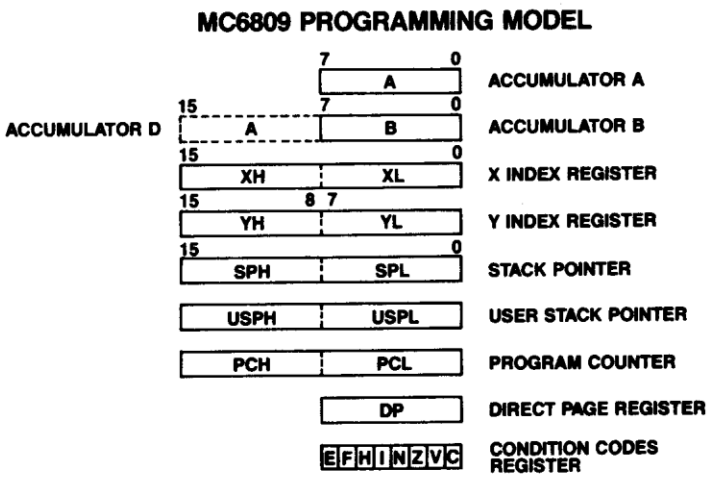
Roland Löhr

Cross-Assembler für MC6809

1. Übersicht

Die CPU 6809 von Motorola wurde in den Heften 15, 18, 19, 20 und 22 mit ihren Signalen, Registern, 59 Befehlen und 9 Adressierungsarten vorgestellt. Heft 35 enthielt ferner einen in FORTH geschriebenen Cross-Assembler für die CPU 6805/68705. Auch der nachfolgende Cross-Assembler ist in FORTH geschrieben, um die Assemblersprache einer fremden Maschine abzubilden. Eigentlich müßte man sagen, es sei ein Cross-Compiler, denn wir finden an vielen Stellen Überprüfungen, wie sie auch im FORTH-Compiler benutzt werden.

Zunächst zusammenfassend und wiederholend zur CPU. Wir haben dort folgendes Registermodell:



Die Akkumulatoren A und B können nach der Manier einer 8 Bit-CPU als 8 Bit breite Register benutzt werden. Es gibt dafür jeweils einen vollen Befehlssatz. Man hat aber auch die Möglichkeit, sie als ein zusammenhängendes 16 Bit breites Register D zu benutzen, vor allem für ladende, speichernde Befehle und für die des Rechnens und der Vergleiche. Dem Anwender stehen ferner in 16 Bit Breite vier weitere Register zur Verfügung, die mit X und Y als Indexregister bezeichnet werden und mit S und U als Hardware- und User-Stackpointer, wobei S beim Aufruf von Unterprogrammen und beim Eintritt von Interrupts die notwendigen Parameter für die spätere Programmfortsetzung aufnimmt. Diese Register sind durch verschiedene Transferbefehle aber auch summarisch als Indexregister zu bezeichnen und zu gebrauchen, zumal die beiden Stackpointer nach einem Reset vom Systemprogramm zu initialisieren sind..

Auch der Programmzähler PC, relativ zu dem adreßunabhängiger Code geschrieben werden kann, hat Wortbreite. Für verkürzte Adressierungen steht ein 8 Bit breites Direct Page Register DP zur verfügung, dessen Inhalt bei entsprechenden Befehlen als 'Seitenzahl' auf den hohen Adreßbus ausgesandt wird. Es bestimmt die bei 6502 bekannte 'Zero Page' in einem anderen Adreßraum. Auch das Statusregister hat 8 Bit.

Der Befehlssatz ist in den Hexbytes seiner Opcodes sehr regelmäßig angelegt. Dafür die nebenstehende Tabelle. Und wir haben folgende Hauptadressierungsarten und ihre Befehlslänge:

a) 1 Byte breite Befehle

Inherent-Befehle wie RTS, DAA (Decimal Adjust) oder NEGA, NEGB,
Befehle, die unmittelbar auf die kurzen Akkumulatoren wirken,

b) 2 Byte breite Befehle

Direktoperand
Direct Page (zusammen mit Seitenregister DP)
Short Branches
Indiziert, nur mit Postbyte und kleinstem Offset
Verschiedene Befehle, z.B. mit Registerlisten im Bitmuster

c) 3 Byte breite Befehle

Long Branches LBRA und LBSR (always and to Subroutine)
Extended (absolute Adressierung)
Indiziert mit Postbyte und kurzen Offset +127 oder -128

d) 4 Byte lange Befehle

Indiziert mit Postbyte und großem Offset
Indiziert, mit Prebyte und kurzem Offset

e) 5 Byte lange Befehle

Indiziert, mit Prebyte und großem Offset

Für den Autor eines Cross-Assemblers ist es bei dieser Vielfalt gar nicht so leicht, die vielen Besonderheiten auch der einzelnen Befehle abzudecken. Sie bedingen in der hier vorliegenden Form einen compilierten FORTH-Code von mehr als 6 KB. Dafür hat man einen strukturierten 1-Pass-Assembler, der viele Freiheiten und Beeinflussungsmöglichkeiten zuläßt, der Überprüfungen vornimmt und der es gestattet, mit externen Symbolen und internen Labeln zu arbeiten. Wichtig ist auch die Fähigkeit, daß man den den für einen beliebigen Adreßraum (Assembleranweisung ORG) erzeugten Code mit ASSIGN an beliebiger Stelle im nicht überschneidenden RAM ablegen kann, um ihn von dort in ein EPROM zu brennen.

Wenn vieles an diesem Assembler auch spezifisch für 6809 ist, so sollten der Aufbau und die verwirklichten Dienstleistungen auch für einen Nachbau für noch andere CPU's Pate stehen können. Cross-Assembler haben den Vorteil, daß sie oft ein besonderes Entwicklungssystem entbehrlich machen.

2. Der Aufbau des Assemblers

Der Assembler bildet unter FORTH ein eigenes VOCABULARY mit dem Namen ASS. Daher auch am Beginn das ASS DEFINITIONS.

Die Befehle der Assemblersprache werden, wie unter FORTH üblich, als Mnemonics mit nachgestelltem Komma geschrieben. Die Notierungsart ist umgekehrt polnisch, es gibt aber viele Freiheiten in der Abfolge von Operanden und Adressierungsarten, so daß der Assembler recht weitherzig ist.

Soweit Befehle zu Gruppen zusammengefaßt werden konnten, wurden für diese mit <BUILDS und DOES> Datentypen geschaffen, so z.B. für die Gruppe INH (inherent), BR (Branches) und LB (Long Branches mit Prebyte). Die größten Gruppen werden unter MI (Verschiebebefehle usw.) sowie MT (Laden, Speichern, Rechnen usw.) zusammengefaßt. Das Wort INDEX ist in beiden für indizierte Adressierungen wichtig. Speziell bei diesem Wort haben wir eine ziemliche Verschachtelungstiefe, die durch in Kommentarklammern gesetzte Ziffern angezeigt wird. Solche komplexen Gebilde sind in FORTH nicht üblich. Empfehlenwert wären hier kleinere Module gewesen.

Die vom Anwender gewünschte Adressierungsart wird in der Systemvariablen MODE geführt. Bei indizierter Adressierung hat auch die Variable POSTBYTE Bedeutung, die vom Defining Wort REG mit PBOR geformt und von INDEX schließlich verwertet wird.

Für speichernde Befehle und solche mit einem Prebyte wurden Definitionen geschaffen, die mit

MAC... beginnende Worte verwenden. Hier soll die Adressierungsart 'immediate' für das Speichern verboten bzw. das Prebyte erzeugt werden.

Am Schluß des Listings finden wir Assembleranweisungen, die noch erklärt werden. Davor liegen die Kontrollstrukturen IF, ELSE THEN usw. Hier ist viel Mühe aufgewandt worden, den Code zu optimieren und mögliche Fehler zu verzeihen: Im Gegensatz zu anderen Assemblern unter FORTH wird grundsätzlich nicht mit JMP-Befehlen gearbeitet, sondern mit Branches. Wenn in den Kontrollstrukturen eine kurze Verzweigung (Offset in +- 7 Bit) nicht ausreicht, dann wird automatisch ein Long Branch erzeugt, sogar mit Ersetzung des Opcodes durch Prebyte und Opcode.

Im mittleren Teil des Listings finden wir Definitionen für die Adressierungsarten für Registerlisten und für besondere Befehle. Daran schließen sich dann die mnemonischen Befehle mit Rückgriff auf die zu ihnen gehörenden 'defining words' an.

3. Bedienung des Assemblers und Formulierungen

Es wurde angestrebt, die Syntax des Motorola-Assemblers benutzbar zu machen, wenn auch in der umgekehrt polnischen Notierung. Die meisten Anweisungen wurden generiert mit ASS DEFINITIONS. Wer neue Anweisungen schaffen möchte oder bestehende abändern will, benutze zunächst diese beiden Worte. Z.B. kann die ORG-Anweisung wie folgt auf eine Sternanweisung geändert werden:

```
ASS DEFINITIONS  : *= ORG ;
```

Neue Definitionen werden also wie üblich zwischen Doppelpunkt und Semikolon geschaffen. Es ist im Quelltext möglich, insbesondere die Rechenhaftigkeit von FORTH mitzubenutzen, um mit Operanden zu rechnen. Z.B. soll der Akku A mit dem dritten Byte aus TABELLE, einem Symbol geladen werden:

```
TABELLE 3 + LDA,
```

Gleiches gilt natürlich auch für die anderen Grundrechenarten und Dienstleistungen des unter dem Assembler nach wie vor erreichbaren FORTH.

Die Zuordung des Zielbereiches auf dem Entwicklungssystem, in dem der erzeugte Code abgelegt wird, geschieht mit xxxx ASSIGN. Z.B. 4000 ASSIGN. Der Code landet dann an einer Adresse, die sich additiv aus dem mit yyyy ORG gesetzten Zuordnungszähler für die Zielmaschine ergibt. Wenn man also z.B. Code für den Speicherbereich ab F000 erzeugen will und wenn dieser Code auf der FORTH-Maschine ab Adresse 3000 stehen soll, dann gibt man:

```
F000 ORG 4000 ASSIGN
```

Die Summe ergibt dann (ohne Übertrag) hex 3000. xxxx wird dabei in der Variablen RAMAREA gespeichert, yyyy in PC. Das Wort VLOC bringt den Inhalt dieses virtuellen Zuordnungszählers auf den Datenstack. LOCATE gibt die physische Adresse, unter der das nächste Byte abgespeichert wird.

Mnemonische Befehle werden mit angehängtem Komma geschrieben. Assembler-Anweisungen beginnen mit einem Punkt, z.B. .BYT.

Registerlisten bei den PUSH- und PULL-Befehlen werden aus dem Namen der Register mit vorangestelltem Komma gebildet, also:

,A	Akku A
,B	Akku B
,D	Akku D = Akku A + Akku B
,X	Register X
,Y	Register Y
,S	Hardware-Stackpointer
,U	Anwender-Stackpointer
,CC	Condition Code Register (Status)
,DP	Direct Page Register
,PC	Programmzähler
,ALL	Alle Register einschl. dem antivalenten Stackpointer

Die Registerlisten dürfen in beliebiger Reihenfolge aufgegeben werden, weil die o.a. Operatoren zusammengeodert werden. Bei den Befehlen TFR, und EXG, (Transfer und Exchange von Registern) gelten ebenfalls die vorstehenden Operatoren. Bei TFR, ist folgendeReihenfolgeeinzuhalten:

Source Destination TFR, z.B. ,D ,X TFR, übertrage von D nach X

Bei den auf den Status wirkenden Befehlen ORCC, und ANDCC, sowie CWAI, wird das zu setzende/zu löschende Bit durch einen vorangestellten Punkt gekennzeichnet. Hierzu haben wir folgende Operatoren:

.E	Entire-Flag
.F	Fast Interrupt Flag von FIRQ
.I	IRQ-Flag
.H	Half Carry Flag
.Z	Zero flag
.N	Negative Flag
.V	Overflow Flag
.C	Carry Flag

Auch hier ist die Reihenfolge der Eingabe beliebig

Nicht indizierte Befehle

Die 1-Byte 'Inherent'-Befehle brauchen selbstverständlich keinen Zusatz. Man schreibt also einfach NOP, RTS, ROLA, usw.

Bei der Adressierung von Zellen im Speicher wird ersatzweise (on default) davon ausgegangen, daß der Anwender 'extended' (absolut) adressieren möchte. 44 LDA, führt daher zu einem 3 Byte langen Befehl. Eine 'forcierte' Adressierung der Direct Page wird erreicht mit dem Operator <<

Die Verwaltung der Direct Page liegt dabei in der Verantwortung des Programmierers. Es findet keine Kontrolle statt, welche DP eingeschaltet ist.

Eine 'forcierte' Adressierung 'extended', die man wohl kaum benutzen wird, wird mit folgendem Operator erreicht: >< . Die Adressierungsart 'immediate' oder Direktoperand wird wie üblich erreicht mit dem #. Die Direktoperanden für ORCC und ANDCC sowie CWAI wurden weiter oben abgehandelt.

Indizierte Befehle

Der 6809 kennt eine Vielzahl von Indizierungsmöglichkeiten, direkt und indirekt. Dabei dürfen Offsets von 0, +-4, +-7 und +-15 Bit verwendet werden oder alternativ Pre-Dekrement und Post-Inkrement der Indexregister X, Y, S oder U. Bei den Offsets sind auch solche mit Offset in Akku A oder B erlaubt und solche relativ zum Programmzähler, um voll verschieblichen Code zu erhalte. Alle Möglichkeiten sind hier abgedeckt.

Indirekte Adressierung wird grundsätzlich durch eine rechte eckige Klammer (]) gekennzeichnet. Wo dieser Operator fehlt, da wird also direkt adressiert.

Der 6809 hat bekanntlich die Adressierungsart 'extended indirect', die bei den indizierten Adressierungsarten mit dem Postbyte 9F untergebracht ist. Sie wird mit folgendem Operator hereingerufen: >] .

Das indzierende Register wird immer mit einem vorangestellten Komma bezeichnet, also ,X, ,Y, ,S und ,U.

Die indirekte indizierte Adressierungsart wird mit folgendem Operator bewirkt: ,] .

Bei der Indizierung wird ersatzweise (on default) davon ausgegangen, daß ohne Offset (Offset = 0) indiziert werden soll, z.B. ,X LDA, (einfach indiziert mit X). Alle anderen Arten des Offsets müssen spezifiziert werden mit einem der nachfolgenden Operatoren:

,0	Offset=0, entbehrlicher Operator, gut aber für Dokumentation
,4	Offset in 4 Bit plus Vorzeichen im 5. Bit

,8 Offset in 7 Bit +- Vorzeichen
,16 Offset in 15 Bit +- Vorzeichen
,A Offset in Akku A
,B Offset in Akku B
,D Offset in Akku D (D=A+B)

Bei den Gruppen ,4 und ,8 prüft das System, ob der auf dem Stack übergebene zahlenmäßige Operand in die Größenordnung des Offsets hineinpaßt. Wenn nicht, dann erfolgt ein Warning, und es wird nur der Teil der Zahl vorzeichenrichtig verwertet, der den unteren 4 oder 7 Bit entspricht.

Ein Offset kann auch relativ zum Programmzähler der CPU spezifiziert werden. Damit erhält man relokativen Code. Hierfür haben wir folgende Operatoren:

.<PC 7 Bit +- Vorzeichen Offset zum PC
,PC 15 Bit +- Vorzeichen Offset zum PC
<PC wahlweiser Ausdruck für .<PC

Für die Register X, Y, S und U gibt es die Gruppe der Indizierungen mit Pre-Dekrement oder mit Post-Inkrement des indizierenden Registers, und zwar um 1 oder um 2. Und alles natürlich auch indirekt. Hierzu die Operatoren:

,X+ Post-Inkrement um 1
,X++ dito um 2
,X- Pre-Dekrement um 1
,X-- dito um 2
,X++] Post-Dekrement um 2, indirekt
,X--] Pre-Dekrement um 2, indirekt

Die letztgenannten Adressierungen dürfen bekanntlich nicht mit Offset benutzt werden. Beim späteren Eintippen der Programme könnte man die Kommas auch weglassen und die Minuszeichen sinnfälliger vor die Registernamen setzen.

In der direkten Adressierung ohne Index sind zwei weitere Operatoren nützlich, mit denen man den niedrigen oder den hohen Teil eines Speicherwortes (Adresse) unmittelbar laden kann:

#< lade unmittelbar den niederen Teil der Adresse
#> dito für den höherwertigen Teil

Strukturierte Statements und ihre Verzweigungsanweisungen

Folgende Kontrollstrukturen sind implementiert. Die Schlüsselworte werden immer mit nachgestelltem Komma geschrieben, um sie vom Kern des FORTH abzuheben.

BEGIN,	UNTIL		Schleife bis zum Erreichen einer Bedingung
BEGIN,	AGAIN,		endlose, bedingungslose Schleife
BEGIN,	WHILE,	REPEAT,	Ausführung ab BEGIN, solange die Bedingung bei WHILE, zutrifft. Danach Fortsetzung hinter REPEAT
IF,	THEN,		Bedingte einfache verzweigung
IF,	ELSE,	THEN,	2-Wege-Verzweigung

Diese Kontrollstrukturen erzeugen relokativen Branch-Code. An den Stellen UNTIL, AGAIN, oder REPEAT, werden automatisch Long Branches eingesetzt, wenn die Verzweigungsweite für einen einfachen Branch-Befehl zu weit geworden ist. - Es wird gelegentlich Fälle geben, wo zwischen IF, und THEN, oder zwischen WHILE, und REPEAT, ein so großer Code-Block liegt, daß mit Long Branches gearbeitet werden muß. Dafür gibt es die Strukturen BEGIN, LWHILE, REPEAT, ferner LIF, THEN, bzw. LIF, LELSE, THEN,. REPEAT, und THEN, erzeugen immer die notwenige Verzweigungsweite.

Alle Kontrollstrukturen dürfen ineinander verschachtelt werden, solange man die innenliegenden immer zuerst auflöst. Falsche Abfolgen erzeugen eine Warnung. Der Datenstack wird zerstört. Man muß dann mit ORG bis zu der höchsten noch nicht aufgelösten Struktur zurückgehen.

Zu den Verzweigungbedingungen im Zusammenhang mit den Strukturen: Das Statement .NE

(Not Equal) WHILE, führt zum Opcode BEQ. Die Schleife wird also solange ausgeführt, wie das Z-Flag noch nicht gesetzt ist. Gleiches gilt für die anderen antivalenten Verzweigungsbedingungen. Hier die Operatoren:

.GT	Greater Than
.LE	Less or Equal
.GE	Greater or Equal
.LT	Less Than
.NE	Not Equal
.EQ	Equal Zero
.VC	Overflow Clear
.VS	Overflow Set
.CC	Carry Clear
.CS	Carry Set
.+	Positive
.–	Negative
.SLT	Signed Less Than (vorzeichenbehaftet weniger als)
.SGE	Signed Greater or Equal (vorzeichenbehaftet größer oder gleich)
.SLE	Signed Less Equal (dito weniger oder gleich)
.SGT	Signed Greater Than (dito, größer)

Verzweigungsbefehle außerhalb strukturierter Statements

Es sind alle kurzen und langen Verzweigungsbefehle mit ihren Mnemonics implementiert, also z.B. BRA, LBRA, BRS, LBSR, LBNE usw. Das möge nicht zu unstrukturierten Programmen verleiten, mag aber insbesondere bei Vorwärtsreferenzen gelegentlich nützlich sein. Alle diese Befehle setzen ein Argument auf dem Datenstack voraus, das die Adresse des anzuspringenden Labels oder Symbols ist. Die Branches rechnen diese Adresse in eine relative Sprungweite um. Man wird sich vor allem der Long Branches bedienen, um außerhalb liegende Systemroutinen zu erreichen.

Assembler-Anweisungen (Direktiven)

Es sind folgende Dienstleitungen als Befehlsworte implementiert, sie beginnen mit wenigen Ausnahmen mit einem Punkt:

ORG	Setzen des Zuordnungszählers auf den virtuellen Speicherbereich der Zielmaschine
.FCB	Form Constant Byte. Der Befehl muß wiederholt werden, wenn eine Reihe von Bytes abgelegt werden soll
.BYT	Wahlweiser Ausdruck für .FCB, Byte-Ablage
.FDB	Form Double Byte, Ablage eines Wortes mit 2 Bytes
.WOR	Wahlweiser Ausdruck für .FDB
.RMB	Reserve Memory Bytes. Hochzählen des Zuordungszählers um die Zahl, die auf dem Datenstack übergeben wurde
.FCC	Form Constant Characters. Ablage eines ASCII-Strings im Speicher. Der String wird nach rechts durch Gänsefüße oder ein RETURN begrenzt
"	Wahlweise für .FCC, Stringablage
.EQU	Equate-Anweisung. Erzeugt eine FORTH-Konstante, die besonders als externes Symbol oder als Label im Programmverlauf benutzt werden kann. Sie legt bei Benutzung des Namens den Wert der Konstanten auf den Datenstack
LABEL	Erzeugt wie .EQU eine Konstante, die z.B. Sprungziel werden kann. Gebrauch: VLOC LABEL Name
BIN %	Beide Operatoren stellen die Zahlenbasis bis zum Ende der Zeile auf binär um. Die Folgezeile versteht also wieder Hex-Eingaben.
T	AIM-spezifische Anweisung, die auf die TOP-Zeile des Editors führt
LOCATE	Bringt die RAM-Adresse auf den Datenstack, in die die nächste Abspeicherung von Zielcode erfolgt. Kann aber auch benutzt werden, um einem Bereich vor dem Brennen eines EPROMs auf hex FF zu bringen: LOCATE HEX 1000 FF FILL
.OPTLIS und .OPTNOL	Ausgabe einer Assembler-Liste oder nicht, AIM-spezifisch ein gewisses Provisorium

Systemkonstante

Die Interrupt- und Reset-Vektoradressen sind als Konstante implementiert, die mit der Benutzung ihres Namens folgende Adressen auf den Datenstack bringen:

FFFE	Reset;
FFFC	NMI
FFFA	SWI
FFF8	IRQ
FFF6	FIRQ
FFF4	SWI2
FFF2	SWI3

Die Befehlsworte WR1 und WR2 schreiben 1 Byte bzw. 1 Wort vom Datenstack an die aktuelle Stelle in das RAM des Zielcodes. Mit Ihnen ist es z.B, möglich, ganze Ketten von Bytes oder Worten mit einer DO-LOOP einzuschreiben.

Hier zunächst das in FORTH formulierte Assemblerprogramm. Daran schließen sich Formulierungsbeispiele an, die keinerlei Programm bilden.

```
( TOP)
( FORTH-ASSEMBLER FUER DEN MC6809 VON MOTOROLA)
( COPYRIGHT BY ROLAND LOEHR, 1983)

FORGET TASK
6 ALLOT ( ROOM FOR 3 VARIABLES)

: MESS CR ." 6809-ASSEMBLER V1.1" CR
  ." COPYRIGHT 1983 BY ROLAND LOEHR" CR ;
MESS
HEX
F6CF VARIABLE EDI    ( T-ENTRY INTO EDITOR ADDR)
: T EDI EXECUTE ; ( AIM 65 SPEZIFISCH)
: % 2 BASE ! ;


VOCABULARY ASS  IMMEDIATE ASS DEFINITIONS

300 CONSTANT PC   ( PC OF MC6809 ASSEMBLER)
302 CONSTANT RAMAREA    ( ALLOCATE ROOM FOR COMPILATION)
304 CONSTANT POSTBYTE
0 VARIABLE LISTEN

CREATE LIST 20 C, 27 C, F7 C, 4C C, 5A C, B0 C,
SMUDGE    ( AIM-SPEZIFISCH)
: ?LIST LISTEN @ IF LIST THEN ;   ( SHALL I LIST?)
: .OPTLIS 1 LISTEN ! ;   ( LIST ON)
: .OPTNOL 0 LISTEN ! ;   ( OFF)

( INTERRUPT VECTORS)

FFFE CONSTANT RESET
FFFC CONSTANT NMI
FFFA CONSTANT SWI
FFF8 CONSTANT IRQ
FFF6 CONSTANT FIRQ
FFF4 CONSTANT SWI2
FFF2 CONSTANT SWI3
```

```
( HILFSWOERTER)
: VLOC PC @ ;    ( GET VIRTUAL ADDRESS)
: ?RAM+ RAMAREA @ + ;
: LOCATE VLOC ?RAM+ ;
: <REL VLOC 1+ - ;    ( OFFSET FOR SHORT BRANCHES)
: REL VLOC 2+ - ;    ( CALCULATE PC-OFFSET)


: @MODE MODE @ ;    ( FETCH THE MODE WORD)
: !MODE MODE ! ;    ( STORE TO MODE)
: ORMODE @MODE OR !MODE ;    ( OR BYTE TO MODE)

: !POST POSTBYTE ! ;
: OMODE 0 !MODE 0 !POST VLOC CR ." *=" 0 D. ?LIST ;
: LO OMODE ?LIST ;
: ASSIGN RAMAREA ! ( DEFINE RAMAREA FOR OPCODE)

  CR ." WARNING: SCRAMBLING OF OTHER RAM POSSIBLE" CR
  OMODE ;
: C@POST POSTBYTE C@ ;
: ERR OMODE CR 3 ERROR ;    ( OUTPUT ERROR MESSAGE)

( AUSGABE VON WARNMELDUNGEN)
: WARN ."  AT LOCATION " PC @ 0 D. CR ;
: WARN1 CR ." 2-BYTE OPERAND" WARN ;
: WARN2 CR ." IMMEDIATE MODE NOT ALLOWED" WARN ;
: WARN3 CR ." OFFSET TO INDEX NOT MATCHING" WARN ;
: WARN4 ." CONFLICT IN REGISTERS/ADDRESSING?" WARN ;
: WARN5 CR ." BRANCH OUT OF RANGE" WARN CR ;

: FFAND FF00 AND ;    ( SEE IF 2-BYTE OPERAND)
: MAND @MODE AND ;    ( SEE WHICH ADDRESS MODE)

: WR1 HEX DUP LOCATE C! 0 D. SPACE 1 PC +! ;
: WR2 HEX DUP DUP FFAND IF ELSE 30 EMIT 30 EMIT THEN 0 D.
  SPACE
  0 100 U/ SWAP 100 * + LOCATE ! 2 PC +! ;
: WRB 0 100 U/ SWAP 100 * + SWAP ?RAM+ ! ; ( LBRA BACK)
: WR01 DUP FFAND IF WARN1 WR1 ELSE WR1 THEN ;
( WRITE & OUTPUT MESSAGE: 2-BYTE OPERAND)

: FFMOD WR1 MODE C@ WR1 OMODE ;   ( WRITE REG LIST FOR TFR)

: -PC VLOC 1- PC ! LOCATE C@ ;    ( FETCH LAST BYTE)

: LBCORR    ( MAKE SHORT BRANCH TO LONG FOR LIF, ETC.)
  -PC DUP 20 =
  IF DROP 16
  ELSE 10 WR1 ( PREBYTE 10)
  THEN WR1 VLOC 0 WR2 ; ( OPCODE & 0-OFFSET PRELIMINARY)

: CORR ( MAKE SHORT BRANCH TO LONG BRANCH)
-PC DUP 20 =   ( BRA-OPCODE)
( 2) IF DROP 1- 16    ( MAKE IT LBRA)
( 2) ELSE DUP 8D =    ( BSR?)
( 3) IF DROP 1- 17    ( MAKE LBSR)
( 3 ) ELSE 10 WR1 SWAP 2 - SWAP   ( PREBYTE OF LBRANCH)
( 3) THEN
( 2) THEN
```

```
   WR1 WR2
OMODE
;

: BACK <REL DUP ( CALCULATE OFFSET TO BEGIN)
   FF80 U< ( OUT OF RANGE?)
( 1) IF CORR
( 1) ELSE FF AND WR1
( 1) THEN
;

( DEFINING WORDS FUER DIVERSE BEFEHLSGRUPPEN)

: BR    ( DEFINING WORD FOR SHORT BRANCHES)
   <BUILDS C, DOES>
   C@ WR1 <REL DUP DUP ( OFFSET)
0< ( 1) IF FF80 U< ( 2) IF WARN5 ( 2) THEN
( 1) ELSE 7F > ( 3) IF WARN5 ( 3) THEN
( 1) THEN
FF AND WR1
OMODE
;

: LB    ( DEFINING WORD FOR 2-BYTE OPCODES)
<BUILDS , DOES>
@ WR2 REL WR2 OMODE ;

: TRANS    ( FOR TFR & EXG)
  WR1 POSTBYTE 1+ C@ WR1 OMODE ;

: INH ( DEFINING WORD FOR 1-BYTE OPCODES)
   <BUILDS C, DOES>
   C@ WR1 OMODE ;    ( FETCH OPCODE, LAY DOWN)


: INDEX    ( BELONGS TO DEFINING WORD MT)
EOO MAND   ( CHECK FOR UNALLOWED ADRESSING)
( -1) IF WARN4
( -1) THEN
WR1    ( OPCODE)
C@POST DUP 9F =
( 0) IF WR1 WR2    ( EXTENDED INDIRECT)
( 0) ELSE DUP 80 <    ( 4-BIT OFFSET)
( 1) IF 2000 MAND    ( IS OFFSET SPECIFIED?)
( 2) IF DUP OF AND   ( CHECK POSTBYTE)
( 3) IF    ( OK, THERE ARE BITS)
( 3) ELSE    ( MUST BE 4-BIT OFFSET)
      OVER 8000 AND ( NEGATIVE?)
( 3A) IF SWAP OF AND OR 10 OR
( 3A) ELSE ( POSITIVE) OVER 70 AND ( TOO LARGE?)
( 3B) IF WARN3 SWAP OF AND OR
( 3B) ELSE OR
( 3B) THEN
( 3A) THEN
( 3) THEN
( 2) ELSE 84 OR    ( DEFAULT 0 OFFSET)
( 2) THEN
     WR1
```

```
( 1) ELSE    ( POSTBYTE > 80, NOW LOOK FOR OTHER INDEXING)
     DUP ( POSTBYTE) 08 AND NOT
     ( SEE IF ,A ,B, ,D ,0 ,R+ ,R++ EVF.)
( 4) IF WR1    ( YES, ALL SPECIFIED)
( 4) ELSE DUP 04 AND    ( SEE IF RELATIVE TO PC)
( 5) IF DUP ( POSTBYTE) 01 AND ( 16-BIT?)
) IF WR1 REL WR2
( 6) ELSE OVER REL DUP 0<    ( THE OFFSET TO PC)
( 7) IF FF80 U<    ( OUT OF BOUNDS?)
( 7A) IF 01 OR WR1 REL WR2 WARN3    ( 16-BIT)
( 7A) ELSE WR1 REL 1+ FF AND WR1
( 7A) THEN
( 7) ELSE    ( OFFSET >0)
     7F >
( 7B) IF 01 OR WR1 REL WR2 WARN3    ( MADE TO-BIT)
( 7B) ELSE WR1 REL 1+ WR1
( 7B) THEN

( 7) THEN
( 6) THEN
( 5) ELSE DUP    ( POSTBYTE) 01 AND
( 8) IF WR1 WR2    ( EXPLICIT 16-BIT)
( 8) ELSE OVER FF > NOT
( 9) IF ( ALL OKAY) WR1 WR1
( 9) ELSE WARN3 01 OR WR1 WR2    ( MAKE IT 16 BIT)
( 9) THEN
( 8) THEN
( 5) THEN
( 4) THEN
( 1) THEN
( 0) THEN
;

: MIMODE    ( USED BY DEFINIG WORDS TO CHECK ADDRESSING)
1000 MAND    ( SEE IF INDEXED)
( 0) IF 20 + INDEX
( 0) ELSE 0400 MAND
( 1) IF ( #-MODE) 8000 MAND
( 2) IF WARN2 2 DROP    ( # NOT ALLOWED)
( 2) ELSE 4000 MAND
( 2A) IF WR1 WR2    ( ALWAYS 2 BYTES TO STORE)
( 2A) ELSE WR1 WR01    ( 1 BYTE)
( 2A) THEN
( 2) THEN
( 1) ELSE 3FFF MAND 0=
( 3) IF    ( DEFAULT EXTENDED MODE)
     30 + WR1 WR2
( 3) ELSE 0200 MAND
( 4) IF    ( FORCED EXTENDED) 30 + WR1 WR2
( 4) ELSE 0800 MAND
( 5) IF    ( DIRECT PAGE)
     10 + WR1 WR01
( 5 4 3 1) THEN THEN THEN THEN
( 0) THEN
OMODE
;
```

```
: MIMODE    ( USED BY MI TO CHECK ADRESSING MODE)
1000 MAND    ( SEE IF INDEXED)
( 0) IF 60 + INDEX
( 0) ELSE 0400 MAND
( 1) IF WARN2
( 1) ELSE FFFF MAND 0=
( 2) IF 70 + WR1 WR2    ( DEFAULT EXTENDED)
( 2) ELSE 0200 MAND
( 3) IF 70 + WR1 WR2    ( FORCED EXTENDED)
( 3) ELSE 0800 MAND
( 4) IF WR1 WR01    ( DIRECT PAGE)
( 4 3 2 1 ) THEN THEN THEN THEN
( 0) THEN
OMODE
;

: MI    ( DEFINIG WORD)
   <BUILDS C, DOES>
   C@ MIMODE
;

: MI ( DEFINING WORD)
   <BUILDS C, DOES>
C@    ( BASIS OPCODE)
MIMODE
;


: MAC1 8000 ORMODE    ( INDICATE STORES)
  MIMODE  ;
: MAC2 4000 ORMODE    ( INDICATE 16-BIT LOAD)
   MIMODE  ;
: MAC11 10 WR1 MAC1 ;    ( STORES WITH PREBYTE 10)
: MAC12 10 WR1 MAC2 ;    ( NORMAL INSTRUCTIONS)
: MAC22 11 WR1 MAC2 ;

: CON    ( DEFINING WORD)
  <BUILDS C, DOES>
  C@ ORMODE ;   ( SET BIT)

: PBOR POSTBYTE @ OR !POST ;    ( OR TOGETHER BITS)

( OPERATORS FOR REGISTER LIST PUSH/PULL/INDEXING)

: REG <BUILDS , , DOES>    ( LAY DOWN 4 BYTES)
   DUP @  DUP @MODE FF AND AND
   IF WARN4
   THEN
   FF MAND ( FLAG) >R
   ORMODE
   2+ @ R> 0= NOT    ( FLAG BACK, POSTBYTE BELOW)
   IF DUP FFF AND PBOR F000 AND 0 10 U/ SWAP DROP
   THEN
   PBOR ;
```

```
( BRANCHING INSTRUCTIONS)

20 BR BRA,    21 BR BRN,    22 BR BHI,    23 BR BLS,
24 BR BHS,    24 BR BCC,
25 BR BLO,    25 BR BCS,
26 BR BNE,    27 BR BEQ,    28 BR BVC,
29 BR BVS,    2A BR BPL,    2B BR BMI,    2C BR BGE,
2D BR BLT,    2E BR BGT,    2F BR BLE,

8D BR BSR,

: LBRA, 16 WR1 REL WR2 OMODE ;
: LBSR, 17 WR1 REL WR2 OMODE ;

1021 LB LBRN, 1022 LB LBHI, 1023 LB LBLS, 1024 LB LBHS,
1024 LB LBCC, 1025 LB LBLO, 1025 LB LBCS,
1026 LB LBNE, 1027 LB LBEQ, 1028 LB LBVC,
1029 LB LBVS, 102A LB LBPL, 102B LB LBMI, 102C LB LBGE,
102D LB LBLT, 102E LB LBGT, 102F LB LBLE,

: SWI2, 103F WR2 OMODE ;    ( SOFTWARE INTERRUPT
: SWI3, 113F WR2 OMODE ;

: TFR, 1F TRANS ;
: EXG, 1E TRANS ;

( ASSEMBLER DIRECTIVES)

: ORG PC !    ( SET PC OF ASSEMBLER)
   CR
   ." WARNING: PC MAY EXCEED ASSIGNED RAMAREA"
    0 !POS1
OMODE
;

: .FCB WR1 OMODE ;    ( WRITE 1 BYTE TO MEMORY)
: .BYT WR1 OMODE ;    ( DITTO)
: .FDB WR2 OMODE ;    ( WRITE WORD TO MEMORY)
: .WOR WR2 OMODE ;    ( DITTO)
: .RMB PC +! OMODE ;    ( RESERVE MEMORY BYTES)
: " 22 WORD HERE C@ 0 DO HERE 1+ I + C@ WR1 LOOP OMODE ;
: .FCC " ;    ( SAME WORD)
: .EQU <BUILDS , OMODE DOES> @ ;    ( FORM A CONSTANT)
: LABEL .EQU ;    ( FORM A CONSTANT)
: BYT WR1 OMODE ;
: WOR WR2 OMODE ;
: RMB PC +! OMODE ;
: EQU .EQU ;
: SPC CR ?LIST ;

: .SPC SPC ;    ( ADVANCE TO NEXT LINE)

( SET BIT IN CCR)

80 CON .E    40 CON .F    20 CON .H    10 CON .I
08 CON .L    04 CON .Z    02 CON .V    01 CON .C

: ORCC, 1A WR1 @MODE WR1 OMODE  ;    ( WRITE TO MEM)
: ANDCC, 1C WR1 @MODE FF XOR WR1 OMODE  ;    ( M OFF)
: CWAI, 3C WR1 MODE FF XOR WR1 OMODE ;  ( MASK OFF)
```

```
( OPERATORS TO DEFINE THE ADRESSING MODE)

0010 0000 REG ,]    ( INDIRECT MODE, NO INDEX)
009F 1000 REG >]    ( EXTENDED INDIRECT)

0080 1000 REG ,X+    ( AUTO INCREMENT/DECREMENT GROUP)
0081 1000 REG ,X++
0082 1000 REG ,X-
0083 1000 REG ,X--
0091 1000 REG ,X++]    ( INDIRECT)
0093 1000 REG ,X--]

00A0 1000 REG ,Y+
00A1 1000 REG ,Y++
00A2 1000 REG ,Y-
00A3 1000 REG ,Y--
00B1 1000 REG ,Y++]
00B3 1000 REG ,Y--]

00C0 1000 REG ,U+
00C1 1000 REG ,U++
00C2 1000 REG ,U-
00C3 1000 REG ,U--
00D1 1000 REG ,U++]
00D3 1000 REG ,U--]

00E0 1000 REG ,S+
00E1 1000 REG ,S++
00E2 1000 REG ,S-
00E3 1000 REG ,S--
00F1 1000 REG ,S++]
00F3 1000 REG ,S--]

1000 1010 REG ,X
2020 1020 REG ,Y
3040 1040 REG ,U
4060 1040 REG ,S

0084 2000 REG ,0    ( 0-OFFSET TO REGISTER XYUS)
0000 2000 REG ,4
0088 2000 REG ,8
0088 2000 REG ,<    ( 8-BIT OFFSET)
0089 2000 REG ,16    ( 16-BIT OFFSET)
0089 2000 REG ,><    ( 16-BIT OFFSET)
508D 1080 REG ,PC
008C 1080 REG <PC
008C 1080 REG ,<PC    ( 8-BIT OFFSET)

( ADDRESSING OF VEKTORS)

: #< FF AND 800 ORMODE ;    ( LOWER PART)
: #> 0 100 U/ SWAP DROP ;    ( HIGHER PART)

8086 0002 REG ,A
9085 0004 REG ,B
008B 0006 REG ,D
A000 0001 REG ,CC
B000 0840 REG ,DP
```

```
( COMPACT TERM TO FORM REGISTER LIST)

0000 00FF REG .ALL

( FORCED ADDRESSING MODES)
0000 0800 REG << ( DIRECT PAGE MODE)
0000 0200 REG >< ( EXTENDED ADDRESSING MODE)
0000 0400 REG # ( IMMEDIATE ADDRESSING MODE)

( CONDITIONALS FOR STRUCTURED STATEMENENTS WITH)
( WHILE, UNTIL, ETC.)
23 INH .GT ( WHILE GREATER THAN)
22 INH .LE ( WHILE LESS OR EQUAL)
25 INH .GE ( GREATER OR EQUAL)
24 INH .LT ( LESS THAN)
27 INH .NE ( WHILE NOT EQUAL)
26 INH .EQ ( EQUAL ZERO)

29 INH .VC ( WHILE OVERFLOW CLEAR)
28 INH .VS ( WHILE OVERFLOW SET)

25 INH .CC ( WHILE CARRY CLEAR)
24 INH .CS ( CARRY SET)
2B INH .+ ( WHILE POSITIVE)
2A INH .- ( WHILE NEGATIVE)

2C INH .SLT ( WHILE SIGNED LESS THAN)
2D INH .SGE ( SIGNED GREATER OR EQUAL)
2E INH .SLE ( SIGNED LESS EQUAL)
2F INH .SGT ( SIGNED GREATER THAN)

( 1-BYTE INHERENT INSTRUCTIONS)

12 INH NOP,
13 INH SYNC,
19 INH DAA,
1D INH SEX,
39 INH RTS,
3A INH ABX,
3B INH RTI,
3D INH MUL,
3F INH SWI,

40 INH NEGA,
43 INH COMA,
44 INH LSRA,
46 INH RORA,
47 INH ASRA,
48 INH ASLA,    48 INH LSLA,
49 INH ROLA,
4A INH DECA,
4C INH INCA,
AD INH TSTA,
4F INH CLRA,

50 INH NEGB,
53 INH COMB,
54 INH LSRB,
56 INH RORB,
```

```
57 INH ASRB,
58 INH ASLB,    58 INH LSLB,
59 INH ROLB,
5A INH DECB,
5C INH INCB,
5D INH TSTB,
5F INH CLRB,

00 MI NEG, 03 MI COM, 04 MI LSR, 06 MI ROR,
07 MI ASR, 08 MI ASL, 08 MI LSL, 09 MI ROL,
0A MI DEC, 0C MI INC, 0D MI TST, 0F MI CLR,

0E MI JMP,

( 8-BIT OPERATIONS IN ACCA & ACCB)

80 MI SUBA, 81 MI CMPA, 82 MI SBCA, 84 MI ANDA,
85 MI BITA, 86 MI LDA,     88 MI EORA,
89 MI ADCA, 8A MI ORA,   8B MI ADDA,
8D MI JSR,

C0 MI SUBB, C1 MI CMPB, C2 MI SBCB, C4 MI ANDB,
C5 MI BITB, C6 MI LDB,     C8 MI EORB,
C9 MI ADCB, CA MI ORB,   CB MI ADDB,

: STA, 87 MAC1 ;
: STB, C7 MAC1 ;
: STX, 8F MAC1 ;
: STY, 8F MAC11 ;
: STD, CD MAC1 ;
: STU, CF MAC1 ;
: STS, CF MAC11 ;

: SUBD, 83 MAC2 ;
: CMPX, 8C MAC2 ;
: LDX, 8E MAC2 ;

: CMPD, 83 MAC12 ;
: CMPY, 8C MAC12 ;
: LDY, 8E MAC12 ;

: CMPU, 83 MAC22 ;
: CMPS, 8C MAC22 ;

: ADDD, C3 MAC2 ;
: LDD, CC MAC2 ;
: LDU, CE MAC2 ;
: LDS, CE MAC12 ;

: LEAX, 30 INDEX OMODE ;
: LEAY, 31 INDEX OMODE ;
: LEAS, 32 INDEX OMODE ;
: LEAU, 33 INDEX OMODE ;

: PSHS, 34 FFMOD ;
: PULS, 35 FFMOD ;
: PSHU, 36 FFMOD ;
: PULU, 37 FFMOD ;
```

```
( CONTROLLLING STRUCTURES)

: AGAIN, ?EXEC 1 ?PAIRS
   20 WR1 ( BRA-OPCODE) BACK
CR
   ?LIST
; IMMEDIATE

: BEGIN, VLOC 1 OMODE !

: WHILE, ?EXEC 1 ?PAIRS 0 WR1 VLOC 3
OMODE
; IMMEDIATE

: REP
   VLOC 2+    ( GET OFFSET)
OVER - DUP 80 < IF SWAP ?RAM+ 1- C!    (    < $80)
     ELSE 3 ERR THEN
;

: IF, VLOC 0 WR1 ( FOR OFFSET) 2 ( FLAG) OMODE ! IMMEDIATE

: THENN VLOC 1- OVER - DUP 80 <
   IF SWAP ?RAM+ C!
   ELSE 3 ERR THEN
;

: REPEAT, ?EXEC
  DUP ( FLAG) 4 >
  IF ( LWHILE) 7 ?PAIRS
   DUP ( LOCATION TO STORE)
   16 WR1 ( LBRA) VLOC SWAP - ( CALC. OFFSET)
   WRB
   VLOC - 2- WR2    ( WRITE LBRA BACK)
  ELSE
   REP 20 FF AND WR1 ( BRA-OPCODE)
   BACK
  DROP
  THEN
OMODE
;

: UNTIL, ?EXEC 1 ?PAIRS
   BACK
OMODE
; IMMEDIATE

: THEN, DUP ( FLAG)
4 >
( 1) IF DUP ( FLAG) 5 =
( 2) IF DROP DUP VLOC SWAP - 2- ( OFFSET) WRB
( 2) ELSE 6 ?PAIRS
     DUP DUP VLOC SWAP - 2- WRB
     OVER - WRB ( OFFSET TO LIF,-STATEMENT)
( 2) THEN
```

```
( 1) ELSE DUP 2 = IF ( WAS `IF.) DROP THENN ( U'SRT OFFSET)
   ELSE 4 ?PAIRS ( SEE IF ELSE, PRECEEDED)
      2DUP THENN - SWAP ?RAM+ C! ( SKIP FROM IF TO ELSE)
   THEN
( 1) THEN
OMODE
;

: ELSE, 2 ?PAIRS 20 WR1 ( OPCODE OF BRANCH ALWAYS)
  VLOC 4 ( FLAG) 0 WR1 OMODE ; IMMEDIATE ( OFFSET 0-DEFLT)
: LIF, LBCORR 5 CR ?LIST ; ( LONG BRANCH, FLAG=5)

: LELSE, 5 ?PAIRS 20 WR1 LBCORR 6 CR ?LIST ;

: LWHILE, ?EXEC 1 ?PAIRS LBCORR 7 OMODE ; ( L-BRANCH)

HERE FENCE !
: TASK ;
ASS
0 ORG 3000 ASSIGN
.OPTLIS
FINIS
```

```
CR
." HANTIERUNGSBEISPIELE"
CR

?LIST
.OPTLIS
?LIST
?LIST
F000 ORG    ( ZUORDNUNGSZAEHLER VIRTUELLER SPEI-HERRAUM)
4000 ASSIGN    ( ABLAGEBEREICH DADURCH $3000)
NOP,    ( INHERENT)
SEX,
RTS,
SPC
55 # LDA,    ( IMMEDIATE)
4567 # LDX,
ABCD # LDD,
SPC
.C ORCC,    ( SETZEN DES CARRY-FLAG)
.C ANDCC,    ( LOESCHEN DES CARRY)
SPC
,D ,X TFR,    ( REGISTER D NACH X)
,A ,B EXG,    ( VERTAUSCHUNG A UND B)
SPC
3456 LDX,    ( DEFAULT EXTENDED)
55 LDA,    ( = DEFAULT EXTENDED, 3 BYTE)
44 << LDA,    ( FORCIERT DIRECT PAGE)
55 >< LDS,    ( FORCIERT EXTENDED)
SPC
1234 >] LDX,    ( EXTENDED INDIRECT)
>] 1234 LDX,    ( REIHENFOLGE DER OPERATOREN BELIEBIG)
SPC
,A ,CC PSHS,    ( REGISTERLISTEN)
,ALL PULU,    ( ALLE REGISTER)
SPC
3 ,4 ,X LEAX,    ( X=X+3 INDIZIERTE BEFEHLE)
```

```
,4 ,X 3 LEAY,    ( WAHLFREIE ABFOLGE)
,X LDA,    ( INDIZIERT, DEFAULT OFFSET=0)
,0 ,X LDB,    ( DITO)
-5 ,4 ,X LDU,    ( INDIZIERT, 4-BIT OFFSET)
45 ,8 ,X LDA,    ( +- 7 BIT OFSET)
-1000 ,16 ,X LDA,    ( +- 15 BIT OFFSET)
SPC
F000 ,(PC LDA,    ( ADRESSE F000 RELATIV ZU PC)
1000 ,PC LDA,    ( VORWAERTSREFERENZ ZUR ADRESSE 1000)
SPC
,A ,X LDA,    ( INDIZIERT MIT OFFSET IM AKKU A)
5 ,4 ,] ,X LDA,    ( INDIZIERT INDIREKT MIT OFFSET 5 ZU X)
SPC
,X+ LDA,
( POSTINKREMENT)
,X++] LDU,    ( POSTINKREMENT INDIREKT)
,U-- LDD,    ( PREDEKREMENT)
SPC
FF .FCB  ( ABLAGE EINES BYTES)
FE .BYT
1234 .FDB    ( WORT-ABLAGE)
BCDE .WOR    ( DITO)
SPC
10 .RMB    ( RESERVIERUNG VON SPEICHERRAUM)
SPC
.FCC STRING"    ( ABLAGE EINES STRINGS)
" STRING"    ( DITO)
SPC
% 111 .BYT    ( BINAERES BYTE)
SPC
VLOC LABEL ZULU
ZULU BRA,    ( BRANCH ALWAYS  TO ZULU)
SPC
ZULU JMP,
SPC
F800 .EQU AUSGABE
AUSGABE LBSR,
SPC
BEGIN,
55 LDA,
.NE
WHILE,
55 DEC,
REPEAT,    ( LEERE DEKREMENTIERUNGSSCHLEIFE)
SPC
BEGIN,
55 DEC,
.EQ
UNTIL,
SPC
BEGIN,
NOP,
NOP,
AGAIN,    ( ENDLOSE SCHLEIFE)
SPC
55 LDA,
.GT
IF,
```

```
56 DEC.
THEN.
SPC
55 LDA.
.NE
LIF.
SPC
F300 ORG  ( GROSSER ZWISCHENRAUM ZW. IF. ELSE. THEN)
SPC
LELSE.
SEX.
F500 ORG
SPC
THEN.
SPC
FINIS
( BOTTOM)
```

Die vorstehenden Hantierungsbeispiele erzeugen folgenden Hexcode im RAM:

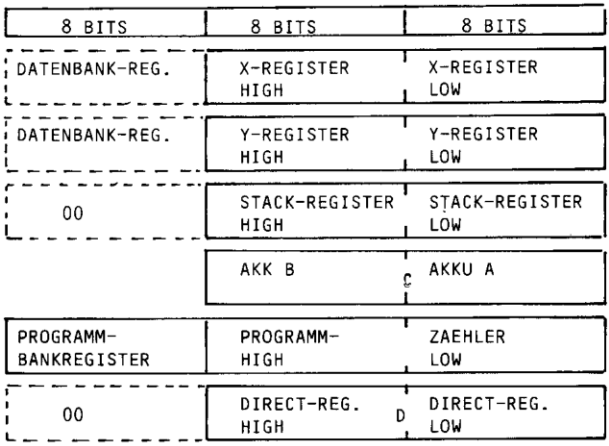
```
<M>=3000 12 1D 39 86 55 8E 45 67 CC AB CD 1A 01 1C FE 1F
< > 3010 01 1E 89 BE 34 56 B6 00 55 96 44 10 FE 00 55 AE
< > 3020 9F 12 34 AE 9F 12 34 34 03 37 FF 30 03 31 03 A6
< > 3030 84 E6 84 EE 1B A6 88 45 A6 89 F0 00 A6 8C C1 A6
< > 3040 8D 1F 8D A6 86 A6 15 A6 80 EE 91 EC C3 FF FE 12
< > 3050 34 8C DE FF FF FF FF FF FF FF FF FF FF FF FF FF
< > 3060 FF FF FF 53 54 52 49 4E 47 53 54 52 49 4E 47 07
< > 3070 20 FE 7E F0 70 17 07 88 B6 00 55 27 00 7A 00 55
< > 3080 20 FB 7A 00 55 26 FB 12 12 20 FC B6 00 55 23 03
< > 3090 7A 00 56 B6 00 55 10 27 02 69 FF FF FF FF FF FF
< > 30A0 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
<M>=3300 16 01 FD 1D FF FF FF FF FF FF FF FF FF FF FF FF
<M>=3500 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

Roland Löhr

CPU 65816 mit 16 Bit

Übersicht

Das Western Design Center hat zwei neue Zentraleinheiten angekündigt, nämlich den W65SC802 und den W65SC816. Es handelt sich um Prozessoren, die im OXI-CMOS-Verfahren hergestellt werden und die von 1 bis 10 MHz Takt schnell sind. Ersterer ist Pin zu Pin kompatibel zum 6502, der andere führt besondere Signale für die Systemsteuerung heraus. Bemerkenswert ist hier, daß auf dem Datenbus während der Taktphase Phi 1 eine Bankadresse gesendet wird, bei Phi 2 wie üblich dann der Datentransport. Damit ist der 65816, wie er hier verkürzt bezeichnet werden soll, in der Lage, 256x64 KB zu adressieren, das sind 16 Megabyte! Und das mit einem 40poligen Gehäuse. Die Prozessoren können im 8 Bit-Modus und in einem mit 16 Bit gefahren werden. Die Umsteuerung erfolgt per Software. Im 8 Bit-Modus sind die Prozessoren mit allen Befehlen und Adressierungsarten zur 6502 kompatibel, haben jedoch bereits dann einen erweiterten Befehlssatz. Im 16 Bit Modus wirken einige Befehle anders. Bei diesen neuen Befehlen ist eine Teilmenge deckungsgleich mit denen der Rockwell CPU R65C02. Nicht enthalten sind Befehle zum Setzen und Löschen von Bits und Verzweigungen in Abhängigkeit von Bits in der Zero Page (letztere Gruppen sind vor allem für die Steuerung besonders bei Einchippern interessant). Dafür tritt ein ganzes Arsenal neuer Befehle und Adressierungsarten hinzu, die die Programmiermöglichkeiten



DIE REGISTER DER CPU W65SC816

erheblich erweitern und eleganter machen. - Mit der internen 16 Bit-Architektur und dem vervielfachten Adressierungsraum ist der 65816 so interessant, daß hier möglichst früh ein erster Überblick gegeben werden soll, wobei das Datenblatt sicher noch nicht alle Einzelheiten nennt. Alle Angaben werden daher unter Vorbehalt gemacht. - Die Prozessoren sollen ab Anfang 1985 verfügbar sein. Im Moment ist noch kein endgültiges Datenblatt beziehbar, man möchte auch unnötige Nachfragen vermeiden, solange noch nicht alles festliegt.

Bei den neuen Prozessoren sind der Akku, die Recheneinheit (ALU), die beiden Indexregister X und Y und der Stackpointer auf 16 Bit verbreitert worden. Hinzu tritt jetzt ein 16 Bit breites Direct Page Register, das die frühere 'Zero Page' mit ihren verkürzten und schnellen Befehlen überall hin zu legen gestattet. Daneben gibt es zwei 8 Bit breite Register, eines für die Nummer der Programm-Bank und eines für die der Daten-Bank (jeweils 0 bis 255). Die Maschine kann also ihr Programm aus einer anderen Bank beziehen als ihre Daten, an denen sie arbeitet. Wir finden nebenstehend ein Registermodell für den vollen 16 Bit-Modus.

Ehe wir auf Einzelheiten eingehen noch einige Überlegungen: Der Adreßraum der frühen Mikroprozessoren hat sich bald als zu eng erwiesen, ob sich das Betriebssystem nun in Festwertspeichern oder per Urlader im RAM befand (CP/M). Oft genug mußte dieser Adreßraum auch noch mit den Interfaces und dem Video-RAM geteilt werden. Für das Hinzuladen von Sprachen (bei ROM-orientierten Maschinen) und Dienstprogrammen zum Editieren und Prüfen ergaben sich immer wieder Engpässe und Überschneidungen. Anwenderprogramme mußten sich in diese Umgebung einpassen. Man behalf sich oft mit umschaltbaren ROM-Bänken oder der Umschaltung von ROM auf RAM. Es waren alles Kompromisse, die aus der Not entstanden oder weil preiswert.

Die Erleichterung kam erst mit den Maschinen, die einen erweiterten Adreßraum haben. Bei den mit '6' beginnenden Prozessortypen ist z.B. der 6509 von Commodore ein Übergang. Durch ein spezielles Register in Adresse 0 (Execution Register) für die Programmausführung und in Zelle 1 (Indirection Register) für Daten ist er in der Lage, 4 weitere Adreßbussignale auszusenden, so daß insgesamt 16 Programm- oder Speicherbänke zu 64 KB erreicht werden können. Computer mit dieser CPU haben allerdings noch wenig Verbreitung gefunden, obwohl Speicher in der Größe von 1 Megabyte für viele Bequemlichkeiten, die man haben möchte, heute eigentlich schon ein 'Muß' sind, wenn man sich über professionelles Computing, auch als Nur-Anwender, unterhalten will. Hingewiesen sei auch auf die Beiträge von Andreas Goppolt in diesem und im letzten Heft zur interaktiven Programmierung und zum Trend bei den Programmiersystemen. Die Unterstützungen,

die ein modernes System bieten muß, können nicht in der Enge des Speicherraumes geboten werden, auch nicht durch die Krücke, daß man die Hilfen zeitaufwendig von der Diskette holen und zur Anzeige bringen kann.

Der 68000 von Motorola, demnächst auch mit 20 MHz Takt und 32 Adreßbusbeinen, Coprozessor usw. ist wohl im Moment das modernste Design, um mit den Adressen 'gerade davor' den notwendigen Speicherraum zu schaffen. Nicht ohne Grund dürfte sich APPLE mit der LISA und dem vielgefragten MacIntosh für 68000 entschieden haben, wobei der Autor zu bedenken gibt, daß bei letzterem das Design von Anfang an wohl zu schmalbrüstig hinsichtlich des möglichen Betriebes großer Speicher geraten ist. Auch wird hier ein standardmäßiges zweites Diskettenlaufwerk vermißt. Professionelle Datenverarbeitung kann man schließlich nur betreiben, wenn man auf bequemen Wegen von einer Diskette eine Sicherheitskopie ziehen kann.

Der hier besprochene 65816 bietet, wie wir noch sehen werden, einen enormen Fortschritt für 65xx und seine vielen Anhänger. Er ist zwar nicht so 'gerade davor' wie der 68000, denn ein Teil der Adressenverwaltung steht in der Verantwortung des Programmierers, indem er vor allem das Data Bank Register verwaltet, dessen Inhalt als hohe Adresse der Bank ausgesandt wird. Beim Program Bank Register ist das wohl nicht ganz so kritisch, es wird durch 'lange' Sprung- und entsprechende Subroutine-Anweisungen gesetzt. Auch haben wir nicht Befehle zur Multiplikation und Division wie beim 68000, dafür aber wie gehabt den dezimalen Modus. Insgesamt werden wir also, bei einigen Wünschen, die offengeblieben sind, uns mit den neuen Möglichkeiten befassen (wohl gewinnbringend) müssen. Der Autor sieht für die CPU vor allem einen lohnenden Einsatz in Entwicklungssystemen in mehrsprachlichen Systemen und in der immer wichtiger werdenden Textverarbeitung voraus, wobei möglicherweise andere wichtige Bereiche noch nicht bedacht wurden. Man muß sich immerhin vor Augen halten, daß aus inzwischen Millionen von 65xx-Betreibern an Heimcomputern in aller Welt sich viele als Assembler- und Anwendungsprogrammierer mit einer Vielfalt von Ideen rekrutieren. Die Zeit arbeitet also für den Betreiber.

Die neuen Befehle und Adressierungsarten

Gemeinsamkeiten mit dem erweiterten Befehlssatz der Rockwell-CPU R65C02

Die hexadezimale Matrix der Opcodes für Befehle wies bei 6502 große Löcher auf. Sie wurden bei der R65C02 mit x7 bzw. xF vor allem mit Bit-orientierten Befehlen aufgefüllt. Das ist eine Ausschlußmenge, die nicht auf der 65816 ist. Gemeinsam sind:

ORA AND EOR ADC STA LDA CMP und SBC mit Opcode x2 und der Schreibweise z.B. EOR (Pointer). Es handelt sich hier um eine indirekte Adressierung, für die man das Y-Register nicht zusätzlich auf '0' bringen muß. Der Pointer in der Zero Page ist die effektive Adresse.

TSB und TRB - Teste und Setze/Lösche Bit des Akkumulators in der Zero Page und in absoluter Adressierung mit den Opcodes 04, 0C sowie 14 und 1C

STZ Store Zero in der Zero Page oder absolut, auch indiziert. Nullsetzen einer Zelle.

INC A DEC A Inkrementieren und Decrementieren des Akkus A.

PHY PHX PLY PLX Stacken und Entstacken der Indexregister auf direktem Wege, ohne den Umweg über den Akku.

BIT immediate mit Opcode 89, Direktoperand

JMP (absolut,X) Sprung über eine Sprungleiste, die mit Index X adressiert ist.

BRA Branch always, kurze unbedingte Sprünge.

Absolut 'lange' Befehle

ORA AND EOR ADC STA LDA CMP und SBC. Diese Befehlsgruppe 1 stand auch bisher mit den meisten Adressierungsarten zur Verfügung. Es tritt eine Erweiterung auf lange Befehle ein, mit denen der gesamte Speicherraum von 16 MB übertrichen wird. Das 4. Byte des Befehls enthält dann die Bankadresse. Wir haben mit xF gebildete Opcodes. Befehle dieser Art sind nicht nur mit der effektiven Adresse im Befehl möglich (also unindiziert), sondern auch mit Indizierung durch X.

MSD \ LSD	0	2	3	4	7	A	B	C	E	F
0		COP s	ORA sr	TSB d	ORA (dl)		PHD s	TSB a		ORA al
1		ORA (d)	ORA (sr),Y	TRB d	ORA (dl),Y	INC acc	TCS imp	TRB a		ORA al,X
2		JSL al	AND sr		AND (dl)		PLD s			AND al
3		AND (d)	AND (sr),Y	BIT d,X	AND (dl),Y	DEC acc	TSC imp	BIT a,X		AND al,X
4		WDM res	EOR sr	MVP xyc	EOR (dl)		PHK s			EOR al
5		EOR (d)	EOR (sr),Y	MVN xyc	EOR (dl),Y	PHY s	TCD imp	JMP al		EOR al',X
6		PER s	ADC sr	STZ d	ADC (dl)		RTL s			ADC al
7		ADC (d)	ADC (sr),Y	STZ d,X	ADC (dl),Y	PLY s	TCD imp	JMP (a,X)		ADC al,X
8	BRA r	BRL rl	STA sr		STA (dl)		PHB s			STA al
9		STA (d)	STA (sr),Y		STA (dl),Y		TXY imp	STZ a	STZ a,X	STA al,X
A			LDA sr		LDA (dl)		PLB s			LDA al
B		LDA (d)	LDA (sr),Y		LDA (dl),Y		TYX imp			LDA al,X
C		REP imm	CMP sr		CMP (dl)		WAI imp			CMP al
D		CMP (d)	CMP (sr),Y	PEI s	CMP (dl),Y	PHX s	STP imp	JML (a)		CMP al,X
E		SEP imm	SBC sr	LPX d	SBC (dl)		XBA imp			SBC al
F	SBC (d)	SBC (sr),y	PEA s		SBC (dl),Y	PLX s	XCE imp	JSR (a,X)		SBC al,X

Die bei 65816 gegen 6502 neuen Befehle und Opcodes in verkürzter Matrix

s = Stack sr = stack relative, Operand auf dem Stack d = direct page
(sr),Y = Pointeradresse auf dem Stack (relativ), Indizierung mit Y
(dl) = Pointer in der direct page mit Bankangabe
(dl),Y = Pointer in der directpage mit Bankangabe, Nach-Indizierung mit Y
a = abolut al = absolut mit Bankangabe al,X = absolut mit Bankangabe und Index X
imm = immediate imp = implied acc = Akku d,X = direct page, indiziert mit X
(a) = absolut indirekt (a,X) = absolut indirekt, indiziert mit X
xyc = VON-Adresse in X, NACH-Adresse in Y, Count im Akku C
r = relativ, kurzer Offset rl = relativ, Offset in 2 Byte

JMP und JSL. Mit Opcode 5C haben wir den 'langen' Sprung und mit 22 die lange Verzweigung zum Unterprogramm. Zu letzterem Befehl gehört der Rückkehrbefehl RTL mit Opcode 6B, der das Programm in der aufrufenden Bank fortsetzt.

JML Mit Opcode DC ist dieses der lange indirekte Sprung. Das 4. Byte im Befehl enthält die zu verwendende Bankadresse.

Direct Page-Befehle

Für die Befehle der Gruppe 1, nämlich ORA, AND, EOR, ADC, STA, LDA, CMP und SBC haben wir neue Adressierungsarten, deren englische Bezeichnungen nicht so ganz hilfreich sind. Es geht

um indirekte Adressierungen, wie wir sie bisher typisch schon kannten mit (POINTER),Y. Ein Pointer liegt in der Zero Page, er zeigt auf eine Adresse. Nun gibt es beim 65816 ja das Direct Page Register. Sein Inhalt wird zum Byte hinzuaddiert, das dem Opcode folgt. Nun ist das Direct Page Register 2 Byte breit. Das heißt, das die Direct Page (früher Zero Page) überall im RAM liegen kann, auch seitenüberschreitend. Die wie üblich kürzeste Ausführungszeit wird erreicht, wenn dieses Register '00' enthält, eine Kurzwegtechnik macht dann die Addition entbehrlich. Die Opcodes mit 12, 32, ... F2 beschaffen mit dieser Adressenbildung (Folgebyte plus Direct Page Register) effektive Adressen, ohne daß Y hinzuaddiert wird. Und dann gibt es hierzu auch eine lange Form, die in andere Bänke hineinzeigt, nämlich mit den Opcodes 07, 27, 4f usw., auch hier ohne Indizierung durch Y. Diese Indizierung wird mit den Opcodes 17, 37 usw. erzielt. Diese langen Formen haben, wenn das Datenblatt richtig beurteilt wird, trotzdem nur zwei Byte lange Befehle. Aus der Direct Page wird nämlich nicht nur der Pointer mit seinen 2 Bytes entnommen, sondern als drittes Byte auch die Banknummer.

In einigen Fällen treten also zusätzliche Zyklen bei den Befehlen für die Zero Page bzw. künftig 'Direct Page' ein. Das ist sicher nicht tragisch, denn das Gerangel um die bevorzugten Plätze in der Zero Page dürfte nun endgültig vorbei sein. Wie erinnerlich, haben alle 65xx Computer mit dem Betriebssystem und der Sprache die Zero Page bereits fast völlig beschlagnahmt. Für den Anwender und seine Routinen blieb da oft nichts nach. Nun können wir künftig beliebig viele Direct Pages haben. Sorgfältig entworfene Firmware wird künftig für jede Sprache, Dienstleistung und das Betriebssystem deutlich zugeordnete eigene Direct Pages belegen. Die Konzepte müssen neu überdacht werden, auch in Hinsicht auf jetzt mögliche Multi-Tasks und Multi-User, die bedient werden können, ohne daß man ganze Seiten und Stackinformationen swapt. Auf den Stack kommen wir noch zu sprechen.

Der Stack als Pointer auf Daten

Wie man dem Registermodell entnimmt, ist der Stackpointer nunmehr 2 Byte breit. Der Stack muß daher nach einem Reset per Programm initialisiert werden. Er kann dann aber an beliebiger Stelle in der Bank 0 liegen. Und man wird ihn benutzen können, um für rekursive Unterprogrammaufrufe jeweils geschützte Parameterbereiche zu schaffen. Diese Technik ist vor allem bei der Implementierung von Sprachen wichtig.

Nun kommen zwei schöne Adressierungsarten hinzu, die die Parameterübergabe an Unterprogramme über den Stack elegant machen, die stack-relativen Adressierungen, wiederum für die Befehle der Gruppe 1 mit ORA, AND usw. Alle Befehle sind 2 Byte lang, wobei das zweite Byte zum Inhalt des Stackpointers hinzuaddiert wird (es bildet einen Offset). In der einen Form dieser Befehle (Opcodes 03, 23 usw.) ist damit eine effektive Adresse gebildet. Man kann also Parameter, die auf dem Stack liegen, direkt bearbeiten. Die zweite Adressierungsart mit den Opcodes 13, 33 usw. faßt das Ergebnis aus Offset und Stackpointer als einen Pointer auf, der mit dem Register Y modifiziert wird. Es handelt sich also um eine indirekte Adressierung, bei der sich die effektive Adresse aus dem Parameter auf dem Stack plus Y ergibt. Man denke hier nur an Zeiger auf Tabellen usw. Weitere Befehle unterstützen die Parameterübergabe über den Stack (s.u.).

Die stack-relativen Adressierungen dürften sich künftig als sehr nützlich erweisen. Parameter wurden bisher entweder in Briefkästen, in den Registern oder auf dem Stack übergeben, auch auf künstlichen Software-Stacks. Hierzu waren immer umständliche Prozeduren nötig, die künftig entfallen können.

Stackbefehle und Registertransfer

PHD und PLD bringen bzw. holen das Direct Page Register auf den/vom Stack.

PHB und PLB vollziehen das Gleiche für das Data Bank Register.

PHK bringt das Register der Programmbank auf den Stack. Einen umgekehrten Weg gibt es nicht. Dieses Register wird jedoch durch Befehle gesetzt, wie JML, JSL und RTL, die lange Sprünge/Unterprogrammaufrufe und Rückkehr betreffen.

PEA Push Effective Adress. Ablage eines Adreßwortes (Direktoperand im Befehl) auf dem Stack. Es werden keine Register beeinträchtigt.

PEI Push Effective Indirect Address. Hierbei werden zwei Pointerbytes aus der Direct Page auf den Stack gebracht. Das zweite Byte im Befehl gibt den Offset zum Direct Page Register an, der beim Laden zu verwenden ist.

PER Push Effective Program Counter Relative Address. Es wird ein AdreßwortDatenwort aus dem zweiten und dritten Byte des PER-Befehles entnommen, zum Stande des Befehlszählers hinzuaddiert. Das Ergebnis wird auf den Stack gebracht. Hier ist eine Möglichkeit gegeben, mit verschieblichem Code zu arbeiten, insbesondere bei der Parameterübergabe an Unterprogramme.

TCD und TDC transferiert Akku C (B+A) in das Direct Page Register und umgekehrt.

TCS und TSC dito für das Stackregister (initialisieren, retten).

TXY und TYX transferiert Register X nach Y bzw. Y nach X.

XBA Austausch der Akkumultoren A und B.

Statusbeeinflussung

SCE Exchange Carry Bit C with Emulation Bit E. Das Bit E im Status ist für den Modus der CPU verantwortlich. E=0 heißt 65816-Modus, E=1 heißt 6502-Emulation. Das E-Bit ist nicht direkt erreichbar, sondern nur im Tausch mit dem C-Bit.

Außer mit den Befehlen wie CLC, SEC, SED usw. läßt sich der Status auch mit Direktoperanden beeinflussen, und zwar mir REP und SEP (lösche/setze Bits). Diese Befehle wird man benutzen, um die neuen Bits M und X im Status zu beeinflussen. M ist dabei für die Steuerung des Akkus und X für die des X-Registers zuständig, ob in 8 oder in 16 Bit Breite (=0) arbeitend. - Der Befehl LPX mit Opcode E4 ist noch nicht dokumentiert.

Verschiedene Befehle

Nun haben wir endlich auch Block-Move-Befehle, und zwar MVN und MVP. Ersterer faßt den Block am ersten Byte an und schreitet zu höheren Adressen weiter. Der andere macht es genau umgekehrt. Die Datenbank VON steht im ersten Byte nach dem Opcode, diejenige für NACH im zweiten. Die Adresse VON wird dem X-Register entnommen, die Adresse NACH dem Y-Register. Die Zählvorgabe (-1) wird dem Akku C entnommen. Wir haben damit elegante Möglichkeiten, z.B. ein Video-RAM zu erneuern.

Der kurze Verzweigungsbefehl BRA (Branch Always) wurde bereits erwähnt. Es gibt ihn auch in Langform als BRL mit Offset in 2 Bytes, womit eine ganze Bank überstrichen werden kann (relokativer Code). Für Entschedungen wurde bereits bei der R65C02 der indizierte Sprung JMP (Tabelle,X) als nützlich genannt. Der 65816 hat zusätzlich ein JSR (Tabelle,X), so daß man auch Unterprogramme in Abhängigkeit von Entscheidungen bequem anspringen kann.

Der Befehl COP soll die Zusammenarbeit mit einem Co-Prozessor ermöglichen. Der Befehl wartet offensichtlich auf einen negativen Puls am Pin ABORT und führt das Programm dann an einer Stelle fort, die durch den COP-Interruptvektor bestimmt wird. WAI wartet auf einen Interrupt NMI oder IRQ. STP hält das Weiterclocken der Maschine an, was nur durch ein Reset zum Fortlaufen gebracht werden kann. Gibt man nach dem Reset ein RTI, so wird mit der auf STP folgenden Instruktion fortgefahren. WDM ist ein NOP. Dieser Befehl wurde für spätere Erweiterungen reserviert, er steht als Denkmal für den Design Engineer William D. Mensch Jr.

Zusammenfassung

Wie wir gesehen haben, zeichnet sich der 65816 durch Software-Kompatibilitätmit dem bewährten 6502 aus. Er schafft einen Adreßraum von 16 MB und stellt in den Registern Program Bank Register sowie Data Bank Register die Möglichkeiten zur Verfügung, diesen Adreßraum in einer sehr durchlässigen, transparenten Weise anzusprechen. Es gibt Befehle von 2 Byte Länge, die diesen großen Adressenbereich unter Bankumschaltung zu erreichen gestatten. Die Direct Page kann überall in der Bank 0 eingerichtet werden, ebenso der Hardware-Stackpointer. Für die Parameterbearbeitung auf dem Stack und für die indirekte Adressierung über auf dem Stack liegende Pointeradressen sind wirksame Adressierungsmöglichkeiten geschaffen worden, auch für das Beschicken

des Stacks mit Parametern. Die Register Akkumulator, X und Y können in 16 Bit Breite benutzt werden, was nicht nur für Indizierungen nützlich ist, sondern auch für das Rechnen, zumal der Prozessor ja auch die dezimale Rechenart hat.

Hardware- und Software-Konzepte müssen für diesen Prozessor überdacht werden. Er ruft gewissermaßen nach der Implementierung von Sprachen, für die er wesentlich besser vorbereitet ist als seine Vorgänger. Vielsprachigkeit wird ihm nicht lange fremd bleiben. Man wird mehrere Sprachen, Assembler, Textbearbeitungssysteme usw. bequem auf der Maschine resident halten können.

ΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔOΔO

David Long, 3000 Hannover

Textverarbeitung mit WORDCRAFT

Ich brauche nicht, wie bei vielen anderen Computerzeitschriften üblich, einen halben Roman über Nebensächliches zu schreiben und auch keine Superlative anzuhäufen. Auch wenn ich versuche, sachlich zu bleiben, so werden sie bald verstehen, warum ich vom Textprogramm bzw. den Textprogrammen 'Wordcraft' so begeistert bin. Textprogramme im Plural deswegen, weil Wordcraft-Versionen inzwischen auf 13 Rechnertypen laufen, nämlich CBM 64, VC 20, 8032, 8296, 700, IBM-PC, Sirius, DEC Rainbow, Apricot, HP 150, Sperry PC, NCR Decision Mate 5 und Olympia People. Eine Version für den Commodore PC ist geplant.

Das System unterstützt Möglichkeiten, Textfiles zwischen diesen Rechnern zu tauschen: Mit Ausnahme des VC 20 sind Textfiles auf allen CBM-Systemen identisch. Man braucht den CBM 8032 oder den CBM 700 nur an ein 4040 Floppy-Laufwerk anzuschließen, um C-64 Textfiles lesen zu können. Das ist ideal für freiberuflich Tätige, die zu Hause arbeiten wollen. Zwischen CBM 8000/700 und 16-Bit-Rechnern kann man Textfiles über V24-Interface oder Modem mittels der im Programm eingebauten Kommunikations- und Codewandlungsroutinen, die auch Error-Recovery- und Block-Repeat-Funktionen beinhalten, übertragen, vorausgesetzt, daß Wordcraft auf beiden Rechnern installiert ist.

Es begann 1978 in England mit der ersten Version, Wordcraft 80 für die CBM 8032-Serie, das von P. Dowson geschrieben wurde, einem ex ICL-Manager. Obwohl leistungsfähig, enthielt Wordcraft damals noch mehrere Bugs. Ab 1979 vertrieb Commodore ein 'verdeutschtes' Wordcraft für 8032 und 8096. Zu dieser Zeit war Wordcraft in der BRD nur über Commodore zu beziehen. Etwa 1981 brachte Commodore dann ihr eigenes Textverarbeitungsprogramm, Comtext, auf den Markt. Comtext, das noch heute verkauft wird, hat eine identische Befehlsstruktur wie Wordcraft, unterstützt jedoch nur Commodore-Drucker. Überraschenderweise hat Comtext die gleichen Bugs wie das damalige Wordcraft, zB. Global Print funktioniert nicht (Drucken eines ganzen Dokumentes). Comtext war auch auf dem Commodore-Stand bei der Commodore-Fachmesse 1983 zusammen mit einem 'Dongle' (Programmschutzstecker, siehe unten) von der Firma Wordcraft zu sehen.

Inzwischen ist Wordcraft durch viele ergänzende Versionen gegangen. Durch Anpassungsprobleme mit jedem neuen Drucker, der auf den Markt kam, verärgert, entwickelten die Engländer ein tabellengesteuertes Druckertreibersystem. Versierte Händler konnten alle Aspekte der Druckersteuerung über eine Tabelle (PDF = Printer Definition File - Drucker Definitions-Datei) mittels einer Utility selber programmieren. Insbesondere kann man aus jedem Zeichen am Schirm ein oder mehrere beliebige Zeichen zum Drucker schicken. Programmierung etwa von Zeichenabstands- und Zeilenabstandseinstellung war gewiß nicht einfach zu meistern. (Das Einstellen seitens des Anwenders vom Programm aus ist dagegen sehr leicht.) Inzwischen ist es noch einfacher geworden: Das PDF ist in Form einer eigens dafür entwickelten Programmiersprache ausgelegt, die als Textfile mit Wordcraft editiert werden kann. Z.Zt. werden 30-40 Druckertypen unterstützt, und die Liste wird täglich erweitert. Davon sind ca. 20 PDF's im Standard-Lieferumfang enthalten. Die Engländer bieten einen kostenlosen PDF-Erstellungsdienst an. Man braucht den Drucker le-

diglich für eine Woche oder so nach England zu schicken. Somit kann man beispielsweise auf dem C-64 Fettschrift, Unterstreichen, rechnergesteuerte Proportionalschrift, obere und untere Indizes (H_2O) usw. auf Typenraddruckern wie Qume oder Ricoh erzeugen. Wordcraft bietet auch ca. 26 frei programmierbare Escape-Codes, die als grafische Zeichen auf dem Bildschirm erscheinen und z.B. für Sonderzeichen oder individuelle Druckersteuerungen benutzt werden können. Man kann vom Programm aus von einem Druckertyp zu einem anderen schalten, solange eine entsprechende PDF vorhanden ist. IEEE- und Seriell-IEEE-Drucker schließt man logischerweise am IEEE-Ausgang an. Mit einem entsprechend verdrahteten Kabel kann man Centronics-Drucker direkt am Userport anschließen. Ähnlich gehts auch mit RS232-Druckern, man braucht aber hier auch eine RS232-Cartridge zum Anpassen der Pegel. PDF's sind identisch für alle CBM Wordcraft-Versionen; bei den 16-Bit Versionen ist es ein wenig anders.

Jetzt zur Texteingabe auf Wordcraft-64. Das Programm ist bildschirmorientiert, d.h. die zur Druckersteuerung (z.B. Fettschrift) bzw. Textformatierung (z.B. Einrücken) benötigten Steuerzeichen sind im allgemeinen im Text versteckt. Somit ist das Layout auf dem Schirm meist identisch mit demjenigen auf dem Papier, mit Ausnahme des Randausgleiches und der Mikrojustage. Wo Steuerzeichen im Text versteckt sind, wird das entsprechende Textzeichen negativ dargestellt. Mittels CBM-c ('Control'-Taste und 'c') wird das Steuerzeichen dann im oberen Bildschirmbereich (Statuszeilen) angezeigt. Oft wird der Effekt solcher Steuerzeichen jedoch unmittelbar sichtbar, z.B. bei einem TAB. Auf dem IBM-PC kann Wordcraft sogar Unterstreichen auf dem Bildschirm darstellen. Wordcraft erlaubt alle üblichen Formatkommandos, z.B. Zentrieren, Einrücken, Tabulieren und dezimal Tabulieren, Trennpunktvorgabe usw. Dazu kommen verschiedene Funktionen wie Suchen und Austauschen, Textblock-Löschen, -Verschieben, -Wiederholen, Aneinanderfügen oder Ineinanderfügen (Mergen) von Dateien auf Floppy oder Bildschirm, automatisches Erstellen von Rundschreiben etc.

Zur Befehlsstruktur: Im Gegensatz zu einigen anderen Programmen mit einem gewaltigen Menübaum in vielen Ebenen sind fast alle Kommandos über die CBM-Taste und einem einzigen Buchstaben zu erreichen. Ein Wort löschen ist z.B. CBM-d (= Delete), neue Seite ist CBM-p (= Page). Wordcraft hat zwei Modi: Text-Modus zur Texteingabe und zum Editieren usw. und Command-Modus zum Drucken, Textabspeichern usw.. Man kommt von einem zum anderen über die STOP-Taste. Lediglich die 8 im Text-Mode meistbenutzten Kommandos sind den 8 Funktionstasten zugewiesen. Eine Tastaturschablone dafür ist im Lieferumfang des Programmes enthalten.

Der Seitenumbruch ist sofort sichtbar.Ein Text (engl. 'Document') wird in 'Kapitel' unterteilt, die jeweils insgesamt in den Arbeitsspeicher passen. Mit C-64 sind 23068 Zeichen für Text frei. Das entspricht etwa 15 Seiten DIN A4, 1 1/2-zeilig mit 10 Zeichen/Zoll.

Automatische Kopfzeilen mit beliebigem Text und durchgehende Zeilennumerierung ('header', h -RETURN- im Command-Modus) werden mit Funktionen wie 'Save and Insert' (neue Kapitel zwischen anderen Kapiteln abspeichern) unterstützt. Man kann auf ein zweites Floppy Disk-Laufwerk erweitern, falls der Text nicht auf eine Diskette paßt (Kapazität über 100 Seiten pro Scheibe mit 4040/1541).

Obwohl der C-64 nur 25 Zeilen zu 40 Zeichen darstellen kann, ist Wordcraft in der Lage, mittels 'Horizontal Scroll' texte mit bis zu 117 Zeichen pro Zeile bzw. über 'Vertical Scroll' 98 Zeilen pro Seite zu bearbeiten. Ränder und Tabstellen können in jeder Zeile neu gesetzt werden. Wordcraft unterstützt bis zu 8 Einzel- oder Doppellaufwerke, letzteres über eine passende IEEE-Karte (z.B. von der Fa. Dams in England). Doppellaufwerke haben den Vorteil, daß Backups leichter durchführbar sind. Wordcraft-8296 unterstützt sogar zwei Multi-User-Systeme (Hydra und TMS) sowie Drucker-Spooling. Zur Programmsicherung wird Wordcraft mit einem vergossenen Schutzstecker ('dongle') geliefert.

Welche Nachteile hat Wordcraft? Außer Wordcraft für den C-64 (354 DM) ist Wordcraft kein billiges Programm. Außerdem bestehen z.Zt. Lieferengpässe, insbesondere für 16-Bit Versionen. Zudem ist das Benutzerhandbuch derzeit nur in Englisch erhältlich. Allerdings sollen die im Programm eingebauten 'Help Files' auch bald in Deutsch zur Verfügung stehen. Allen Messeversprechungen Commodore's zum Trotz sind für deren Rechner Umlaute bisher noch nicht verfügbar.

Sobald diese zur Verfügung stehen, ist die Umstellung durch Abänderung der Printerfiles jedoch sehr einfach durchzuführen. Dabei ist es gleichgültig, welche ASCII-Codes den Umlauten seitens des Benutzers zugewiesen sind.

Zukunftsaussichten? Umlaute kommen bald, aber nicht von Commodore! Ein Bastler entwickelte jetzt eine Schnittstelle zum Photosatz. Ein Liefertermin wurde hierfür allerdings noch nicht genannt. In England läuft Wordcraft-PC mit online Telex im Hintergrund. An anderer Stelle wurde auch Datex-P erwähnt. ...

Bücher

Winter, H. D.: Analyse des Commodore-BASIC 4.0 und BASIC 3.0. Mikro + Kleincomputer Verlag, Bern 1982, 224 S. ISBN 3-907007-01-8. Erst jetzt wurde der Rezensent auf dieses nützliche Buch für den maschinensprachlichen Programmierer aufmerksam. Es hat folgende Hauptkapitel: Allgemeine Aufgaben eines Betriebssystems, das Betriebssystem des CBM-Rechners, Speicherverwaltung, Ein/Ausgabe, Mathematik, BASIC-Interpretation, Erweiterung des Betriebssystems sowie etwa 45 Seiten Anhänge u.a. mit den wichtigen Parameternder Rechner. Das Buch zeigt in einer gut lesbaren Art, wie man vor allem die Dienste des Betriebssystem in der maschinensprachlichen Programmierung benutzen kann. Es zeigt, welche Parameter dann an welchen Stellen übergeben werden müssen. Es sind zahlreiche Programmierbeispiele enthalten, die den besonderen Wert des Buches ausmachen. Seine Anregungen wird man auch vergleichend bei anderen CBM-Modellen verwerten können, denn es werden ja überall etwa die gleichen Grundfunktionen vorgehalten.

Lewis, T. G.: Der perfekte Führer zu Ihrem IBM-Computer. IDEA-Verlag, Puchheim 1984, ca. 256 S., ISBN 3-88793-016-9, DM 32,--. Das Buch ist aus dem Amerikanischen übersetzt worden. Es zeigt in einer allgemeinverständlichen Form mit ca. 40 Abbildungen und zahlreichen Beispielen, wie man den Rechner hantiert, um auf diesem Wege Betriebssicherheit zu vermitteln. Der Stoff ist von unnötigem Ballast freigehalten worden. Nach zwei einleitenden Kapiteln finden wir: Wie man den Computer startet, wie man Text verarbeitet (EasyWriter), wie man Tabellenkalkulationen erstellt (VisiCalc), wie man eigene Programme schreibt und schließlich hat es eine Einführung in die Datenverwaltung. Da die Hantierung eines Rechners oft umständlicher ist als seine Programmierung, mag dieses Buch damit für den Anfänger hilfreich sein. In Ermangelung eines PC's wurde das hier nicht getestet.

Creekmore, W.: Mikro-Wissen griffbereit. Ashton Tate und Vieweg-Verlage, Frankfurt 1984, 64 S. ISBN 3-528 04317-2, DM 24,80. Das amerikanische Softwarehaus und der deutsche Verlag haben vor einiger Zeit eine Partnerschaft vereinbart. Hier liegt nun die erste Veröffentlichung vor. Sie ist in ihrer Grafik außerordentlich aufwendig gestaltet. Inhaltlich werden die allereinfachsten Dinge der Computerei schematisch erklärt. Es ist damit kein Buch zum Einarbeiten, sondern eher eine Fibel für solche Mitarbeiter, die in Ihrem Betrieb plötzlich vor einem Computer oder Terminal sitzen sollen. Ihnen kann man vielleicht etwas die Angst vor dem Unbekannten nehmen.

Editorial

Dieses Heft enthält an erster Stelle wiederum einen nachdenklichen Beitrag von Andreas Goppold zu Trends in der Entwicklung von Programmiersystemen. Der Herausgeber glaubt, daß wir von Zeit zu Zeit unsere Schritte zum Zwecke einer solchen Besinnung anhalten müssen, um den weiteren Weg zu bedenken. Aus den Zeitschriften gewinnen wir den Eindruck, daß der Einsatz und die Verwendung des Microcomputers rasch fortschreitet. In mengenmäßiger Hinsicht dürfte der Eindruck angesichts vieler hunderttausend abgesetzter Home-Computer stimmen, auch angesichts der zunehmenden Verwendung der Mikros für Steuerungen der verschiedendsten Art. In qualitativer Hinsicht wird man aber wohl noch immer Einschränkungen machen müssen. Der Personal Computer ist noch wenig in die vielen mittelständischen Betriebe eingedrungen, und in den großen gibt es oft die Barriere der zentralen EDV-Abteilung, die initiativen Abteilungsleitern den Weg zur eigenen kleinen Problemlösung 'off line' verlegt. Unter dem Gesichtspunkt 'einheitlicher' Lö-

sungen in einem Unternehmen mag das in mancher Weise berechtigt sein, nicht aber unter dem Gesichtspunkt des Fortschrittes, denn ein Fachbereich kennt seine Aufgabenstellung am besten, und es ist nicht falsch, wenn man sich dort Gedanken macht, wie man die Technik der Zeit gewinnbringend benutzen kann. Und es braucht seine Zeit, um ausreichend Erfahrungen sammeln zu können. Man sollte sie nutzen, um nicht immer wieder von Fortschrittslücken zu hören.

Bei allem Optimismus, den man aus den Berichten über den technischen Fortschritt auf der Seite der Hardware vielleicht gewinnt (schnellere Taktzeiten, größere Speicher usw.) sollte man besonders den Faktor Zeit nicht aus den Augen lassen. Sicher, die Leistungsfähigkeit nimmt schnell zu, nicht jedoch ihre Ausnutzung und die Ausstattung der Hardware mit der notwendigen Software und Dokumentation. Gute Dinge brauchen nach wie vor viel Zeit bis zur Reife. Ab Herbst 1976 wurde der KIM-1 in größerer Zahl ausgeliefert. Zwei Jahre später folgten der AIM 65 und der PET 2001 von Commodore, wie auch der TRS 80. Etwas früher war wohl der APPLE am Markt. Wenn der Herausgeber richtig erinnert, gab es auf der ELECTRONICA im November 1978 bereits ein vorläufiges Datenblatt für den 68000 von Motorola. Sicher ist, daß er im Dezember 1979 in Unterföhring am ersten 68000-Lehrgang teilnahm. Damals hatte es weniger als zehn Teilnehmer. Schauen wir auf den KIM. Der Markt hat ihn schnell vergessen, nachdem es Systeme mit einer bequemeren Eingabe für Programme gab. Für den AIM 65 hat es im Grunde genommen Jahre gedauert, bis für ihn größere Anwenderprogramme zur Verfügung standen. Die gleiche Aussage gilt für den PET und seine zahlreichen Modelle. Ein gewisser Höhepunkt wurde erst mit den Programmen für den 8032 erreicht. Für die Modelle der Serie 600 und 700 muß man feststellen, daß sie wegen lange bestehender Unsicherheiten und wegen zu geringer Dokumentation und Unterstützung durch den Hersteller wenig Akzeptanz gefunden haben, obwohl sie in ihrer Leistungsklasse eine Konkurrenz für den IBM-PC hätten bilden können. Bei der Abfassung dieser Zeilen kommt dem Herausgeber eine Gratis-Sonderausgabe der Zeitschrift BYTE auf den Tisch mit dem Titel 'Special IBM Issue'. Dabei mag man über die Tatsache nachdenken wie man will, warum man z.B. ein 300-seitiges Heft mit wenig Reklame außerhalb des Abonnements erhält. IBM kam spät hinzu, fand jedoch vor allem in den USA einen offenen Markt und die Unterstützung durch viele Lieferanten von Nachbauten und Ergänzungskarten sowie durch die Softwarehäuser. Die Architektur des Rechners mag sehr konventionell sein (um nicht ein anderes Wort zu gebrauchen). Wir stehen jedoch vor der Tatsache, daß auch eine konventionelle Technik Jahre braucht, um zu einem großen Erfolg zu führen.

Wenn wir einmal von Steuerungen absehen, über die man sicher zu wenig weiß, so ist zum 68000 zu sagen, daß er eine massenhafte Verbreitung erst mit der LISA und vor allem mit dem MacIntosh gefunden hat, und zwar in der Form anwenderfreundlicher Systeme, in denen nicht die Programmierung im Vordergrund steht, sondern die Benutzung. Der 68000 und der 68020 haben eine hervorragende Architektur, es braucht aber auch hier Jahre, bis sie mit nützlichen Programmen bekleidet wird. In mancher Weise wird das auch auf den 65816 zutreffen, der in diesem Heft vorgestellt wird. Es braucht seine Zeit, bis gute Ideen in ihn investiert worden sind.

An der Leistungsfähigkeit der Hardware wird also weiterentwickelt. Heute schon sind Megabyte von Adreßraum erreichbar. Benutzt werden sie aber noch viel zu wenig. Immerhin sind jetzt die Möglichkeiten gegeben, sowohl für den Anwender wie auch den Programmierer die benutzerfreundlichen Interfaces zu bieten. Alle Arten von Hilfs- und Auskunftsfunktionen können resident auf den Computer gelegt werden, so daß sie auf Knopfdruck zur Verfügung stehen. Und es sind auch Fortschritte in den Programmiersprachen denkbar und vonnöten, um zu einer interaktiven Programmierung zu kommen, die die Produktivität erhöht und die die großen und oft nicht mehr überschaubaren Komplexe in Module auflöst.

Auf diesem Wege kann die Sprache FORTH Hilfen bieten, ein wichtiger Grund, um auf ihre Möglichkeiten in dieser Zeitschrift hinzuweisen. Natürlich spielt auch hier der Zeitfaktor hinein. Die Sprache ist hier noch gar nicht solange verbreitet. Vielleicht ist es auch nicht FORTH in der jetzigen Version, was die interaktive Programmierung befördern wird. Möglicherweise sind es die hier verarbeiteten Grundgedanken, nämlich Erweiterbarkeit, Programmieren in überschaubaren Modulen, die sofort compiliert und erprobt werden können, und die Möglichkeit, sogar den Compiler zu verändern. Man kann sich vorstellen, daß man auf diese Grundeigenschaften noch Attribute aufstocken kann, die die innere Transparenz und die Interaktivität erhöhen.

Nostalgie

Beim Räumen fiel dem Herausgeber zufällig wieder ein Prospekt der Zuse KG aus dem März 1961 in die Hand. Die nachfolgenden Merkmale mögen für Vergleiche vielleicht ganz interessant sein:

1.2 Vergleich zwischen ZUSE Z 22 und ZUSE Z 23

Erläuterungen	ZUSE Z 22	ZUSE Z 23
Bauelemente	450 Röhren 2300 Dioden	2400 Transistoren 6700 Dioden
Schnellspeicherkapazität (einschließlich Rechenregister)		
Grundausführung	14 Worte	246 Worte
mit Erweiterung	25 Worte	— —

Erläuterungen		ZUSE Z 22	ZUSE Z 23
Operationszeiten			
Elementaroperationen (Transportbefehle, Addition im festen Komma, logische Operationen, Entscheidungen, Verschiebungen usw.)		0,6 msec	0,3 msec
Addition	im gleitenden Komma	32,5-52,5 msec	10,6 msec
Subtraktion	im gleitenden Komma	34 - 54 msec	12 msec
Multiplikation	im gleitenden Komma	30 msec (Z22R)	20 msec
Division	im gleitenden Komma	70 msec	20 msec
Ein- und Ausgabezeiten im gleitenden Komma			
Streifeneingabe			
Grundausführung		15 Zeichen/sec	200 Zeichen/sec
mit Erweiterung		100 Zeichen/sec	— —
Streifenausgabe			
Grundausführung		10 Zeichen/sec	10 Zeichen/sec
mit Erweiterung		20 Zeichen/sec	50 Zeichen/sec
Fernschreiberausgabe		10 Zeichen/sec	10 Zeichen/sec
Zeilendruckerausgabe		50 Zeichen/sec	80 Zeichen/sec
Zahlendarstellung			
Wortlänge		38 Bits	40 Bits
Gleitkommamantisse		29 Bits=knapp 9 Dezimalst.	30 Bits= 9 Dezimalst.
Gleitkommaexponent		7 Bits	8 Bits
Zahlenbereich		10^{-20} - 10^{+19}	10^{-39} - 10^{+38}

Kleinanzeigen

Professionelle Textverarbeitung für CBM-64? WORDCRAFT ist jetzt verfügbar für CBM 64. Textfiles identisch mit denen vom Wordcraft 8032/710 usw. Alle Druckersteuerfunktionen, ASCII-Codes usw. über Drucker-Datei frei definierbar. Photosatzanschluß. WC 64 mit Anleitung usw. DM 354. **Neue Modems aus England:** Buzzbox: 300 Bd Vollduplex (V.21) DM 399. M-1000: 1200 halbduplex und 1200/75: DM 399. M-2000 wie M-1000, jedoch software-umschaltbar DM 445. Nur am Netz der UK Post zugelassen. Postkarte an Buglass + Long, Heinrichstraße 24, 3000 Hannover, Tel. 0511-31 73 75.

Assembler

R65C02-Assembler

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

6805/68705 Cross-Assembler

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

6805/68705-Assembler unter FORTH

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

Mathe-ROM für 6502

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

Texterfassung und Lichtsatz

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

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

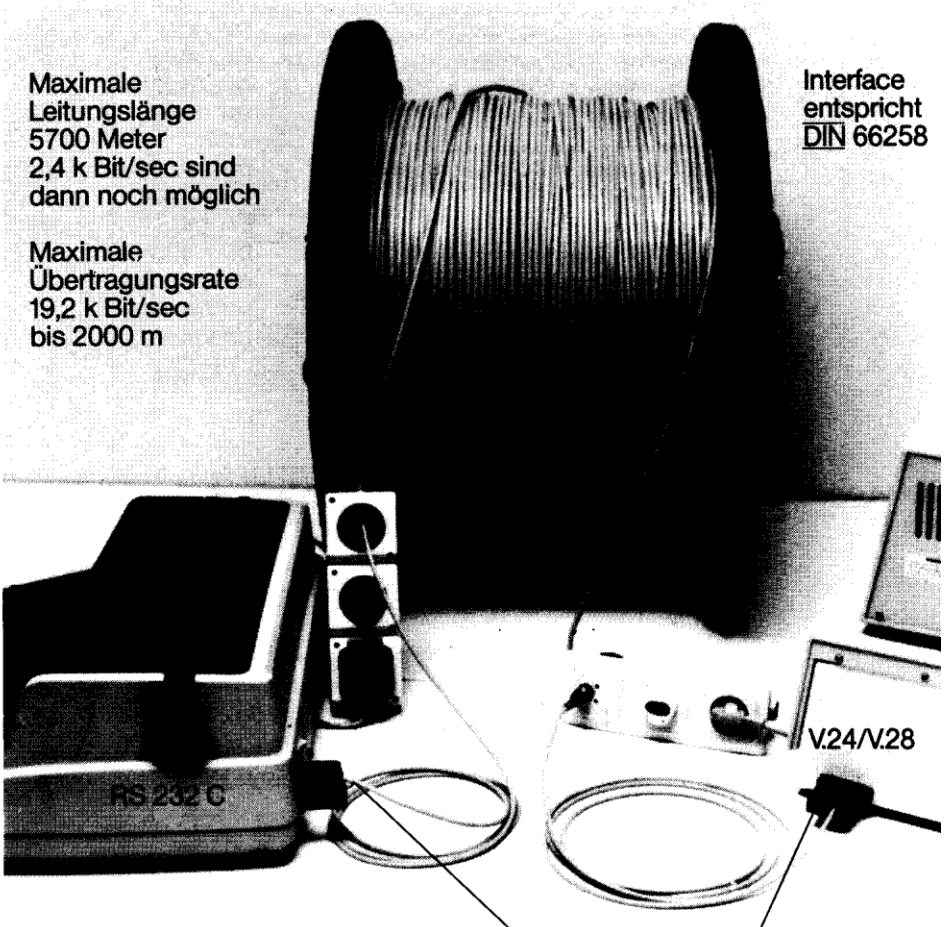
Roland Löhr, Hansdorfer Str. 4, D-2070 Ahrensburg
Tel.: 04 102 - 55 816

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

Maximale Leitungslänge 5700 Meter 2,4 k Bit/sec sind dann noch möglich

Maximale Übertragungsrate 19,2 k Bit/sec bis 2000 m

Interface entspricht DIN 66258



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

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

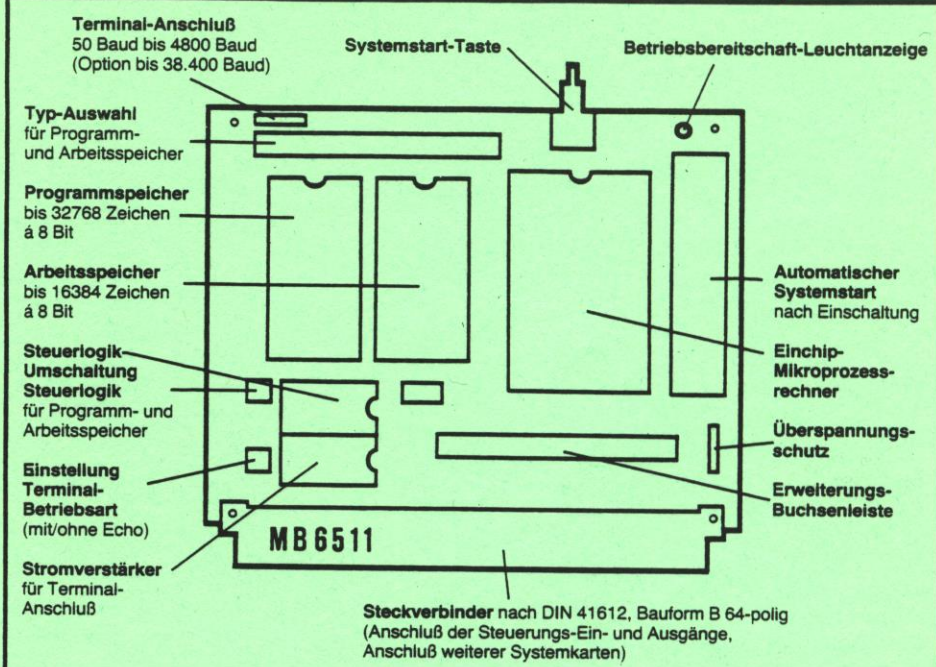
DBGM: G 8331 081
G 8336 080

INGENIEURBÜRO
STECKER

5000 Köln 60 (Nie
Postfach 60 07 66
Delmenhorster Str.
Tel. (02 21) 712 401

8408

MINIBEE-MB6511



TECHNISCHE DATEN

Leiterplattengröße	100mm x 80mm
Gesamtmaße	100mm x 88mm x 14mm
Kartenmaterial	Glasfaser-Epoxy 1,5mm, beidseitig Lötstoplack
Gewicht	80g
Leistungsbedarf	2W
Spannungsversorgung	5V ± 5%
Systemtakt	2MHz (Option 1MHz)
Arbeitstemperaturbereich	0°C bis +70°C
Programmspeicher	2kByte bis 32kByte BYTEWIDE
Arbeitsspeicher	2kByte bis 16kByte BYTEWIDE
extern erweiterbar bis	4MByte (256 x 16kByte)
Terminal-Anschluß	TTL-kompatibel

RECHNER DATEN

Einchip-Mikroprozessor	R6511Q (Rockwell/NCR)
Rechnerstruktur	6500
Programmierung	softwarekompatibel zu 6500 60 Befehle, 14 Adressierungsarten
Ein-/Ausgabe-Anschlüsse	24 O.C. mit Pullup, 8 Tristate, alle TTL kompatibel
Zähler/Zeitmesser/Takt	1 voll programmierbar, retriggerbar 1 für serielle Schnittstelle
Serielle Schnittstelle	voll duplex, asynchron, synchron
Programm-Unterbrechung (IRQ)	10 (je 2 positiv, negativ, Zähler, Serielle Schnittstelle; 1 unbedingte Unterbrechung (NMI)

EINCHIP-MIKROPROZESSOR-STEUERUNG

mit R6511Q

DM 350,– + Mwst

BRÜHL ELEKTRONIK ENTWICKLUNGS-GESELLSCHAFT mbH

8500 Nürnberg 10, Hegelstr.10 TEL. 0911-359088

MICRO MAG

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

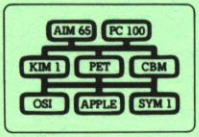
MICRO MAG (vormals 65xx MICRO MAG) erscheint zweimonatlich, jeweils Mitte Februar, April usw.. Offsetdruck: L & L Druckservice, Hamburg 73

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

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

Leser-Service des Herausgebers

Das Buch 7-13 des 65.. MICRO MAG



340 Seiten, DM 42,--

Nachliefermöglichkeiten:

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

Lieferbar: 'Das Buch 7-13 des 65xx MICRO MAG' zu DM 42,-- sowie die Hefte 14-37 zu DM 7,80/St. (ab 10 St. in einer Sendung: DM 6,--/St.).

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

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

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

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