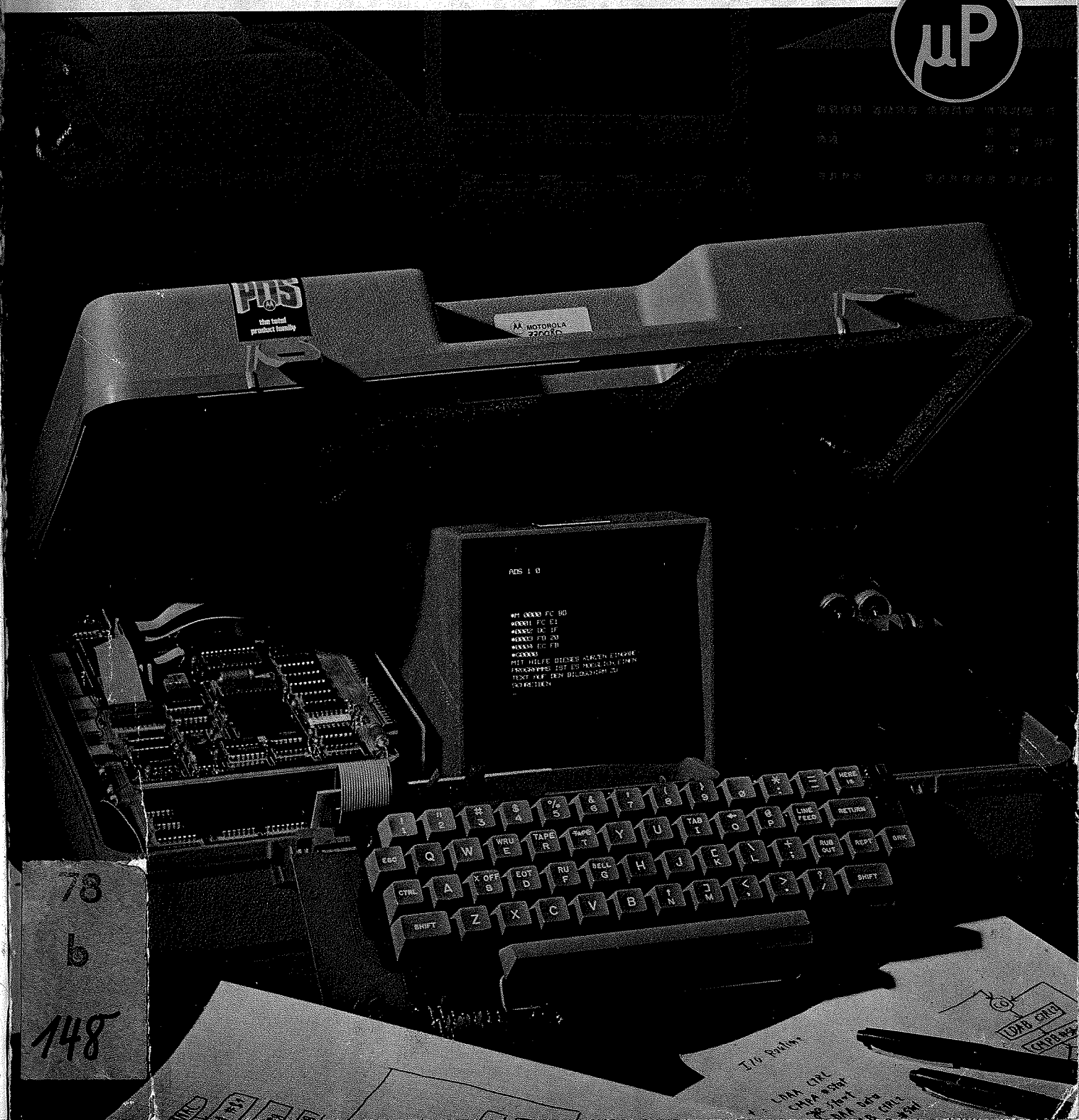
Elektronik

Sonderheft II

Preis DM 16.–

Mikro prozessoren

software



8085 - der einzigartige Intel - Mikrocomputer

Der 8085-Mikrocomputer und alle anderen Bauelemente des Intel-Mikrocomputer-Systems MCS-85 sind jetzt lieferbar.

Der 8085 ist Software- und Buskompatibel mit dem Industrie-Standard 8080A. Damit stehen Ihnen die ausgereifte Software, Peripheriebausteine und Entwicklungssysteme zur Verfügung, die den Erfolg des 8080A-Systems begründet haben.

Was unterscheidet den 8085-Mikrocomputer vom bekannten 8080A?

Der 8085 ist um 50% schneller. Alle Bausteine des MCS-85 Systems sind höher integriert: Ein 10-Chip-MCS-80-System kann durch drei MCS-85-Bausteine ersetzt werden. Alle Bausteine der MCS-85-Familie benötigen nur eine 5-Volt-Versorgungsspannung.
Das 3-Chip-MCS-85-System besteht aus:
8085 CPU
8155 RAM mit 256 Byte, E/A und Timer
8755 EPROM mit 16K Byte und E/A
(kompatibel zum 8355 ROM mit E/A)

Was bedeutet das für bisherige 8080A-Anwender und für Entwickler, die jetzt mit dem 8085 beginnen?

Da der 8085 voll kompatibel mit der 8080A-Software und den Peripherie-Bausteinen der Serie 8200 ist, behalten bereits gemachte Investitionen und Erfahrungen ihren Wert. Sie können auf Neuentwicklungen übergehen, ohne wieder von vorne anfangen zu müssen. Gleichzeitig können in Ihren MCS-80 und MCS-85-Systemen die neuen intelligenten Peripherie-Steuereinheiten einsetzen: Den Floppy-Disk-Controler 8271, den synchronen Data-Link-Controler 8273, den CRT-Controler 8275 und den Keyboard/Display-Baustein 827[illegible] Diese neuen Bausteine sind voll programmierbare Ein-Chip-Lösungen für System-Schnittstellen.

Sie können wie bisher Intels höhere Programmiersprache PL/M verwenden, die Ihre Software-Entwicklung um Monate verkürzen kann. Das universelle INTELLEC®-Mikrocomputer-Entwicklungs-System mit dem ICE-85-„In-Circuit"-Emulator und symbolischen Test- und Fehleranalyse-Möglichkeiten reduzieren den Aufwand in der Test- und System-Integrationsphase. Unsere Schulungskurse und Seminare ermöglichen es Ihnen mit geringem Zeitaufwand die Vorteile der MCS-85-Familie kennen zu lernen und zu nutzen. Eine umfassende Software-Bibliothek steht Ihnen ebenfalls zur Verfügung.

Ihre Vorteile bei Einsatz des MCS-85-Mikrocomputer-Systems sind bei erheblich geringeren Kosten, eine schnellere und leistungsfähigere System-Entwicklung mit vertrauten Hilfsmitteln.

Den Entwicklungsvorgang, den Sie bisher mit dem System 8080A und seinen Entwicklungshilfen hatten, können Sie nun mit dem 8085 entscheidend ausbauen.

Fordern Sie das ausführliche MCS-85-Handbuch* an oder vereinbaren Sie einen Termin mit Ihrem Intel-Vertriebsingenieu
* Schutzgebühr DM 15,–

intel®

INTEL SEMICONDUCTOR GMBH
8000 MÜNCHEN 2, SEIDLSTRASSE 27
Telefon: 089/55 81 41 Telex: 5-23 177

mikroprozessoren
microcomputer
speichersysteme
rams·roms·proms

intel liefert unterstützt

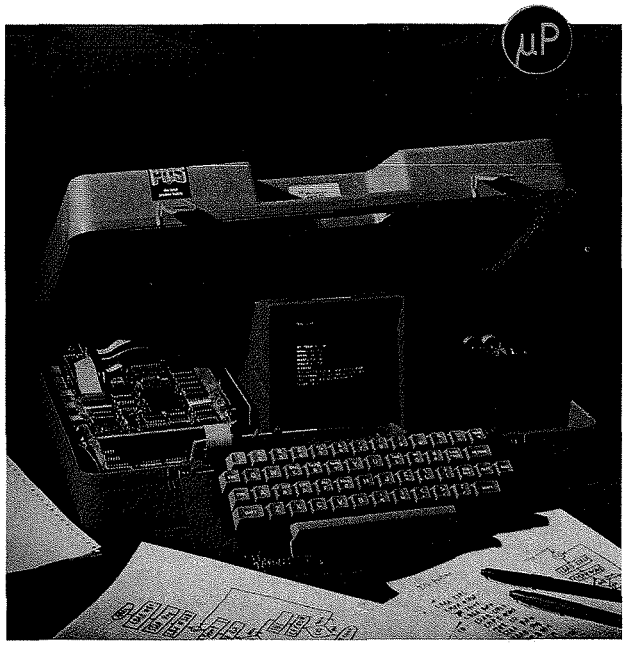
Vorwort

Hardware und Software sind bei Mikroprozessoren untrennbar miteinander verbunden. In der ELEKTRONIK wurde zuerst die Hardware ausführlich behandelt, da sie dem Schaltungsentwickler grundsätzlich nicht fremd ist und an das vorhandene Wissen leichter angeknüpft werden kann. Aus den zahlreichen Artikeln entstand ein Sonderheft, das sehr großen Erfolg hatte. Als folgerichtige Weiterführung dieses Konzepts wurde seit Ende 1976 das Schwergewicht auf die Software gelegt. Im vorliegenden Sonderheft II sind die seither in der ELEKTRONIK zu diesem Thema erschienenen Beiträge zusammengestellt. Daneben sind auch Artikel enthalten, die bisher noch nicht veröffentlicht sind oder nur in diesem Sonderheft erscheinen.

Ähnlich wie schon im ersten Sonderheft I, das jetzt in einer erweiterten 2. Auflage erscheint, legten wir auch hier größten Wert auf eine fundierte Einführung in die Grundlagen. Da es aber gerade für die Programmierung von Mikroprozessoren keinen „Königsweg" gibt, wird die Problematik von verschiedenen Seiten beleuchtet und erklärt. Der gesamte Abschnitt „Grundlagen und Programmierpraxis" vermittelt eine sichere Basis für den tieferen Einstieg.

Dem Fortgeschrittenen bietet dieses Heft unter anderem konkrete Anwendungsbeispiele, in denen zahlreiche Ideen für eigene Entwicklungen stecken, eine Übersicht über Entwicklungshilfsmittel und eine Einführung in die Logik-Analyse. Selbst ein kritischer Blick auf die „Kehrseite der Medaille Mikroprozessor" wurde nicht vermieden. Wir haben ihn an den Anfang gestellt – nicht um den Leser abzuschrecken, sondern um ihm den Blick zu schärfen für die Klippen auf dem Weg zum erfolgreichen Mikroprozessor-Anwender.

Die ELEKTRONIK-Redaktion



Die Software für Mikroprozessoren wird auf Entwicklungssystemen erstellt. Viele künftige Anwender scheuen jedoch die hohen Anschaffungskosten. Ihnen bietet die Firma Motorola mit dem PDS-System eine preiswerte Alternative, die sich für den Anfang als eigenständiges Entwicklungssystem und für später als Terminal einer größeren Anlage eignet. Mehr darüber erfahren Sie auf den Seiten 59...63.
(Foto: D. Lechner)

Mikrocomputer und Mikroprozessor

Der Mikrocomputer ist der „kleine Bruder" des Minicomputers. Er sieht ihm ähnlich und löst ähnliche Aufgaben, wird auch in den gleichen Wortlängen geliefert, hat aber im allgemeinen weniger Speicherkapazität, die der Kompaktheit und der niedrigen Kosten wegen ausschließlich durch Halbleiterspeicher dargestellt wird, während der Minicomputer noch gerne Kernspeicher verwendet. Auch die Zahl der Befehle, der Register usw. ist beim Mikrocomputer geringer – also eine Sparausgabe des Minicomputers, die aber für bestimmte Aufgaben völlig genügt. Der Mikroprozessor ist das „Herz" des Mikrocomputers. Umgekehrt: Aus dem Mikroprozessor kann ein Mikrocomputer werden, wenn man ihn mit Zusatzspeichern, Schaltern, Anzeigelämpchen, Netzteil und Gehäuse ausrüstet. Diese Übergangsmöglichkeit sowie die Übereinstimmung der Architektur und der Aufgabenbereiche führen häufig zur Verwischung der Grenzen zwischen Mikroprozessor und Mikrocomputer. Das „Herz" des Mikroprozessors wiederum ist die CPU, die zentrale Datenverarbeitungs- und Recheneinheit. Sie wird heute stets auf einem einzigen Chip in LSI-Größe realisiert, und zwar in allen gängigen Halbleiter-Technologien, wie P-Kanal-MOS mit Silicon Gate, aber auch in N-Kanal-MOS, CMOS, SOS und sogar in Bipolartechnik. Im Extremfall werden auf den CPU-Chips auch ROMs und RAMs sowie Ein-/Ausgangslogik gepackt, um den gesamten Mikroprozessor in einem Baustein unterzubringen – gängiger-aber ist es, den Mikroprozessor auf mehrere Bausteine zu verteilen und auf einer Leiterplatte aufzubauen. Im Gegensatz zu festverdrahteten Logik-Netzwerken oder Einzweck-Rechnern sind Mikroprozessoren frei programmierbar und somit der jeweiligen Aufgabe flexibel anpaßbar. Mikroprozessoren können mikroprogrammierbar sein, oder sie enthalten eine Anzahl festverdrahteter Befehle, vorzugsweise per ROM gespeichert und damit gegen Verlust durch Stromabschaltung gesichert.

Kurze Produktbesprechungen

1978

Franzis-Verlag GmbH, München, Karlstr. 37
Redaktion: Rudolf Hofer, für den Inhalt verantwortlich Hans J. Wilhelmy, beide in München.

Druck: Franzis-Druck GmbH, München. Printed in Germany. Imprimé en Allemagne

INHALT

Dipl.-Ing. Hans-Jürgen Labudda

Mikroprozessor – Kehrseite der Medaille

Den Slogan „Mikro macht den Markt munter" wählte ein Aussteller auf der Electronica 76. Seine Gültigkeit soll hier nicht bezweifelt werden. Eines läßt sich aber auch mit Sicherheit behaupten, nämlich, daß viele nach dem Besuch der Ausstellung verwirrter und ratloser als jemals zuvor waren. Warum? Nun, das Angebot ist mehr als vielfältig und die Sprachverwirrung ist groß. Mit dieser Situation befaßt sich der nachfolgende Beitrag, der – obwohl „lockerleicht" geschrieben – dennoch interessante Informationen enthält und schließlich auch zum Nachdenken anregt. Eines steht allerdings fest: Eine Mikroprozessor-Kontrastellung nützt niemandem, dazu ist er zu wichtig, und vor den Erfolg haben die Götter bekanntlich „den Schweiß gesetzt". Schließlich sind die Röhre, der Transistor und die integrierte Schaltung auch nicht „über Nacht" und ohne Schwierigkeiten aufgenommen worden.

Schlagworte wie „microcomputer design is a snap" oder „the simple cheap microprocessor" tauchen immer häufiger in namhaften Fachzeitschriften auf; sie seien hier stellvertretend für eine ganze Serie rosiger Werbesprüche herausgegriffen. Allzu leicht erliegt der unbefangene Leser ihren Verlockungen, sei er nun Techniker oder Technischer Direktor. Zweifellos stellt die Entwicklung des µP (amerikanisches Kürzel für Mikroprozessor) einen technologischen Fortschritt dar, dessen immense Bedeutung sich erst abzuzeichnen beginnt; er wird zwangsläufig eine Reihe von bewährten Konstruktionsprinzipien in vielen Bereichen der Technik stark verändern, wenn nicht gar ganz verdrängen. Der folgende Beitrag soll all jene auf den Boden harter Realitäten stellen, die bereits im Taumel allgemeiner Begeisterung mit dem Gedanken spielen, diesen preiswerten Alleskönner unbesehen in die Serienproduktion eigener Geräte oder Steuerungen aufzunehmen.

Sprachverwirrung

Die µP-Hersteller und ihre Vertriebsfirmen bearbeiten den Markt mit Werbesprüchen, die Waschmittelproduzenten nicht schlecht anstünden: Jeder hat das beste, universellste Produkt mit der einfachsten Handhabung, und fleißig werden fremdartig klingende Formulierungen bemüht. Dabei sind die erstaunlich niedrigen Preisangaben – teilweise 10 Dollar pro Stück – noch recht leicht als Augenwischerei zu entlarven; wer ordert schon – zumindest in der Anfangsphase – zehntausend Stück auf einen Schlag!

Der wichtige Unterschied zwischen den beiden Mikros, dem Mikroprozessor und dem Mikrocomputer, bleibt dagegen oft unklar: Mikroprozessor heißt jenes Stück Keramik oder Plastik mit rund 40 Anschlüssen, das eine Siliziumscheibe von etwa 5 mm mal 5 mm umhüllt. Hier spielt sich das komplizierte Innenleben des µP ab; Additionsvorgänge, Datentransfer, Shifts, logische Verknüpfungen und Entscheidungen wechseln in sinnvoller Reihenfolge.

Erst wenn weitere Bauelemente wie z.B. Daten- und Programmspeicher, Taktgenerator, Interface-Einheiten, Stromversorgung, Schalter, Lampen, Stecker, Kabel und ein Gehäuse hinzugefügt werden, kann man mit etwas gutem Willen von einem Mikrocomputer sprechen.

Klassen-Einteilung

µP ist nicht gleich µP; allein die unterschiedlichen Einsatzgebiete rechtfertigen schon gewisse Klassen, und zwar die der 4 bit, 8 bit oder gar 16 bit parallel verarbeitenden Chips. Jedermann rät, den 4-bit-„Winzling" mit einfachen Steuer- und Entscheidungsaufgaben oder mit der Verarbeitung von Zahlen zu betrauen, das 8-bit-„Mittelding" in der bekanntlich byte-orientierten Datentechnik einzusetzen und den 16-bit-„Tausendsassa" in eine komplexe Prozeßsteuerung oder in einen komfortablen Minicomputer zu stecken.

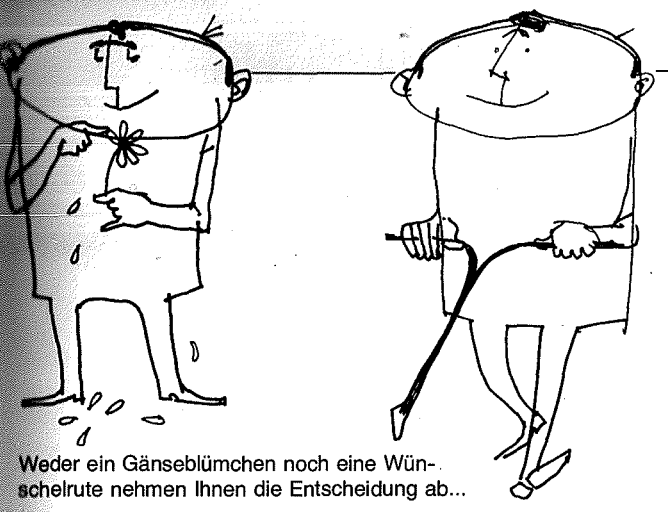
Diese Einteilung kann aber bestenfalls als Richtschnur dienen; denn die Mischzone zwischen zwei reinen Anwendungsgebieten ist oftmals breiter als diese selbst. Schon mancher hat auf eine 4-bit-Maschine gesetzt und mußte später auf einen 8-bit-Typ umsteigen. Ebenso warten viele auf das Vordringen der 16-bit-Chips, obwohl sich ihr spezielles Problem bereits heute ohne Schwierigkeiten mit einem gängigen 8-bit-Typ preisgünstig lösen ließe.

Auch die sogenannten „Mikroprozessor-Slices" – meist 4 bit orientierte Chips, die sich zu 8, 12, 16 bit usw. aneinanderreihen lassen – bieten keinen Ausweg; zu umfangreich sind die benötigten Zusatzbauelemente wie Carry-look-ahead, Mikroprogrammspeicher, Sequenzer, Multiplexer usw., so daß der Einsatz vorerst nur in der „echten" Computertechnik oder in speziellen Projekten sinnvoll scheint.

Qual der Wahl

Zur Zeit tummeln sich mehr als 40 verschiedene µPs unterschiedlicher Hersteller zumindest in den Fachgazetten, wobei niemand so recht weiß, welche Eintagsfliegen bleiben werden und welche als die künftigen Renner gelten. Wer als Anwender heute mit großem finanziellen Aufwand auf das falsche Pferd setzt, steht unter Umständen nach zwei Jahren ohne Roß allein auf weiter Flur. Hat man dagegen einen µP gewählt, der später in Massen verkauft wird, partizipiert man automatisch am sinkenden Preis.

Ganz sicher wird man jedoch bald eine Flurbereinigung erleben, wobei die wirtschaftliche Macht der Großen ebenso eine Rolle spielt wie die relativ unwägbaren Faktoren Technologie (P-Kanal, N-Kanal, CMOS, bipolar, I^2L), Geschwindigkeit (0,1 bis 10 µs Zykluszeit), Befehlsvorrat (50 bis 100), Adressierungskomfort (direkt, indirekt, absolut, relativ, indiziert, immediate), Komplexität der Bauelemente, Leistungsverbrauch, Spannungs- und Taktversorgung. Viel Erfahrung gepaart mit einem Schuß Intuition ist nötig, um auf Anhieb die richtige Wahl zu treffen.



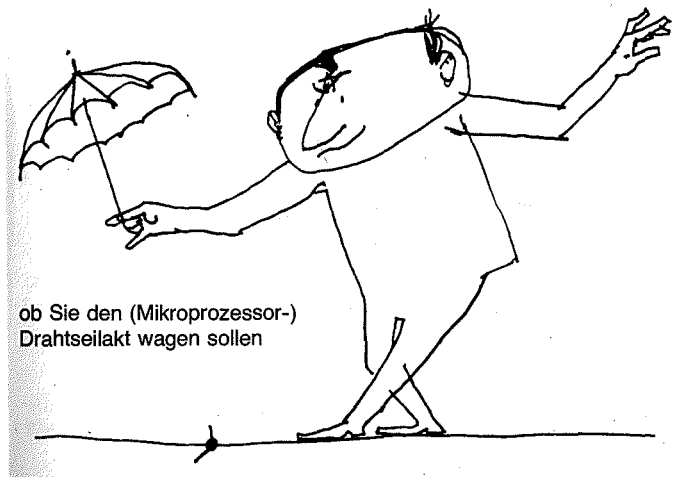
Weder ein Gänseblümchen noch eine Wünschelrute nehmen Ihnen die Entscheidung ab...

Fremde Software

Daß ein µP ohne das passende Programm nicht arbeiten kann, hat sich herumgesprochen. Manche Leute nennen es Mikroprogramm, was nicht nur falsch ist, sondern auch gefährlich wird, sobald damit die Vorstellung von einem niedlichen und harmlosen Ding einhergeht. Software für moderne „µPen" kann sich ohne weiteres mit der von bekannten Minicomputern messen; entsprechend aufwendig gestaltet sich daher auch die Entwicklung.

Mikroprozessoren reagieren auf Bitmuster – aber je nach Typ vollkommen unterschiedlich: Beispielsweise führt der Typ A eine Addition, der Typ B einen Ladevorgang aus, wenn ihnen das Bitmuster 11001110 als Befehl angeboten wird. Bitmuster sind für uns nicht gerade anschaulich, dagegen sagt der Ausdruck ADD fast schlüssig aus, daß ein Additionsbefehl vorliegt. Die Verwendung dieser sogenannten mnemotechnischen Ausdrücke verbunden mit symbolischen Adressen erleichtert die Arbeit des Programmierens ungemein. Man nennt diese Sprachebene Assembler-Sprache. Ein passendes Übersetzungsprogramm, der Assembler nämlich, führt das in der Assembler-Sprache geschriebene Programm in die passenden Bitmuster, die sogenannte Maschinen-Sprache, über.

Jeder µP-Hersteller bietet den zu seinem µP-Typ passenden Assembler entweder in Form von ROMs (resident assembler) oder in Form eines Magnetbandes bzw. auf Lochkarten (cross assembler) an. Es ist gleichgültig, welche Version man kauft, beide nehmen sich recht hübsch auf dem Schreibtisch aus. Es sei denn, man kauft zum Beispiel des Resident-Assemblers mindestens ein sogenanntes Prototypsystem mit Fernschreibmaschine zum Preis von etwa DM 15 000.– hinzu. Noch anspruchsvoller gebärdet sich der Cross-Assembler, erfordert er doch eine komplette Minicomputer-Anlage im Wert von mindestens DM 40 000.–, die zudem Fortran IV verstehen muß. Alle gängigen Cross-Assembler sind in Fortran IV geschrieben; aber man hüte sich vor der Annahme, daß ein Fortran IV-Programm auf dieser Maschine auch auf Anhieb läuft; denn stets sind bestimmte Zusatzkonventionen zu beachten, die von Computer zu Computer variieren.



ob Sie den (Mikroprozessor-) Drahtseilakt wagen sollen

Gerne wird mit Compilerprogrammen geworben. Sie sind wie der Assembler Übersetzungsprogramme mit jedoch wesentlich gesteigertem Komfort für den Benützer. Ein geübter Programmierer erzielt damit in der Tat einen beachtlichen Zeitgewinn während der Programmentwicklung. Dafür rächt sich die Elektronik später mit im Vergleich zur Assemblerprogrammierung größerem Speicherbedarf, längeren Programm-Laufzeiten und schlechtem Realzeitverhalten.

Ein unnötiges Zuviel im Programmaufwand von vielleicht 100 Bytes kann u.U. die Kosten für den Speicher verdoppeln. Man programmiere deshalb stets so nahe wie möglich am Ort des Geschehens, d.h. in Assembler! Für ganz simple Anwendungen – wenige hundert Bytes – genügt dagegen die „Zu-Fuß-Methode", das Programmieren mit Bitmustern, also in der Maschinensprache.

Vergessen kann man auch die Simulationsprogramme. Sie erfordern wie der Compiler auf jeden Fall einen größeren Computer und bilden zudem nur das logische Verhalten des µP-Chips, niemals aber das Realzeitverhalten und die I/O-Beziehungen innerhalb des gesamten Systems nach.

Man sollte dagegen den allergrößten Wert auf ein gutes Testprogramm (debugger) legen. Es gestattet nicht nur das schrittweise Austesten eines Programms im Zeitlupentempo, sondern stoppt auf Wunsch an vorgegebenen Stellen, druckt den Inhalt von Speicherplätzen und Registern aus und erlaubt Änderungen von Befehlen oder Daten. So kann der Benutzer leicht einen Blick in das Innenleben seines Systems werfen, Befehlsketten verfolgen, Datenflüsse beobachten und so Fehler erkennen.

Später zeigen weitere Programmhilfen ihre Nützlichkeit wie Ladeprogramm (linking loader), Editor (zum Aufbau von Texten) und Monitor (koordiniert und überwacht verschiedene Programmabläufe sowie die Aktivitäten des Operateurs am Teletype).

Um eines kommt der ernsthafte Benutzer niemals herum: Knochenarbeit zu leisten, indem er das vom Programm beeinflußbare Innenleben des µP sowie sein Befehlsrepertoire bis auf den i-Punkt erforscht.

Eigene Software

Mit den eben genannten Programmhilfen kann der µP-Hersteller alle Kunden gleichermaßen versorgen. Seine spezifische Anwender-Software muß der Käufer dagegen selbst erstellen, was im Sinne einer Befriedigung von Schutzbedürfnissen nicht unbedingt einen Nachteil bedeutet. Dafür ist diese Art von Software besonders teuer; denn sie wird eigens für ein bestimmtes Produkt entwickelt. Man erwarte dabei keinerlei Hilfe von den µP-Herstellern oder deren Distributoren, sofern man nur bescheidene Stückzahlen im Auge hat. Selbst für einige hundert Systeme reißt man sich dort kein Bein aus. Entweder überträgt man die Entwicklung einem geeigneten Software-Haus oder führt sie im eigenen Hause durch. Ein Nur-Programmierer ist hierbei jedoch fehl am Platz, zu eng sind in der Regel die Verflechtungen zwischen Programm und Außenwelt.

Neben den reinen Personalkosten sind Investitionen für verschiedene Hilfsmittel (Rechner, Prototypsystem, Programmiergerät, Datenspeicher, Arbeitsplatz) zu berücksichtigen, die leicht das Jahresgehalt des Mitarbeiters übersteigen können. Eine gute Kraft benötigt erfahrungsgemäß zwei bis vier Monate, um ein Programm mit etwa 1000 Befehlen zu schreiben; Test, Debugging und Dokumentation sind eingeschlossen.

Es kommt vor, daß die Laufzeit des fertigen, fehlerfreien Programms länger ist, als es die Aufgabenstellung zuläßt.

Ob Sie dabei auf den Bauch fallen...
oder am Ende erfolgreich sind...

Dann muß die Arbeit mehrerer Monate u.U. weggeworfen und eine geänderte Programmstruktur erarbeitet werden. Der Grund: Man hat versäumt, zu Beginn der Entwicklung das Realzeitverhalten des Prozesses genau zu untersuchen. Stellt sich heraus, daß der µP bereits am Ende seiner Leistungsfähigkeit ist, bleibt nur das Ausweichen auf einen anderen Typ, wobei das erworbene Know-how und die installierte Hardware wertlos werden. Zudem schlägt der Zeitverlust von einem halben Jahr zu Buche, so daß der Einsatz des µP überhaupt fraglich werden kann.

Dieses Beispiel soll verdeutlichen, daß am Anfang einer Entwicklung die Problemanalyse mit einer ausgedehnten Systembetrachtung stehen muß; dieser Zeitaufwand wird sich immer auszahlen.

Zusatzkosten

Glaubt man den Herstellern, so funktioniert ihr µP fast ohne Hilfsmittel, besonders wenn Taktgenerator und Powerfail/Restart auf dem µP-Chip integriert sind. Meistens aber kommen spezielle Interface-Schaltungen, Transistoren, Triacs, Relais, Analog-Digital-Umsetzer, Anzeigelampen, Schalter, Platinen, Stecker, Kabel, Stromversorgung und Gehäuse hinzu, so daß deren Kosten leicht das Dreifache der Kosten für µP und Speicher erreichen.

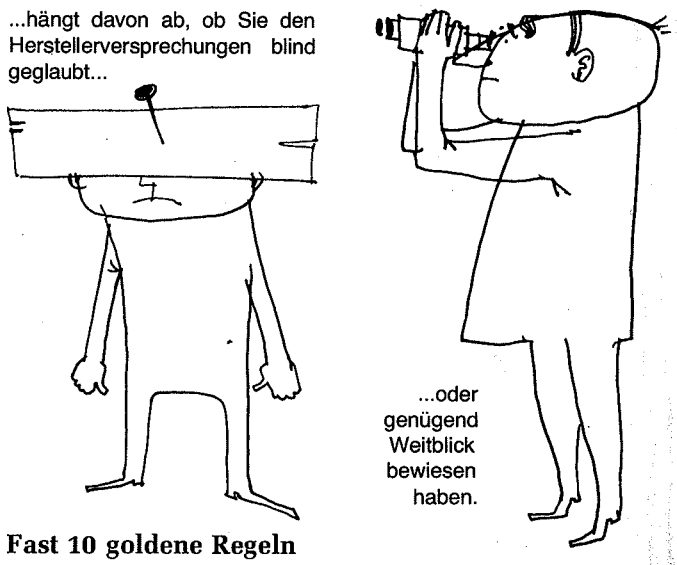
Man überprüfe auch, ob die Fertigung in der gewohnten Form weiter arbeiten kann oder ob nicht kostspielige Veränderungen vorgenommen werden müssen, die den speziellen Erfordernissen der hochintegrierten und teilweise empfindlichen Bauelementen Rechnung tragen. Wie soll der Wareneingang 200 µP-Bausteine auf korrekte Funktion innerhalb des vorgegebenen Frequenz- und Temperaturbereiches prüfen? Passende Prüfautomaten sind teuer, der Selbstbau ist auch nicht gerade billig.

Teure Überraschungen

Nach vielen Mühen arbeitet das neue System endlich zufriedenstellend, die Nullserie hat harte Prüfungen erfolgreich bestanden und der Start für die erste Serie von 500 Einheiten steht bevor. Da passiert's: Der Hersteller X stellt die Fertigung des sorgfältig ausgewählten µP Y ein, der Distributor schimpft und Sie säßen auf dem Trockenen, wenn Sie nicht frühzeitig den Hersteller Z als „second source" ins Auge gefaßt hätten. Aber ein Anruf bei dessen Vertretung genügt, um für den Rest des Tages zerstört am Boden zu liegen: second source? – ja, die Lizenzen habe man auch, aber die Produktion laufe noch lange nicht, zudem müsse man sich angesichts der Pleite des Partners überlegen, ob ...

Störenfriede

In einer mit Störungen verseuchten Umgebung wird ein µP nur mit Mühe arbeiten können, da der Störabstand der allgemein verwendeten Bauelemente zu gering ist. Dies gilt besonders in den Fällen, wo Signal- oder Steuerleitungen ungeschützt in der Nachbarschaft von Starkstromkabeln verlegt wurden. Jedes Schütz, jeder Motor bringt dann die Elektronik aus dem Tritt. Oft hilft nur der konsequente Einsatz von Optokopplern oder HINIL-Bauelementen, Netzfiltern und Transformatoren mit Innenschirm aus der Not.



Fast 10 goldene Regeln

1) Nehmen Sie nur die Hälfte der Werbeaussagen für bare Münze; verdoppeln Sie dafür ihr Mißtrauen.
2) Analysieren Sie genau Ihr Problem, bevor Sie sich für eine 4-bit- oder 8-bit-Maschine entscheiden. Überlassen Sie die 16-bit-Typen den Computerbauern.
3) Wählen Sie keine Exoten, orientieren Sie sich an der Produktionsmenge der großen Hersteller. Holen Sie ggf. den Rat eines neutralen Spezialisten ein.
4) Überlassen Sie Compiler und Simulatoren ruhig den Theoretikern. Unentbehrlich sind Assembler und ein gutes Debug-Programm.
5) Anwender-Software kostet das meiste Geld. Wenn Sie im eigenen Hause entwickelt werden soll, beachten Sie die nötigen Hilfsmittel.
6) Meistens benötigen Sie neben den µP-Bauelementen zusätzliche periphere Dinge, die den Preis beträchtlich in die Höhe treiben.
7) Prüfen Sie die Lieferfähigkeit „Ihres" µP-Herstellers. Mindestens eine echte „second source" muß verfügbar sein.
8) In der Regel müssen Sie spezielle Maßnahmen vorsehen, wenn ein µP-System in störungsverseuchter Umgebung betrieben werden soll.
9) Verlieren Sie trotz allem nicht die Lust am „Mikroprozessieren". Der µP ähnelt einem Super-Rennwagen; wer ihn zu fahren weiß, erreicht leicht und sicher als einer der ersten das gewünschte Ziel, der weniger Geübte dagegen kann im Graben landen.

(Die Karikaturen wurden mit freundlicher Genehmigung der Siemens AG dem Buch „Kennen Sie Mikrocomputer?" entnommen.)

Dipl.-Ing. Hans-Jürgen Labudda, geboren in Berlin, studierte in Aachen Nachrichtentechnik und trat 1968 bei AEG-Telefunken im Konstanzer Computerwerk ein, wo er mit Entwicklungs-Aufgaben auf dem Gebiet der Datenverarbeitung betraut war. Als Senior-Ingenieur spezialisierte er sich auf mikroprozessorgesteuerte „Intelligente Terminals". Ab 1975 untersuchte er die Einsatzmöglichkeiten von Mikroprozessoren in der industriellen Steuerungs- und Regelungstechnik bei der Firma Zentra A. Bürkle KG. Heute ist er bei der Dornier System GmbH in Friedrichshafen als System-Ingenieur tätig und beschäftigt sich mit Systemplanung und Realisierung von prozeßrechnergesteuerten Anlagen.
Hobbies: Segeln, Filmen, Reisen, Gitarre.
Privattelefon: (0 75 31) 4 32 79
ELEKTRONIK-Leser „fast von Anfang an".

Otmar Feger

Grundsätzliche Arbeitsweise bei der Mikrocomputer-Programmierung

Wer Mikroprozessoren einsetzen will, muß sie beherrschen. Dazu gehören umfassende Kenntnisse der Hardware und Software. Während die Mikroprozessor-Hardware dem Elektroniker meist weniger Schwierigkeiten bereitet, fehlt ihm zur Software oft der Zugang. Speziell auf die Einführung in die Mikroprozessor-Software zugeschnittene Literatur fehlt weitgehend.

Der folgende Aufsatz ist der erste von mehreren Beiträgen zur Programmierung von Mikroprozessoren. Bewußt befaßt er sich ausschließlich damit, in welchen Schritten man bei der Lösung eines Problems vorgehen sollte. Dem Praktiker mögen diese Gedanken zwar etwas „philosophisch" anmuten, sie ersparen ihm aber viel Zeit und Mühe, denn der richtige Arbeitsstil ist genauso wichtig wie etwa das Verstehen von Befehlen.

Die Softwarehandbücher der Mikroprozessor-Hersteller setzen die Kenntnis des Programmierens mehr oder weniger voraus. Gelegentlich wird an Beispielen, etwa dem Ablaufdiagramm eines Waschmaschinenprogrammes, die Logik der Programmierung erklärt und das Weitere der Phantasie des Lesers anheimgestellt. Hinweise darüber, was in das Erlernen des Programmierens zu investieren ist, fehlen meist. Dies täuscht leicht darüber hinweg, daß Umfang und Komplexität des Softwaregebietes keineswegs trivial sind. Nicht ohne Grund hat sich die Informatik in wenigen Jahren an den Universitäten fest etabliert. Trotzdem ist kein Informatikstudium erforderlich, um zu lernen, wie Mikroprozessoren zu programmieren sind. Die nicht gerade billigen Seminare vermitteln meist mikroprozessorspezifisches Grundlagenwissen. Und genausowenig, wie in beispielsweise einer Woche eine Sprache zu erlernen oder ein Elektroniker auszubilden ist, genügt dies, um das Programmieren zu erlernen.

Der Begriff Programmieren beinhaltet zwei Tätigkeiten: Das Lösen eines Problems durch das Finden eines Lösungsverfahrens und die Realisierung dieses Verfahrens in Form eines Programms für einen verfügbaren Computer. Besonders bei der Mikroprozessor-Programmierung lassen sich beide Tätigkeiten kaum trennen. Dies liegt daran, daß die Problemlösungen aus Kostengründen möglichst gut an die Eigenschaften des Mikroprozessors anzupassen sind. Wie beim Erlernen einer Sprache, ist beim Erlernen einer Programmiersprache viel Übung erforderlich, um sich ein Gefühl bzw. die Erfahrung für die Möglichkeiten anzueignen. Übung heißt in diesem Falle Kennenlernen von Programmen und Lösen selbstgestellter Aufgaben. Wie bei einer Sprache, führt dies mit der Zeit dazu, in der Programmiersprache zu denken und ökonomisch zu formulieren.

1 Die Arbeitsweise beim Programmieren

Der „Profi" wird Philosophieren über den richtigen Arbeitsstil beim Programmieren für überflüssig halten. Die Praxis zeigt jedoch, daß der Anfänger oft vermeidbare Fehler begeht. Meist fehlen ihm eine geeignete Anleitung oder ein nachahmenswertes Vorbild. Auch Fachbücher gehen nur selten auf diese Thematik ein. Ein Grund mag sein, daß den Fachleuten die vorausschauende Planung und gewisse methodische, kontrollierende Arbeitsweisen selbstverständlich geworden sind. Ein weiterer Grund ist, daß an kleinen, überschaubaren Beispielen, die in den Rahmen eines Vortrags oder Aufsatzes passen, die Notwendigkeit eines bestimmten Arbeitsstils nicht leicht zu demonstrieren ist. Fehler, die in diesen Fällen in Minuten zu beheben sind, können größere Projekte möglicherweise zu Fall bringen oder erheblich verzögern. Vielen Praktikern liegt die rein theoretische Arbeit nicht. Trotzdem sollten sie ihre Arbeitsweise kritisch sehen, nach dem Wirkungsgrad beurteilen und zu verbessern trachten. Die Unbestechlichkeit des Computers hilft dabei.

Der nachfolgende Abschnitt geht auf die einzelnen Phasen bei der Programmentwicklung näher ein. Der häufigste Fehler ist wohl, daß ein Bearbeitungsschritt begonnen wird, bevor die dafür notwendigen Vorarbeiten abgeschlossen sind. Der Anfänger beginnt oft mit dem Codieren des Programms, bevor er hinreichend über die gestellte Aufgabe bzw. den Lösungsweg nachgedacht hat. Auch brennt er darauf, das eben codierte Programm auf dem Computer laufen zu lassen. Die Folge sind umfangreiche Testzeiten und u.U. mehrfaches Neuschreiben von Programmteilen. Programmierer, die aus Unkenntnis oder Bequemlichkeit diesen Arbeitsstil kultivieren, argumentieren gelegentlich, der Computer fände einen Fehler viel schneller als der Programmierer beim mühsamen Durchdenken des Programms. Vordergründig mag dies stimmen, nur ist die Fehlersuche in einem Programm eine der schwierigsten und zeitraubendsten Tätigkeiten. Die Zahl der möglichen Fehler und damit der entsprechenden Reaktionen ist sehr groß. In der Praxis bewegt sich der Zeitbedarf je nach Arbeitsstil etwa zwischen folgenden Extremen, was an einem hypothetischen Beispiel gezeigt sei:

Fall 1: Zwei Wochen Denken und Codieren. Zwölf Wochen Testen und Neucodieren. Die Testzeit ist deshalb so lang, weil nach langwieriger Fehlersuche oft auch Konzeptfehler gefunden werden, die teilweises Neucodieren erfordern usw.

Fall 2: Vier Wochen Denkarbeit am Schreibtisch. Eine Woche Codieren, eine Woche Kontrollieren am Schreibtisch und eine Woche Austesten.

Im zweiten Fall wird nur die halbe Zeit benötigt. Solche Unterschiede sind in der Praxis keinesfalls utopisch.

2 Die einzelnen Schritte der Programmentwicklung

Wie schon angedeutet, beinhaltet der Begriff Programmieren die Lösung eines Problems durch einen Algorithmus (ein Lösungsverfahren) und die anschließende Programmierung des Algorithmus. Die Phasen sind im wesentlichen:

Aufgabenspezifikation

Das Ziel ist die exakte Beschreibung der ursprünglich meist vage vorliegenden Aufgabe. Umgebungs-, System- und Wirtschaftlichkeitsüberlegungen spielen hierbei eine Rolle.

Lösungsverfahren

Die gestellte Aufgabe soll durch das gefundene Verfahren gut gelöst werden. Aus dem Lösungsverfahren ergibt sich die Aufgabe der Programmkonstruktion.

Programmkonstruktion

Die Programmkonstruktion umfaßt die Planung und die Codierung des Programms.

Test

Der Programmtest soll die (immer) vorhandenen Fehler aufdecken.

Dokumentation

Die Dokumentation muß auf dem neuesten Stand und ausreichend ausführlich sein.

2.1 Aufgabenspezifikation

Ziel dieser Projektphase ist, aus einer mehr oder weniger vagen Aufgabenstellung und verschiedenen Rahmenbedingungen zu einer exakten, vollständigen, schriftlich fixierten Beschreibung der gewünschten Leistung eines Programms zu kommen. In dieser Phase sind Überlegungen über den Umfang bzw. die Allgemeingültigkeit des anzustrebenden Verfahrens anzustellen. Da das Ergebnis solcher Überlegungen die Lebensdauer des Programms beeinflussen kann, sei darauf näher eingegangen: Auf der einen Seite steht der Wunsch nach einer genau auf das spezielle Problem zugeschnittenen Lösung. Diese soll mit einem Minimum an Zeit und Kosten realisiert werden. Dem gegenüber steht der Gedanke, durch Verallgemeinern der Problemstellung eine gewisse Universalität und damit vielseitigen Einsatz zu ermöglichen. Dies verursacht zusätzliche Kosten, die u.U. auf das Vielfache einer unmittelbaren Lösung anwachsen. Solche Überlegungen sind bei größeren Projekten wichtig. Es gab Programme, die vor der Fertigstellung veralteten, weil sich die Umweltbedingungen änderten. Beispielsweise wurde ein Programm zur automatischen Konstruktion gedruckter Schaltungen auf eine bestimmte Kartengröße und für bestimmte Bausteintypen festgelegt. Später war es praktisch kaum zu verwerten. Weitere Überlegungen beziehen sich auf die Standardisierung der Ein- und Ausgangsdaten, um die Leistungen eines Programms per Programm weiterverarbeiten zu können. Generell flexible Programme zu schreiben, ist andererseits aus Zeit- und Kostengründen meist nicht möglich. Die zu schließenden Kompromisse könnten so aussehen, daß für die Planungsphase die erforderliche Zeit für solche prinzipiellen Überlegungen einzukalkulieren ist. Ein meist gangbarer Kompromiß ist die Planung eines komfortableren Programms, das dann zuerst in der unbedingt erforderlichen Ausbaustufe realisiert wird. Je nach Bedarf kann es später einfach erweitert werden, da dies von vornherein vorgesehen ist. Auf diese Weise verbaut man sich nicht den Weg für später erforderliche Änderungen oder Erweiterungen.

2.2 Lösungsverfahren

Die eben beschriebenen Überlegungen beeinflussen unmittelbar den folgenden Arbeitsschritt – das Suchen nach einem Lösungsverfahren. Im Bereich der numerischen Mathematik liegen Algorithmen als Formeln vor. Bei vielen organisatorischen Aufgaben sind meist Algorithmen zu suchen. Für ein bestimmtes Problem ist oft eine Vielzahl mehr oder weniger guter Lösungen denkbar. Der Programmierer gibt sich mit einer Lösung zufrieden, die den geforderten Bedingungen entspricht und mit vernünftigem Zeitaufwand realisierbar ist. Die Suche nach einem geeigneten Lösungsweg sollte sich soweit wie möglich auf der logischen Ebene abspielen. Es ist falsch, jetzt schon nach den Befehlen zu schielen. Grenzfälle sind schwer erfüllbare Zeit- oder Speicherplatzanforderungen. In diesen Fällen spielen die Realisierungsmöglichkeiten auf einem bestimmten Computertyp in die Wahl des Lösungswegs hinein. Dies setzt die Kenntnis des verfügbaren Computers voraus. Daß diese Kenntnis nicht überflüssig ist, beweisen Programme, die trotz eleganter Lösungen des Problems nicht einsatzfähig waren, weil beispielsweise die Laufzeit eines Programms mehrere Tage betrug. Ein wichtiges Hilfsmittel bei der Problemlösung sind Flußdiagramme, besonders bei komplexen Abläufen erlauben sie eine übersichtliche Darstellung des Programm- bzw. Datenflusses. Das Vorgehen bei der Problemlösung unterscheidet sich nicht von entsprechenden Verfahren in anderen Bereichen:

- Sammeln von Fakten (wie wurden gleiche oder ähnliche Probleme gelöst),
- Ordnen der Daten,
- Auswerten der Daten.

Dabei wird das Problem erforderlichenfalls in Teilprobleme zerlegt, um diese dann nacheinander zu lösen. Bei den Teilproblemen werden wiederum zuerst die Randbedingungen beschrieben. Es gilt für die Teilprobleme prinzipiell das gleiche wie für die Lösung des Gesamtproblems.

2.3 Programmkonstruktion

Ein gutes Programm ist leicht zu bedienen und übersichtlich (modular) zu konstruieren. Dadurch ist es leicht zu verstehen und zu warten. Programmiertricks sind deshalb zu vermeiden. Sie sind meist schwer verständlich und nicht leicht zu ändern. Größere Programme werden nach funktionellen Gesichtspunkten in einzelne Teilprogramme, sogenannte Moduln, zerlegt. Diese Methode erhöht die Übersichtlichkeit und leichte Beschreibbarkeit des Programms. Solche Teile können sein: Eingabe, Problemlösungsteil, Ausgabe, Fehlerbehandlung. Jeder dieser Teile ist möglicherweise weiter zu zerlegen. Die Leistungen und Schnittstellen der Moduln sind gut zu dokumentieren. Gelingt es, die Schnittstellen zu standardisieren, lassen sich Teilprogramme leicht anschließen bzw. austauschen. Bei der Programmkonstruktion muß ferner folgenden Komplexen besondere Beachtung geschenkt werden:

- Besondere Überlegungen sind über den Bedienkomfort angebracht. Eingaben in das Programm sollten angefordert und quittiert werden. Dadurch wird die Gefahr der Fehlbedienung vermindert.

- Das Programm ist gegen fehlerhafte Handhabung abzusichern. Die eingegebenen Daten sind zu prüfen. Der Programmaufwand kann dabei durchaus den Teil zur Lösung des Problems übersteigen.
- Die leichte Wart- und Änderbarkeit des Programms hängt mit von der Programmkonstruktion ab. Gelegentlich lassen sich elegante und leistungsfähige Verfahren zur Problemlösung finden, die jedoch schwer zu verstehen und zu ändern sind. In solchen Fällen ist zu prüfen, ob es überschaubare Lösungen gibt. Diese verlängern erfahrungsgemäß die Lebensdauer des Programms.
- Schon beim Programmentwurf wird über die leichte Testmöglichkeit und damit über den Zeitaufwand für die Testphase mit entschieden. Oft werden besondere Programmteile zur übersichtlichen Ausgabe von Zwischenergebnissen vorgesehen. Diese, nach den einzelnen Programmoduln bei Bedarf aufzurufenden Ausgabeprogramme, ermöglichen die übersichtliche Kontrolle der Moduln.

Bei der Codierung geht es ausschließlich um die Verwirklichung der Planung. Konzeptüberlegungen sind in dieser Phase nicht mehr aktuell.

2.4 Programmtest

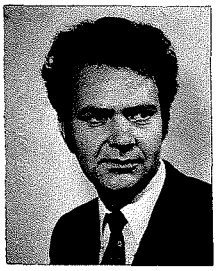
Der Programmtest soll das fehlerfreie Arbeiten des Programms nachweisen. Da Programme beim ersten Testlauf im allgemeinen nicht fehlerfrei sind, ist ein systematisches Vorgehen bei der Fehlersuche sehr wichtig. Es empfiehlt sich, vor dem ersten Start das Programm per Hand genau durchzuspielen. Dann ist das Programm mit einer Reihe ausgewählter Testdaten zu starten, deren Ergebnis bekannt ist. Diese sind so zu variieren, daß jeder Programmzweig mindestens einmal durchlaufen wird. Insbesondere ist auf die richtige Behandlung von Grenzwerten und Fehlersituationen zu achten. Anhand von Ergebnissen und Zwischenergebnissen wird das fehlerhafte oder fehlerfreie Arbeiten des Programms festgestellt. Ein Testrahmen für die Ausgabe zahlreicher Zwischenergebnisse oder ein Testsystem, das die schrittweise Programmausführung überwacht, erleichtert und verkürzt die Fehlersuche. Schwerwiegende Fehler im logischen Konzept führen zurück zur Phase der Programmkonstruktion. Dieser Zyklus wird u. U. mehrmals durchlaufen.

2.5 Dokumentation

Eine gute Dokumentation ist für ein Programm genauso wichtig, wie für jedes andere technische Produkt. Die Programmdokumentation sollte gleichzeitig mit dem Programm entstehen. Sie muß stets auf dem aktuellen Stand sein, andernfalls ist sie wertlos. In manchen Fällen war es schon einfacher ein Programm neu zu schreiben, als ein vorhandenes, schlecht dokumentiertes zu ändern. Die Grundlage der Dokumentation ist ein im Schriftbild übersichtlich gegliederter, durch sachbezogene Namensgebung und Kommentare gut lesbar gestalteter Programmtext. Zur Dokumentation gehören:

- Allgemeine Programmbeschreibung, Leistung des Programms
- Bedienungsanleitung
- Verfahrensbeschreibung
- Modulbeschreibung
- Hinweise auf geplante oder mögliche Ausbaustufen.

Ein Abschnitt sollte für Eintragungen über Verbesserungsvorschläge oder Schwachstellen des Programms reserviert sein. Werden später größere Änderungen aktuell, sind diese bei Bedarf mit zu berücksichtigen.

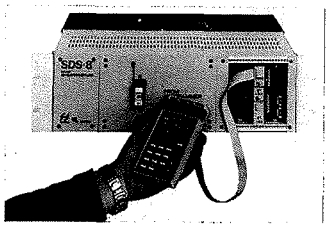


Otmar Feger, geboren in Säckingen, beschäftigte sich vier Jahre mit der Fernsehtechnik bei der Firma Blaupunkt. Danach arbeitete er elf Jahre in Konstanz auf dem Gebiet der Computertechnik, wo er mit der Hardware ebenso befaßt war wie mit der Software. Seit zwei Jahren sind die Mikroprozessoren sein Arbeitsgebiet. Privattelefon: (0 76 57) 6 35

Entwicklungssystem für SC/MP

Unter der Bezeichnung SDS 8 ist von der Firma E. Springorum ein Entwicklungssystem für den Mikroprozessor SC/MP erhältlich, das auch als Mikrocomputer in Systeme für industrielle Steuerungen eingebaut werden kann. Die Grundeinheit enthält neben der CPU die Stromversorgung, ein 1-K-RAM, eine V-24-Schnittstelle und ein Monitorprogramm. Erweiterungskarten enthalten z. B. 4-K-RAM, 4-K-PROM und eine PROM-Programmiereinrichtung. Auch Peripheriegeräte wie Lochkartenleser sind zu diesem System lieferbar.

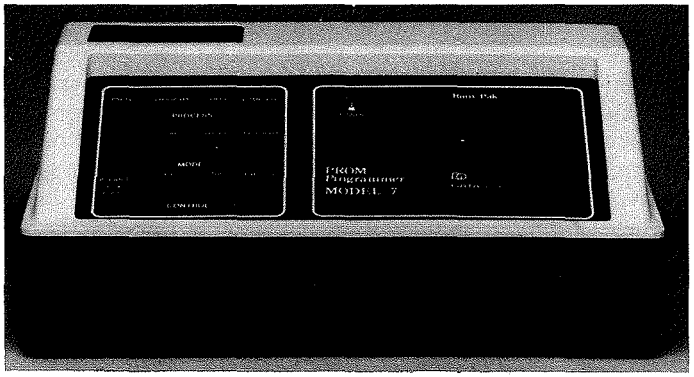
□ Vertrieb: RTG, E. Springorum KG, Bronnerstr. 7, 4600 Dortmund 1, Tel. (02 31) 54 95-1.



PROM-Programmierer mit Fernbedienung

Der mikroprozessorgesteuerte PROM-Duplizierer Modell 7 von der Firma Date I/O ist über eine universelle Schnittstelle (seriell oder parallel) an beliebige Peripheriegeräte anschließbar. Er kann mit einem Datenterminal, einem Computer oder einem Mikroprozessor-Entwicklungssystem per Fernbedienung betrieben werden. In dieser Betriebsart lassen sich Operationen wie „selektive Datenmanipulation und Programmierung", „Zuordnung virtueller Speicherbereiche", „Anwahl verschiedener Datenformate" usw. durchführen. Programmiermoduln für die über 200 kommerziell erhältlichen PROM-Typen, Diodenmatrizen und FPLAs sind lieferbar. Eine hohe Programmierausbeute wird durch einen universellen Kalibrator zur Überprüfung bzw. Nacheichung der Moduln garantiert. Die PROMs werden grundsätzlich über den eingebauten RAM (8...32 K) programmiert, wobei eine automatische Bitsummenprüfung vor und nach dem Programmiervorgang durchgeführt wird. Zusammen mit dem RAM-PAK können die Daten durch eine sogenannte In-Circuit-Emulation überprüft werden. Der für die Manipulation der Daten vorgesehene Romulator, eine kombinierte Tastatur- und Anzeigeeinheit, ermöglicht dabei die Veränderung der Daten. Unter der Modellbezeichnung M 100 ist ein Gerät lieferbar, das Tastatur und Anzeigeeinheit für Daten und Adressen enthält. Der Preis für das Modell 7 beträgt 2950.– DM.

□ Vertrieb: Macrotron GmbH, Cosimastr. 4, 8000 München 81, Tel. (0 89) 91 50 61.



Es ist eine bekannte Tatsache, daß man das Programmieren ebenso wenig ohne praktische Betätigung erlernen kann wie etwa das Klavierspielen. Trotzdem sind natürlich solide theoretische Kenntnisse notwendig. Der folgende Beitrag soll diese Grundlagen vermitteln. Zur Vertiefung der Erkenntnisse und als Anregung zur Lösung ähnlicher Probleme wird die ELEKTRONIK in Zukunft Anwendungsbeispiele veröffentlichen, deren Software detailliert beschrieben ist.

Otmar Feger

Einführung in die Mikrocomputer-Programmierung (I)

Meist beginnen Einführungen in die Programmierung mit der Beschreibung von Zahlensystemen, Rechnerstrukturen und Befehlswirkungen. So wichtig diese Themen auch sind, so wenig eignen sie sich, die Phantasie des Lesers anzuregen. Dies mag mit ein Grund für die verbreitete Ansicht sein, das Programmieren sei eine trockene Wissenschaft. Das Gegenteil ist der Fall: Sieht man von den Grenzen der Rechnergeschwindigkeit und des Speicherplatzes ab, so begrenzen nur Phantasie und Ausdauer des Programmierers das Mögliche. Mit anderen Worten: *Der Mensch wird die Möglichkeiten des Computers nie ausschöpfen!*

Genau wie in der Hardware gibt es in der Programmierung die logischen Grundelemente UND, ODER, NEGATION. Diese und die Möglichkeit, Information zu speichern, bilden die Grundlage der Computertechnik. Wie in der Hardware die Logik unabhängig von der jeweiligen Realisierungsart ist, ist die Programmierlogik unabhängig vom Rechnertyp. Wie es in der Hardware auf die Grundelemente aufbauende Schaltungen, wie Multiplexer, Rechenwerke usw. gibt, verfügt der Programmierer über eine breite Palette von Standard-Programmiertechniken. Ein Programm setzt sich aus einer Folge von Befehlen zusammen. Im Gegensatz zur Hardware mit parallel und/oder seriell ablaufenden Vorgängen arbeitet der Computer die Befehle der Reihe nach (sequentiell) ab. Dies und die Ähnlichkeit des Befehlsvorrates mit einem Wortschatz führte zum Vergleich mit Sprachen. Man spricht von Programmiersprachen.

1 Flußdiagramme

Flußdiagramme bzw. Programmablaufpläne sind grafische Beschreibungsformen von Programmen. Sie ermöglichen die anschauliche und übersichtliche Darstellung komplexer Zusammenhänge. Ein Programm ist durch mehrere Flußdiagramme unterschiedlich detailliert beschreibbar. In der Grobstruktur sind jeweils umfangreiche Teilaufgaben in einem Symbol zusammengefaßt. Ein solches Diagramm kann eine erste Übersicht über ein größeres Programm geben. Die Beschriftung der Symbole ist verbal gehalten. Ein feiner strukturiertes Diagramm bezieht sich mehr auf Programmierungsdetails, wobei einem Symbol meist mehrere Befehle im Programm entsprechen. Bei der Programmkonstruktion und Dokumentation empfiehlt es sich, mit Flußdiagrammen zu arbeiten. DIN 66 001 legt die Sinnbilder fest. Die wichtigsten Symbole zeigt *Bild 1*.

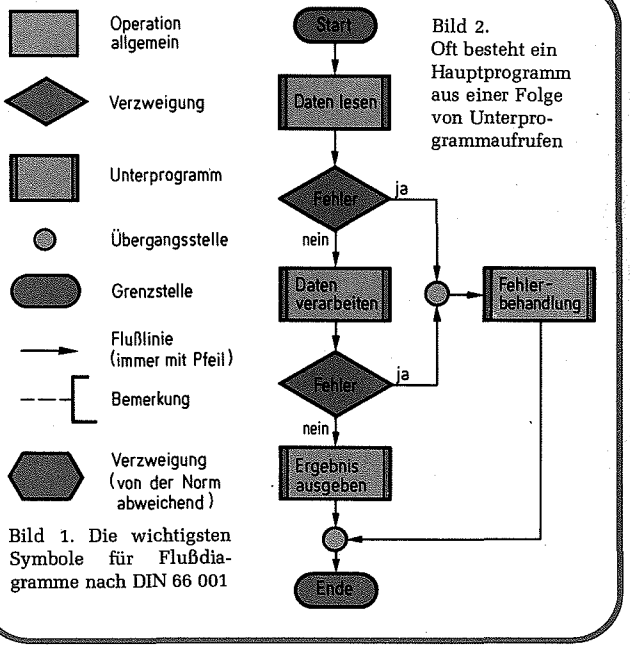


Bild 1. Die wichtigsten Symbole für Flußdiagramme nach DIN 66 001

Bild 2. Oft besteht ein Hauptprogramm aus einer Folge von Unterprogrammaufrufen

Ein Unterprogramm ist ein in sich abgeschlossener Programmteil, der von beliebigen Stellen im Hauptprogramm aufgerufen werden kann. Es gibt zwei Gründe, ein Hauptprogramm als Unterprogramm zu organisieren: 1. Die gleiche Befehlsfolge kommt in einem Programm an mehreren Stellen vor (Beispiele sind Sortier- oder Multiplizierprogramme). 2. Programmteile, die eine Teilaufgabe leisten, aber nur einmal im Programm benötigt werden, sind gelegentlich zweckmäßig als Unterprogramm zu formulieren. Dadurch wird das Hauptprogramm kürzer und übersichtlicher. Oft besteht es nur aus einer Folge von Unterprogrammaufrufen. Ein Beispiel dafür zeigt Bild 2.

Verzweigungen setzen einen Vergleich voraus. Beispielsweise ob eine Zahl größer, kleiner oder gleich Null oder mit einer anderen Zahl identisch ist. Je nachdem, ob die Bedingung erfüllt ist oder nicht, fährt der Computer mit der folgenden Anweisung fort oder verzweigt zu einem anderen Programmteil, d. h., er überspringt einen Teil des Programms. Springt er zurück, kann der schon durchlaufene Programmteil wiederholt durchlaufen werden.

1.1 Programmbeispiele

Es soll aus den Zahlen a, b, c die größte ermittelt und in D reserviert werden. *Bild 3* zeigt das Diagramm für eine mögliche Lösung. In diesem Beispiel läßt sich die richtige Arbeitsweise des Verfahrens leicht nachprüfen. Die Daten a, b, c können in sechs verschiedenen Kombinationen auftreten, die der Reihe nach durchzuspielen sind. Bei komplexeren Aufgaben begnügt man sich mit Testbeispielen, die das Nachvollziehen jedes Programmzweiges gestatten. Es sei dem Leser empfohlen, das Beispiel nachzuprüfen. An diesem Beispiel fällt auf, daß es außer den Variablen a, b, c auch den Platz D gibt. Der Pfeil bezeichnet den Transport einer Variablen zu einem Speicherplatz, in dem sie dann jederzeit verfügbar ist.

Oft befinden sich zu verarbeitende Daten in Listenform im Speicher. Das Programm muß dann auf bestimmte Teile der Liste zugreifen oder sie durchsuchen können. Im Beispiel von *Bild 4* wird aus dem Inhalt der vorgegebenen Liste der größte Wert herausgesucht und in „D" abgelegt. „D" bezeichnet einen Platz. ⟨D⟩ bezeichnet den Inhalt von „D". Die Schreibweise ⟨D⟩ + 1→D bedeutet, daß der Inhalt von „D" um eins erhöht wieder in „D" abgelegt wird. Ein Speicherplatz wird durch seine Adresse eindeutig bezeichnet. „D" ist in diesem Fall eine symbolische Adresse.

Um nun eine Liste im Speicher, die in aufeinanderfolgenden Adressen abgelegt ist, abzuarbeiten, muß das Programm sich die jeweils erforderlichen Adressen errechnen. Das Programm arbeitet wie folgt: Nach dem Start wird die Anfangsadresse der Liste „LIA" in der Speicherzelle „X" abgelegt. Der Inhalt der Speicherzelle „D" wird gelöscht. Als nächstes vergleicht der Computer den Inhalt vom Inhalt von X (dies wird durch ⟨⟨X⟩⟩ dargestellt) mit dem Inhalt von „D". ⟨⟨X⟩⟩ ist der Inhalt des ersten Listenelements. ⟨X⟩ ist die Adresse des ersten Listenelements, und diese Adresse steht in „X". Durch geeignete Befehle kann der Computer über die Adresse „X" auf das entsprechende Listenelement zugreifen. Ist ⟨⟨X⟩⟩, der Inhalt des ersten Listenelementes, größer als ⟨D⟩, so wird dieser Wert in „D" abgelegt. Danach folgt die Abfrage, ob ⟨X⟩ identisch mit der Adresse des letzten Listenelements „LIE" ist. Da dies nach dem ersten Durchlauf noch nicht der Fall ist, wird ⟨X⟩ um eins erhöht. In „X" steht jetzt die Adresse des folgenden Listenelementes. Der Zyklus wiederholt sich nun, bis die Liste abgearbeitet ist. In „D" steht dann der größte Listenwert. Das Beispiel wurde so ausführlich beschrieben, weil das Rechnen mit Adressen und die Schleifentechnik anfangs schwer zu durchschauen sind.

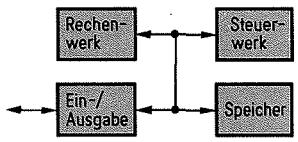
Das in *Bild 5* dargestellte Beispiel sortiert den Listeninhalt in aufsteigender Reihenfolge. Für die Adressenrechnung benötigt das Verfahren zwei Hilfszellen „X" und „Y". Das Programm vergleicht jedes Listenelement mit jedem anderen und tauscht sie erforderlichenfalls. Der Leser möge mit einem kleinen Beispiel das Programm testen.

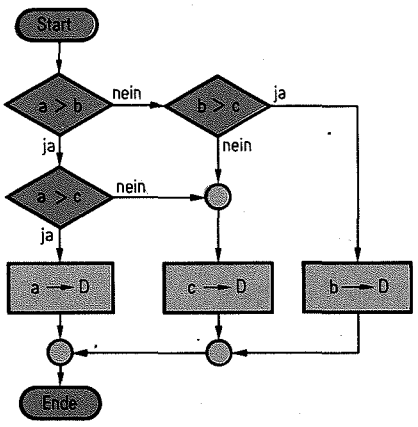
Bild 6 zeigt abschließend die Funktionen UND, ODER und EXKLUSIV-ODER als Flußdiagramm.

2 Befehlstypen

Um die Wirkung und Funktion der Befehle besser zu verstehen, sei kurz auf die Computerstruktur (*Bild 7*) eingegangen. Der Computer besteht aus den vier Teilen Ein/Ausgabe-Einheit, Speicher, Rechenwerk und Steuerwerk. Bei den Ein-Chip-Computern befinden sich alle vier Teile auf einem Chip. Ein/Ausgabe-Einheit und Speicher sind durch zusätzliche Bausteine erweiterbar. Die weit verbreiteten Mikroprozessoren 8080, 6800 und SC/MP enthalten das Rechen- und Steuerwerk und die unmittelbare Ein/Ausgabe-Schaltung im beschränkten Umfang auf demselben Chip. Letztere stellt die Verbindung des Systems mit der Außenwelt her. Es gibt zahlreiche Peripheriegeräte zur Ein- und Ausgabe von Daten und Programmen. Der Speicher dient zur Aufnahme von Programmen und Daten. Steuerwerk und Rechenwerk greifen unmittelbar auf den Speicher zu. Der Speicher besteht aus einer fortlaufend durchnumerierten Folge von Speicherworten. Ein „Wort" ist die über eine Nummer zu erreichende Informationseinheit. Die Wortlänge beträgt bei den genannten Mikroprozessoren 8 bit (= 1 Byte). Die jeder Speicherzelle eindeutig zugeordnete Nummer heißt Adresse. Über sie kann der Inhalt der betreffenden Zahl gelesen oder überschrieben werden. Ob in einer Zelle ein Befehl oder ein Datum steht, ist aus dem Inhalt der Zelle nicht ersichtlich. Dies ist Sache der Interpretation des Speicherinhaltes. D. h., dem Computer ist mitzuteilen, wo ein Programm steht.

Bild 7. Blockschaltung eines Computers





◀ Bild 3. Das Programm ermittelt die größte von drei Zahlen und speichert sie in „D"

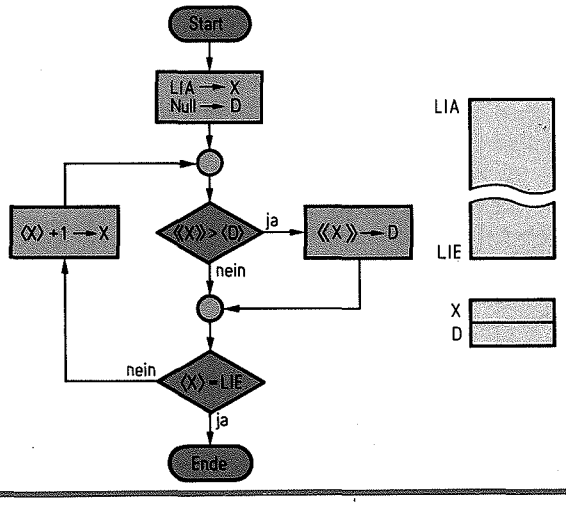


Bild 4. ▶ Das Programm ermittelt den größten Wert aus einer beliebig langen Liste

Im Rechenwerk führt der Computer arithmetische und logische Operationen aus. Es besteht aus einem oder mehreren Registern, den Akkumulatoren. Die Verarbeitungsbreite des Rechenwerks entspricht normalerweise der Wortlänge eines Speicherwortes. Das Steuerwerk steuert die Abläufe im Rechner und das Zusammenspiel der verschiedenen Computerteile in der vom Programm vorgeschriebenen Weise. Die Befehle eines Programms stehen in aufeinanderfolgender Reihenfolge im Speicher. Ein Register im Steuerwerk, der Befehls- oder Programmzähler, enthält die Adresse des gerade adressierten Befehls. Nach jeder Befehlsausführung erhöht sich der Befehlszähler um Eins. Über diese Adresse greift das Steuerwerk auf die Befehle des Programms zu.

Der beim Programmablauf feststehende Zyklus pro Befehl besteht darin, den gerade aktuellen Befehl aus dem Speicher in das Steuerwerk zu transportieren, ihn dort zu erkennen bzw. zu entschlüsseln und die Anweisung auszuführen. Ein Befehl enthält immer einen Operationsteil. Dieser gibt Aufschluß über das, was der Computer zu tun hat. Die meisten Befehle beziehen sich auf den Speicher. Diese enthalten als zweite Angabe einen Adreßteil. Bei der Verknüpfung zweier Operanden, z. B. einer Addition, steht der eine Operand im Akkumulator, der zweite Operand steht unter der im Befehl bezeichneten Adresse im Speicher. Die Befehle der erwähnten Mikroprozessoren sind ein, zwei oder drei Byte lang.

Es gibt auf der Maschinenebene folgende Befehlstypen:

Transportbefehle: Sie transportieren Daten zwischen Rechenwerk, Speicher und Ein/Ausgabe-Einheit.

Setz- und Löschbefehle: Es können Inhalte von Registern oder Speicherzellen gesetzt bzw. gelöscht werden.

Schiebebefehle: Sie dienen zum Verschieben der Information innerhalb eines Bytes im Akkumulator oder in einer Speicherzelle. Bei Mikroprozessoren wird jeweils um eine Bitstelle nach rechts, links oder im Kreis geschoben.

Arithmetische Befehle: Bei den Mikroprozessoren sind dies im wesentlichen der Additions- und der Subtraktionsbefehl. „Höhere" Funktionen lassen sich damit aufbauen. Gelegentlich gibt es Befehle zum bequemen Rechnen im Dezimalmodus.

Logische Befehle: Es gibt die Befehle UND, ODER und bei manchen Mikroprozessoren EXKLUSIV-ODER und NEGATION. Sie verknüpfen den Inhalt einer Speicherzelle bitweise mit dem Inhalt eines Akkumulators.

Vergleichsbefehle: Diese Befehle setzen entsprechend dem Vergleichsergebnis verschiedene Bits im Bedingungsregister, ohne die Operanden zu verändern. Sie sind oft Voraussetzung für bedingte Sprungbefehle.

Sprungbefehle: Die Sprungbefehle bilden die Grundlage für einige wesentliche Fähigkeiten des Computers, wie das wiederholbare Ausführen eines Programmteils und das Verzweigen in verschiedene Programmteile, abhängig von Entscheidungen. Das automatische Hochzählen des Befehlszählers nach jeder Befehlsausführung steuert den linearen Programmablauf. Sprungbefehle laden den Befehlszähler mit einer neuen Adresse, wodurch das Programm an dieser Stelle fortfährt. Es gibt unbedingte und bedingte Sprungbefehle. Die unbedingten Sprünge führen in jedem Falle zur Fortsetzung auf der im Adreßteil angegebenen Adresse. Die bedingten Sprünge verzweigen nur, wenn eine bestimmte Bedingung erfüllt ist. Andernfalls fährt das Programm mit dem folgenden Befehl fort.

Ein spezieller Sprungbefehl ist der Unterprogrammsprung. Unterprogramme sind in sich geschlossene Programmteile, die von beliebigen Stellen des Hauptprogramms aus aufgerufen werden können. Der Computer sorgt nach dem Durchlaufen des Unterprogramms für den Rücksprung auf die Adresse, die dem Unterprogramm-Sprungbefehl folgt.

3 Adressierungsarten

Das Rechnen mit Adressen spielt beim Programmieren auf der Maschinenebene eine große Rolle. In großen Computern wurden dafür schon früh kleine Spezialrechenwerke einge-

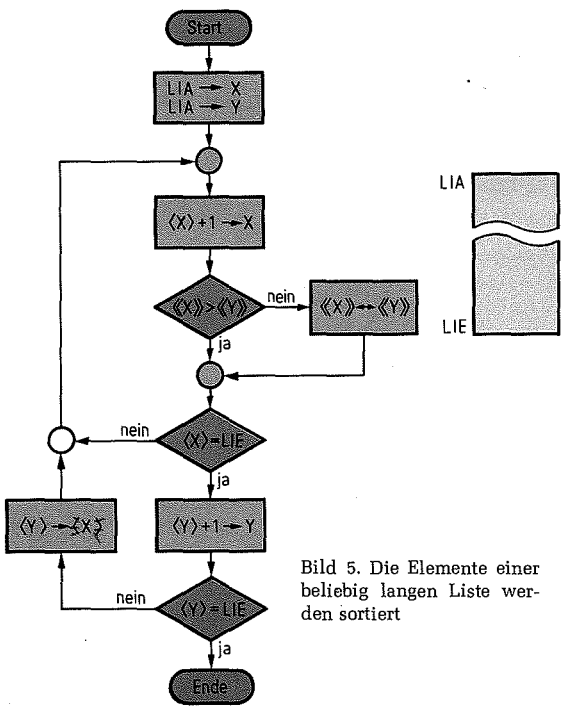


Bild 5. Die Elemente einer beliebig langen Liste werden sortiert

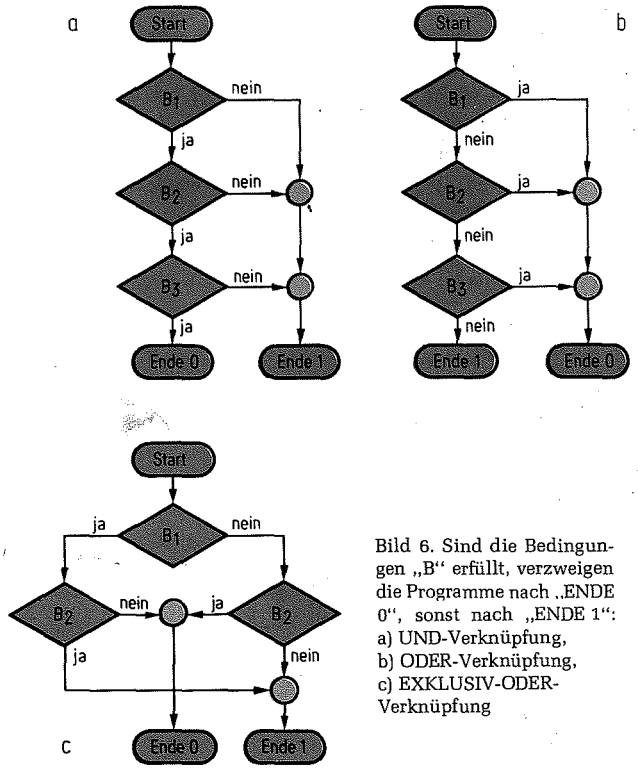


Bild 6. Sind die Bedingungen „B" erfüllt, verzweigen die Programme nach „ENDE 0", sonst nach „ENDE 1":
a) UND-Verknüpfung,
b) ODER-Verknüpfung,
c) EXKLUSIV-ODER-Verknüpfung

baut. Aus ihnen entwickelten sich die Mikroprozessoren. Um Adreßrechenoperationen zu erleichtern, gibt es ein oder mehrere Indexregister und Befehle zum Arbeiten mit diesen Registern. Viele Befehle lassen sich mit mehreren Adressierungsarten [1, 4] verwenden. Die Adressierungsart bestimmt den Zugriff zum Operanden.

Die grundsätzlichen Möglichkeiten sind:

- *Implizite Adressierung*
  Diese Adressierungsart bezieht sich auf Register. Die Registerbezeichnung steht unmittelbar im Operationscode, weshalb diese Befehle keinen Adreßteil aufweisen (beispielsweise Schiebe- und Registertausch-Befehle).
- *Unmittelbare Adressierung*
  In diesem Adressierungsmodus wird der Inhalt des Adreßteils nicht als Adresse, sondern selbst als Operand verwendet (*Bild 8a*). Beispielsweise steht nach der Befehlsausführung „Lade Akkumulator" (LDA) der Inhalt des Adreßteils selbst im Akkumulator.
- *Direkte und erweiterte Adressierung*
  Mit der direkten Adressierung kann auf Operanden im Speicherbereich von 0...255 zugegriffen werden. Der Vorteil liegt im geringeren Platzbedarf des Befehls. Der Adreßteil umfaßt ein Byte. Bei der erweiterten Adressierung besteht der Adreßteil aus 2 Byte. Damit läßt sich ein Adressenraum von 64K bearbeiten. Manche Mikroprozessor-Typen verwenden nur die erweiterte Adressierung. *Bild 8b* zeigt ein Beispiel. Der Befehlszähler steht auf der Adresse 50. Bei der Ausführung des Befehls wird der Inhalt der Adresse 71 in den Akkumulator geladen.
- *Indizierte Adressierung*
  Bei der indizierten Adressierung wird vor der Befehlsausführung zur Adresse im auszuführenden Befehl der Inhalt des Indexregisters addiert. Im Beispiel *Bild 8c* wird deshalb der Operand nicht aus der Speicherzelle 70 sondern aus 72 geholt. Diese Adressierungsart ist besonders beim Durchsuchen oder Löschen von Speicherbereichen oder beim Modifizieren von Sprungbefehlen usw. bequem, da der Inhalt des Indexregisters herauf oder heruntergezählt werden kann und sich dadurch die Befehle innerhalb einer Schleife nicht ändern müssen.

  Ein Beispiel soll dies verdeutlichen:
  Es soll der Speicherbereich 80...100 gelöscht werden (siehe *Programmliste*). Zuerst wird das Indexregister mit einem unmittelbaren (I = *immediate*) Befehl mit 20 geladen. Dann wird mit dem indizierten (X = *indexed*) Löschbefehl CLR mit der Adresse 80 der Inhalt der Adresse 100 gelöscht (20+80=100). Der folgende Befehl DEX (dekrementiere Indexregister) zieht vom Inhalt des Indexregisters Eins ab. Der Sprungbefehl BNE springt um zwei Befehle solange zurück, bis das Ergebnis der Subtraktion im Indexregister Null ergibt. Danach sind die 20 Speicherzellen gelöscht, und das Programm verläßt die Schleife.

Programmliste für das Löschen des Speicherbereichs 80...100

Operationsteil	Adressierungsart *	Adreßteil
LDX	I	20
CLR	X	80
DEX		
BNE		–2

* Die Adressierungsart wird nur dann angegeben, wenn für einen Befehl verschiedene möglich sind.

- *Relative Adressierung*
  Manche Mikroprozessoren gestatten relative Adressen bei Sprungbefehlen. Dabei steht im Adreßteil des Befehls die Sprungweite zum Sprungziel und nicht die Adresse des Sprungziels selbst. Der Vorteil liegt in der Befehlslänge (*Bild 8d*). Der Adreßteil ist auf ein Byte beschränkt, was eine maximale Sprungweite von +128 und –127 ergibt. Das erste Bit im Adreßteil wird als Vorzeichen gewertet.

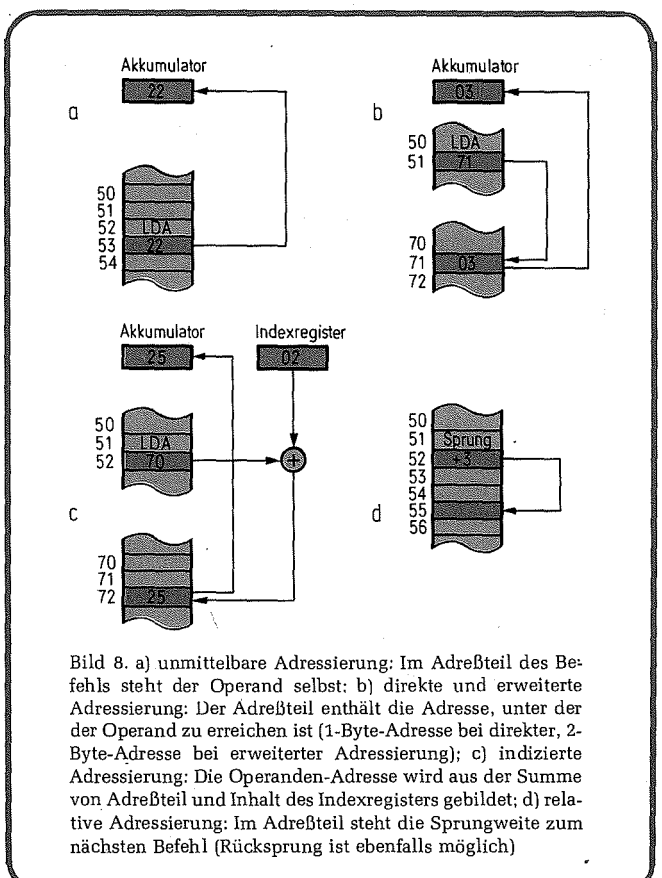


Bild 8. a) unmittelbare Adressierung: Im Adreßteil des Befehls steht der Operand selbst; b) direkte und erweiterte Adressierung: Der Adreßteil enthält die Adresse, unter der der Operand zu erreichen ist (1-Byte-Adresse bei direkter, 2-Byte-Adresse bei erweiterter Adressierung); c) indizierte Adressierung: Die Operanden-Adresse wird aus der Summe von Adreßteil und Inhalt des Indexregisters gebildet; d) relative Adressierung: Im Adreßteil steht die Sprungweite zum nächsten Befehl (Rücksprung ist ebenfalls möglich)

4 Programmierungstechniken

4.1 *Bitmanipulationen*

Mit den logischen Befehlen lassen sich leicht Binärstellen innerhalb eines Wortes manipulieren, d. h. ausblenden und einblenden. Im *Bild 9a* blendet der UND-Befehl mit der im Speicher stehenden Maske drei Binärstellen aus der im Akkumulator stehenden Information aus. Der ODER-Befehl (*Bild 9b*) fügt die in der Maske gesetzten Bits in das im Akkumulator stehende Wort ein. Durch Kombinieren beider Möglichkeiten lassen sich beliebige Informationen zusammenmischen.

4.2 *Verteiler*

Muß ein Programm nach mehr als zwei Stellen verzweigen, spricht man von Verteilern. Die Strukturdarstellung und eine Realisierungsmöglichkeit zeigt *Bild 10*. Ein Verteiler ist immer mit einer Größe verbunden, die verschiedene Werte annehmen kann. Er verzweigt bei jedem Wert zu einem bestimmten Programmteil. Für den Fall, daß dem Verteiler ein unzulässiger Wert angeboten wird, soll ein Fehlerausgang vorgesehen sein. Ein einfaches, verbal beschriebenes Programmbeispiel soll zur Verdeutlichung dienen: Der

Computer soll nach einer manuellen Eingabe eine Zufallszahl (1...6) grafisch ausgeben. Die manuelle Eingabe erzeugt einen Interrupt, d. h., das laufende Programm wird unterbrochen, und es wird ein Interruptprogramm gestartet. Nach dem Rücksprung aus dem Interruptprogramm gibt der Computer die Regie an das Hauptprogramm zurück. Der Hauptteil des „Würfelprogramms" besteht aus einer Zählerschleife. Im Indexregister wird fortlaufend von 1...6 gezählt. Wird ein Interrupt wirksam, so steht die Zufallszahl im Indexregister. Das Interruptprogramm führt nun einen indizierten Sprungbefehl auf eine Tabelle aus. In dieser Tabelle stehen wiederum Sprungbefehle, die zu den entsprechenden Ausgabeprogrammteilen führen.

4.3 Schleifentechnik

Programmschleifen dienen der wiederholten Ausführung eines Programmteiles. Sie bestehen aus vier Komponenten:

a) Initialisierung (Schleifenkriterium)
b) Bearbeitung
c) Schleifenkriterium verändern
d) Endabfrage.

Gelegentlich kann die Bearbeitung unmittelbar das Schleifenkriterium ergeben. *Bild 11* zeigt das Schema. Ist die Zahl der Schleifendurchläufe immer größer Null, kann die Endabfrage nach der Veränderungsphase erfolgen. Für die gleichartige Bearbeitung zusammenhängender Speicherbereiche werden meist indizierte Zählschleifen verwendet. Der Zähler ist dann Index zur fortlaufenden Adressierung des Bereichs.

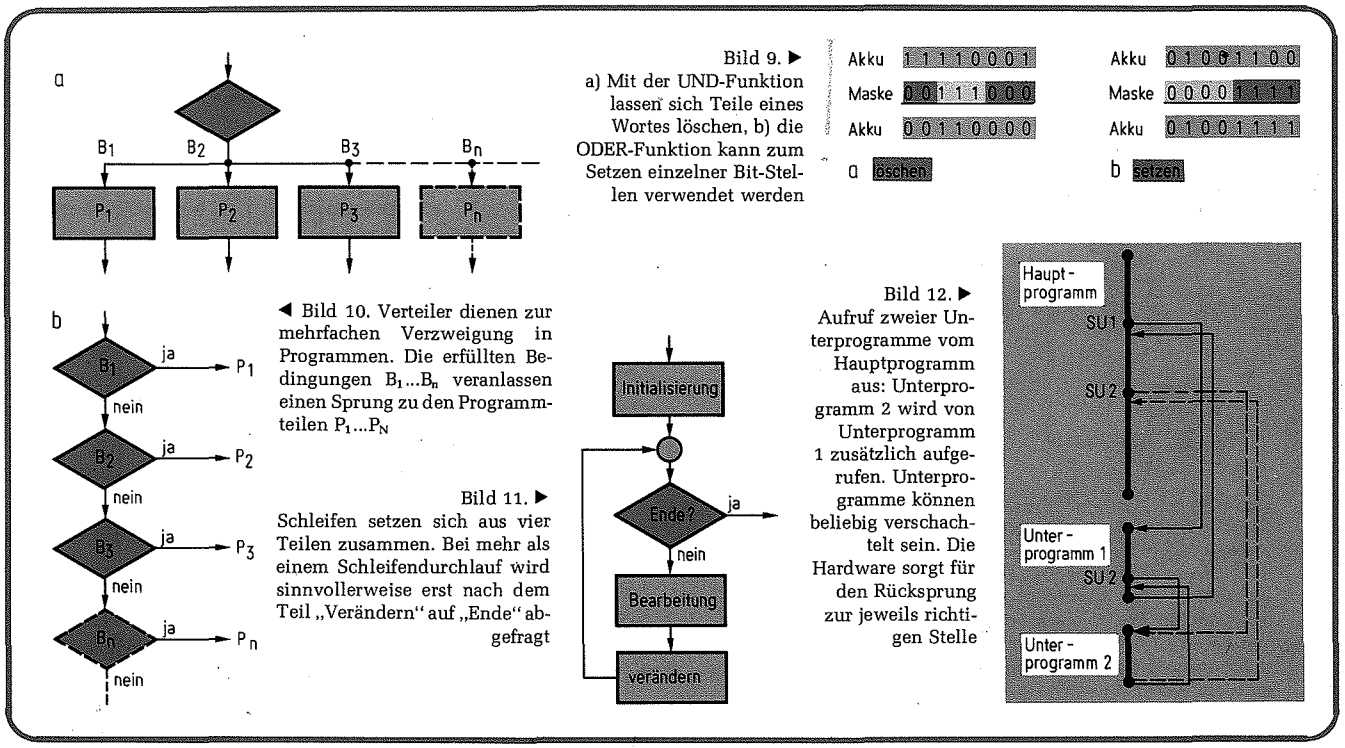
4.4 Unterprogramme

Kommt die gleiche oder ähnliche Befehlsfolge in einem Programm an mehreren Stellen vor, so kann sie als Unterprogramm organisiert werden. Auch die Lösung von Standardaufgaben, die in verschiedenen Programmen einsetzbar sind, stellt man als Unterprogramme zur Verfügung. Ein Unterprogramm kann von beliebiger Stelle im Hauptprogramm aufgerufen werden. Ein spezieller Sprungbefehl veranlaßt den Computer, die Rücksprungadresse sicherzustellen. Nach Abarbeitung des Unterprogramms erfolgt der Rücksprung auf diese Adresse.

Unterprogramme können ihrerseits weitere Unterprogramme aufrufen usw. Der Computer muß deshalb über einen Stapelspeicher (*Stack*) die verschiedenen Rücksprungadressen so verwalten, daß beim jeweiligen Rücksprung die richtige Adresse zur Verfügung steht. In *Bild 12* ist ein möglicher Ablauf dargestellt. Vom Hauptprogramm aus wird zuerst das Unterprogramm U1 mit SU1 angesprungen. Innerhalb von U1 folgt ein Sprung auf das Unterprogramm U2. Später wird U2 nochmals vom Hauptprogramm mit SU2 aufgerufen.

Ändern sich in Unterprogrammen von Aufruf zu Aufruf bestimmte Größen, z. B. Speicheradressen für Ergebnisse oder Berechnungskonstanten, werden diese als Parameter dem Unterprogramm übergeben. Für die Parameterübergabe gibt es verschiedene Möglichkeiten.

1. Es werden bestimmte Adressen des Speichers vereinbart. Das Hauptprogramm legt die Parameter dort ab und das Unterprogramm greift darauf zu. Diese einfache Form schränkt die allgemeine Verwendung von Unterprogrammen ein.
2. Bei wenigen Parametern sind diese in den Registern des Rechenwerkes zu übergeben.
3. Ist die Parameterzahl für die Registerübergabe zu groß, genügt die Angabe der Anfangsadresse eines Speicherbereiches, in dem die Parameter in vereinbarter Reihenfolge stehen.
4. Im Stapelspeicher können ebenfalls Parameter an Unterprogramme weitergereicht werden. Normalerweise gibt es zur Handhabung des Stapelspeichers besondere Befehle (PUSH, POP).
5. Früher wurden die Parameter oft unmittelbar in die Speicherzellen nach dem Programmaufruf gelegt. Dies führt jedoch zur Vermengung von Daten- und Programmbereichen und läßt sich auch nicht immer in Festwertspeichern (ROMs) realisieren.



◄ Bild 10. Verteiler dienen zur mehrfachen Verzweigung in Programmen. Die erfüllten Bedingungen $B_1...B_n$ veranlassen einen Sprung zu den Programmteilen $P_1...P_N$

Bild 11. ► Schleifen setzen sich aus vier Teilen zusammen. Bei mehr als einem Schleifendurchlauf wird sinnvollerweise erst nach dem Teil „Verändern" auf „Ende" abgefragt

Bild 9. ► a) Mit der UND-Funktion lassen sich Teile eines Wortes löschen, b) die ODER-Funktion kann zum Setzen einzelner Bit-Stellen verwendet werden

Bild 12. ► Aufruf zweier Unterprogramme vom Hauptprogramm aus: Unterprogramm 2 wird von Unterprogramm 1 zusätzlich aufgerufen. Unterprogramme können beliebig verschachtelt sein. Die Hardware sorgt für den Rücksprung zur jeweils richtigen Stelle

5 Programmiersprachen

Das sequentielle Abarbeiten eines Programmes und der an einen Wortschatz erinnernde Befehlsvorrat führte zum Vergleich mit Sprachen. Ein Programm kann durch einen Namen symbolisch bezeichnet werden. Dieser Name läßt sich auch als Befehl ansehen, hinter dem die spezifische Leistung des Programmes steht. Tatsächlich gibt es oft in Computern, auch Mikrocomputern, Befehle, die ihrerseits durch interne Programme realisiert sind. Solche Befehle können höhere mathematische Funktionen, Tabellensuchoperationen, Operationen zum Retten von Registerinhalten usw. sein.

Diese Eigenschaft ermöglicht es, auf jedem Computer neue Befehle mit beliebigen Eigenschaften zu erfinden bzw. zu definieren. Entwickelt man für eine Problemkategorie besonders geeignete Befehle, führt dies zur Bildung problemorientierter Sprachen. Für die Kategorie Mathematik wurden u. a. FORTRAN und ALGOL, für kommerzielle Zwecke COBOL und für Simulationen SIMULA entwickelt. Es gibt eine Vielzahl weiterer standardisierter Sprachen.

Die Einführung dieser Sprachen brachte den Vorteil der Unabhängigkeit von Computertypen, vorausgesetzt die Sprachbegriffe waren in die jeweils benötigte Maschinensprache zu übersetzen. Beispielsweise muß der mathematische Begriff „Wurzel" für jeden Computertyp durch ein anderes Programm formuliert werden.

Die höheren Programmsprachen versuchen, sich möglichst an die Umgangssprache anzulehnen. Dadurch soll erreicht werden, daß sich Programme leicht verständlich selbst beschreiben und daß der Programmierer sich nicht um Rechnerstrukturen kümmern muß. Beispielsweise lassen sich mathematische Formeln unmittelbar niederschreiben. Der Vorteil der bequemen Handhabung und der Unabhängigkeit vom Computertyp wird durch Standardisierung erreicht. Der Nachteil standardisierter Sprachen liegt in einer gewissen Starrheit gegenüber Weiterentwicklungen und darin, daß besondere Leistungen eines Computertyps oft nicht ausgenutzt werden. Programme, die Zeit- oder Speicherplatz-kritisch sind, müssen u. U. ganz oder teilweise in der Maschinensprache geschrieben werden. Ist ein Programm in einer problemorientierten Sprache geschrieben, so übersetzt es der Compiler in den Maschinencode. Der Compiler ist ebenfalls ein Programm, das für jeden Computertyp vorgegeben sein muß.

Auch auf der Maschinenebene läßt sich mit Hilfe des Computers das Programmieren erleichtern. Die bitweise Anordnung von Programmen oder Daten im Computer wird bei der Ein- und Ausgabe oft im Oktal- oder Sedezimalsystem dargestellt. Ein weiterer Schritt ist die Verschlüsselung des Befehlscodes in mnemotechnische Ausdrücke. Der Befehl „Addiere" wird dann beispielsweise durch „A" ersetzt. Die Zahlen werden dezimal und Texte werden in Klarschrift dargestellt.

Der Assembler, ein Programm, das dem Compiler sehr ähnlich, jedoch einfacher ist, übersetzt diese Schreibweise in den Maschinencode. Im Gegensatz zum Compiler entspricht dabei jeder Externbefehl einem Internbefehl. Nach der Übersetzung ist das Programm betriebsbereit.

Eine weitere Vereinfachung bringt die Einführung von symbolischen Adressen. Bei einem lauffähigen Programm beziehen sich bestimmte Befehle auf Adressen im eigenen Programm (Sprungbefehle) und auf Adressen des Datenspeichers. Da sich in der Entstehungsphase eines Programmes die Adressenzuordnung oft ändert, gelegentlich auch noch gar nicht möglich ist, erfolgt diese beim Assemblieren. Zuvor verwendet man symbolische Adressen (Namen). Beispiel: Speichere nach „Ergebnis". Der Assembler reserviert sich automatisch einen Speicherort für „Ergebnis" und setzt dann die absolute Adresse in den Befehl ein. Der Befehl springe nach „Teil 1" wird entsprechend interpretiert. Die Bezeichnung „Teil 1" muß sich vor dem Befehl befinden, auf den gesprungen wird.

6 Hinweise für den Einstieg

Von wenigen Ausnahmen abgesehen, sind die verschiedenen Mikroprozessor-Typen nicht softwarekompatibel, d. h., ein für den Typ 6800 geschriebenes Programm läuft nicht auf dem Typ 8080 und umgekehrt. Der Grund liegt im unterschiedlichen Hardware-Aufbau, aus dem verschiedene, wenn auch ähnliche Befehlssätze folgen. Andererseits gibt es zwischen den einzelnen Mikroprozessoren und auch zwischen ihnen und großen Rechenanlagen keine prinzipiellen Unterschiede. Die Unterschiede beziehen sich auf die Zahl und den Komfort der Befehle, auf die parallel zu verarbeitende Wortlänge, die Schnelligkeit, die Peripherie-Anschlußmöglichkeit usw. Leider gibt es bei den Mikroprozessor-Herstellern oft unterschiedliche Bezeichnungen für Befehlstypen und Adressierungsarten bei gleicher Funktion. Da die Kurzbezeichnungen der Befehle Abkürzungen der englischen Namen sind und so auch von den Assemblern verstanden werden, ist keine Normung möglich.

Besonders beim Erlernen des Programmierens ist die unmittelbare Kontrolle durch den Computer wichtig, um einen guten Wirkungsgrad erzielen zu können. Deshalb sollte ein entsprechendes Computersystem mit geeigneter Einführungsliteratur zur Verfügung stehen. Es gibt bis jetzt wenig auf Mikroprozessoren zugeschnittene Software-Einführungen, jedoch viele Veröffentlichungen über das Programmieren größerer Computersysteme. Wer auf einen Computer zugreifen kann und die geeignete Literatur dazu bekommt, sollte unbedingt mit diesem System das Programmieren erlernen. Dies ist auch dann zu empfehlen, wenn später andere Computer zu programmieren sind, für die keine Einführungsliteratur zur Verfügung steht. Der Vorteil, sich an einem Computer einzuarbeiten, rechtfertigt den relativ geringen Aufwand des späteren Umstiegs auf andere Systeme.

Obwohl Mikrocomputer inzwischen für jedermann erschwinglich sind, ist die erforderliche Peripherie meist teuer. Sie kommt im allgemeinen aus dem Bereich größerer Anlagen. Dort bewegen sich die Preise noch in entsprechenden Größenordnungen. Ebenfalls aufwendig sind die von den Mikroprozessor-Herstellern angebotenen Programmentwicklungssysteme. Sie ermöglichen komfortables Programmieren und Austesten, sie setzen aber oft eine größere Rechenanlage voraus. Inzwischen sind preiswerte, arbeitsfähige Mikrocomputersysteme auf dem Markt, die sich für die Einarbeitung in die Hardware oder in die Software eignen. Systeme mit Schaltereingabe und LED-Anzeige-Einheiten auf der Bitebene sind für die Programmierausbildung nicht geeignet.

Literatur

[1] *Tireford, H.*: Die Adressierungsarten bei Mikroprozessoren. ELEKTRONIK 1976, H. 12, S. 49...53.

[2] *Koch, G.*: Stand und Trends der Programmierung von Mikroprozessoren. ELEKTRONIK 1977, H. 1, S. 63...66 und H. 2, S. 66...71.

[3] *Gößler, R.* und *Schwerte, J.*: Auf dem Weg zur Mikroprozessor-Praxis. ELEKTRONIK 1975, H. 3, S. 74...86.

[4] *Scheytt, W.*: Minicomputer. ELEKTRONIK 1975, H. 10, S. 83...90.

Dipl.-Ing. Reinhard Gößler

Einführung in die Mikrocomputer-Programmierung (II)

Der nachfolgende Aufsatz ist der zweite einer Beitragsreihe, die die Grundlagen für die Programmierung eines Mikrocomputers liefert. Diese Beschreibung legt einen voll ausgebauten Mikrocomputer zugrunde, wie er in dieser Form von einer Vertriebsfirma angeboten wird, und dessen Aufbau speziell an die Erfordernisse bei der Einarbeitung und Schulung angepaßt ist. Auch das fest in diesem Computer abgespeicherte Betriebsprogramm wurde eigens für Ausbildungszwecke geschrieben, und mit freundlicher Genehmigung der amerikanischen Lehrfirma Integrated Computer Systems bildet es unsere Arbeitsgrundlage. Damit erhält der Leser die Möglichkeit, sich im Selbststudium bis ins Detail in die Mikrocomputer-Technik einzuarbeiten und darauf aufbauend eigene Anwendungen zu realisieren. Aber auch für eine erste theoretische Beschäftigung mit der Materie ist unsere stets an der Praxis orientierte Aufsatzreihe besonders geeignet.

1 Konzeption

Eine fundierte Einführung in die Programmierung kann nur am Beispiel eines realen Mikroprozessors erfolgen. Aus dem breitgefächerten Marktangebot von derzeit rund 50 verschiedenen Mikroprozessoren [1] wurde dazu der Standardtyp 8080 ausgewählt, obwohl es inzwischen leistungsfähigere Nachfolgemodelle gibt, wenn man nur an die Typen 8085 oder Z 80 denkt [2, 3]. Ausschlaggebend für die Wahl des 8080 waren dessen weite Verbreitung, seine beispielhafte Architektur und der gleichzeitig angebotene Hardware-Aufbau mit leistungsfähigem Betriebsprogramm [4]. Natürlich ist es nach der Einarbeitung auf diesem Prozessor problemlos möglich, mit einem anderen Modell weiterzuarbeiten; die damit verbundene Umstellung ist mit derjenigen vergleichbar, die beim Kauf eines bestimmten Autotyps auftritt, nachdem man auf einem anderen Modell die Fahrschule absolviert hat.

Die Programmierung erfolgt auf der untersten Ebene in *Maschinensprache*; dazu ist nicht nur der geringste Aufwand erforderlich, sondern diese Form ist auch noch mit einem wesentlich größeren Lerneffekt verbunden als es beim unmittelbaren Einsatz eines Assemblerprogramms der Fall wäre [5]. Das liegt an der engen Verflechtung von Hard- und Software, die beim Programmieren in Maschinensprache und dem Assemblieren von Hand besonders leicht durchschaubar wird.

2 Die MTS-Hardware

Das Mikrocomputer-Trainings-System (MTS) der Firma ICS ist ein voll ausgebauter Mikrocomputer in seiner Minimalkonfiguration (*Bild 1*). Außer den Grundbestandteilen eines Mikrocomputers [6] enthält dieser Aufbau eine achtstellige Anzeige und eine Tastatur, um die Kommunikation zwischen Mensch und Maschine zu ermöglichen. Die Blockschaltung des Systems (*Bild 2*) wird später an verschiedenen Punkten im Detail aufgelöst, um die Auswirkung bestimmter Befehlsfolgen auf die Hardware zu veranschaulichen. Die vollständige Detailschaltung ist beispielsweise aus [7] zu ersehen.

Das Vertrautwerden mit der Hardware ist die unerläßliche Voraussetzung für den Einstieg in die Programmierung, weil die Hardware-Eigenschaften in wesentlichem Maße vom Programm beeinflußt werden und umgekehrt.

Aus Bild 1 geht recht eindrucksvoll hervor, welch geringen Umfang der Mikroprozessor selbst innerhalb des Mikrocomputers einnimmt; ohne die Vielzahl der ergänzenden Hilfsbausteine wäre das System überhaupt nicht arbeitsfähig, und dieser Aufbau kann als Beispiel dafür dienen, daß der Aufwand für den Mikroprozessor vielfach zu Unrecht in den Vordergrund gestellt wird. Allerdings soll hier nicht verschwiegen werden, daß mittlerweile Mikrocomputer auf einem Chip angeboten werden, die ein Großteil der hier dargestellten Komponenten in einem einzigen Baustein vereinen.

Zum Betrieb des MTS-Computers sind extern +5 V bei 1,5 A und +12 V bei 0,2 A anzuschließen.

2.1 *Mikroprozessor 8080*

Die Anschlüsse zwischen dem Mikroprozessor und den übrigen Systemkomponenten kann man in vier Gruppen einteilen (*Bild 3*):

- Adreßbus
- Datenbus
- Steuer-Ein- und Ausgänge
- Stromversorgung

2.1.1 Adreßbus

Der Adreßbus ist 16 bit breit und kann demzufolge einen Speicherbereich von 64 K ansprechen. Die Information auf dieser Sammelleitung stammt aus dem 16-bit-Programmzähler, der bei der normalen, sequentiellen Programmausführung schrittweise hochgezählt wird und damit einen Befehl nach dem anderen adressiert. Innerhalb des möglichen Speicherbereichs von 64 K kann eine beliebige Aufteilung in RAM- und ROM-Bereich vorgenommen werden.

2.1.2 Datenbus

Über den bidirektionalen 8-bit-Datenbus laufen in zeitlicher Aufeinanderfolge Befehle, Operanden und Adressen. Ein vom Programmzähler adressierter Befehl gelangt aus dem Speicher über den Datenbus ins Befehlshalteregister, wird dort decodiert und ausgeführt; Operanden-Ein- und Ausgaben erfolgen ebenfalls über diese Sammelleitung.

2.1.3 Steuer-Ein- und Ausgänge

Außer drei momentanen Zustandssignalen zur Systemkoordination und neben den zwei Takteingängen besitzt der Prozessor sieben Steuerleitungen, über die er mit den peripheren Komponenten kommuniziert.

Die *RESET-Leitung* dient zur Initialisierung des gesamten Systems, beispielsweise nach dem Einschalten der Versor-

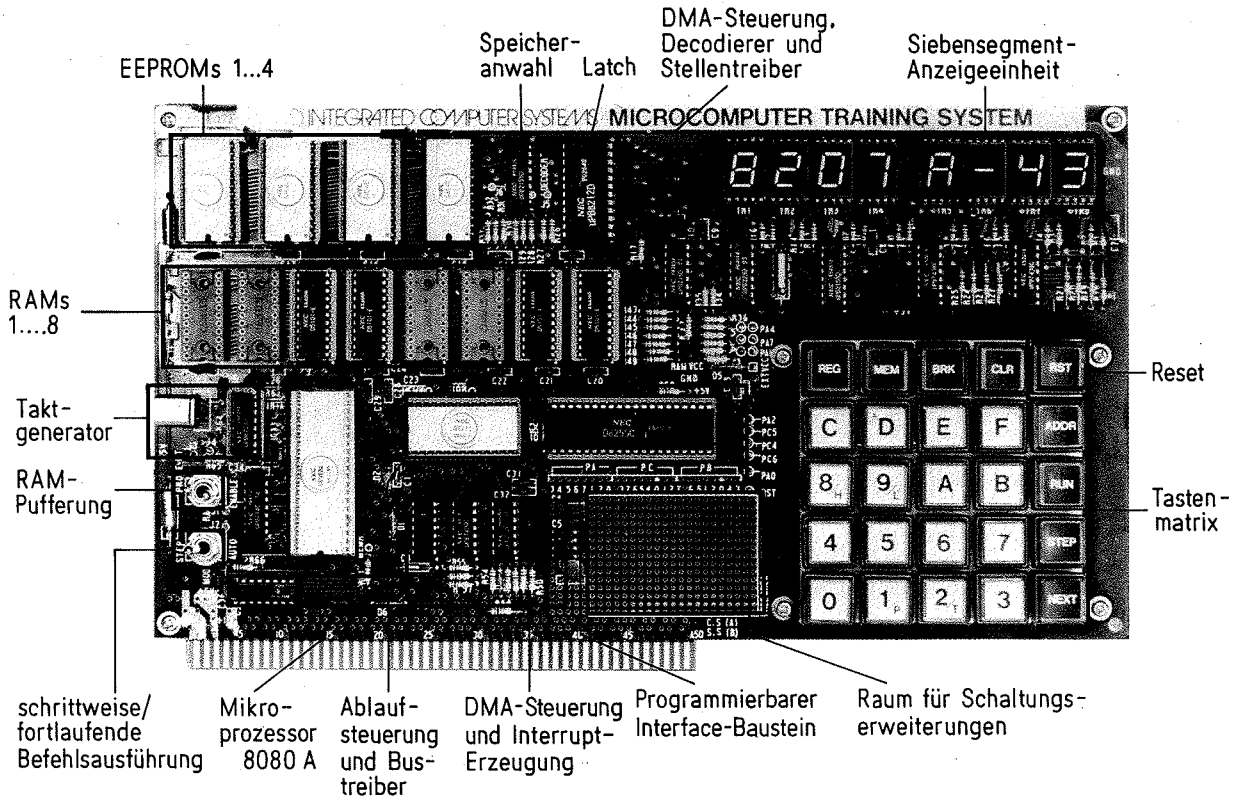


Bild 1. Das Mikrocomputer-Trainings-System (MTS) mit Darstellung der einzelnen Funktionsblöcke

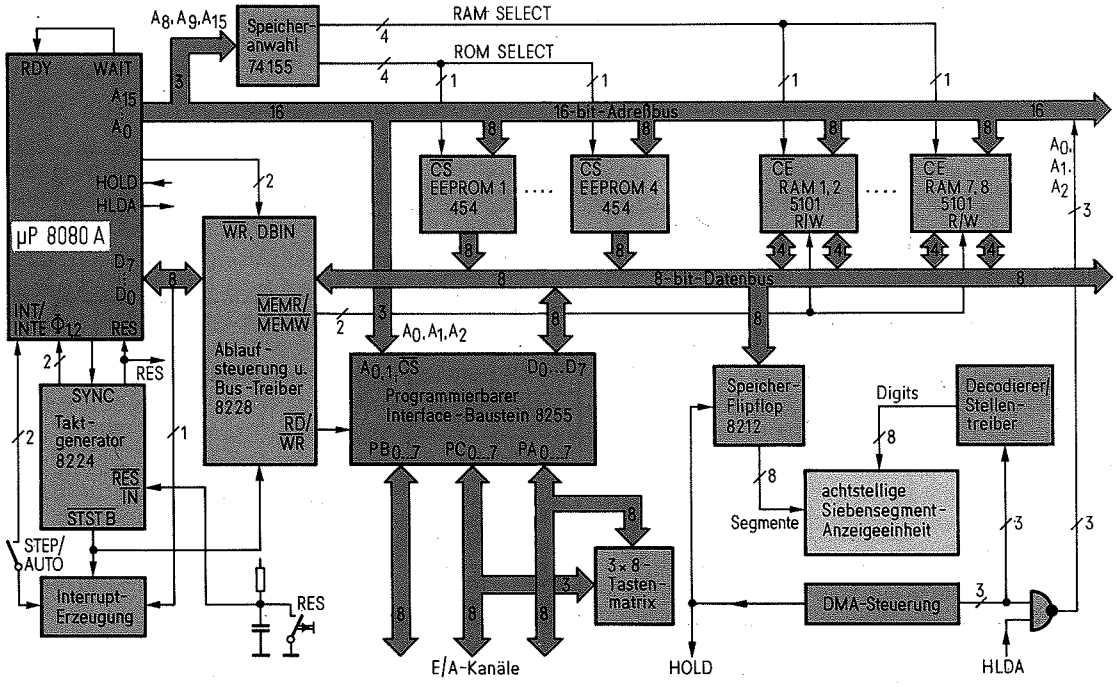


Bild 2. Blockschaltung des MTS-Ein-Karten-Computers

gungsspannung. Ein RESET-Impuls setzt den Programmzähler auf Null und sorgt dadurch dafür, daß die Programmausführung ausschließlich an dieser definierten Stelle beginnt. Außerdem wirkt sich dieses Zurücksetzen auf den *Programmierbaren Interface-Baustein* aus, dessen drei Kanäle anschließend auf Eingabe-Betrieb geschaltet sind. Die RESET-Funktion ist außer der externen Programmunterbrechung die einzige Möglichkeit, hardwaremäßig in den Systemablauf einzugreifen; diese Eigenschaft unterscheidet die Rücksetz-Taste wesentlich von den anderen Eingabetasten, von denen das Einlesen software-gesteuert erfolgt.

Die übrigen sechs Steuerleitungen lassen sich in drei Paare zusammenfassen:

Bei H-Potential am *HOLD-Eingang* (HOLD) führt der Prozessor den laufenden Maschinenzyklus noch aus und schaltet anschließend den Daten- und den Adreßbus in den hochohmigen Zustand. Dies meldet er durch ein *HOLD-ACKNOWLEDGE-Signal* (HLDA) zurück, woraufhin die Busverwaltung von außen übernommen wird. Auf diese Weise spielt sich der direkte Speicherzugriff (DMA) ab. Sobald die HOLD-Anforderung zurückgeht, arbeitet der Prozessor das laufende Programm weiter ab. Wesentlich ist, daß er in der Zwischenzeit inaktiv bleibt, im Gegensatz zur externen Programmunterbrechung (Interrupt), bei der er zwischenzeitlich ein anderes Programm ausführt.

Wenn der Prozessor softwaremäßig dafür vorbereitet ist (*Interrupt enabled*), führt er nach Aktivierung des Unterbrechungseingangs INT den laufenden Befehl noch aus, „rettet" den aktuellen Programmzählerstand und springt dann in ein Unterprogramm, um die unterbrechende Stelle zu bedienen. Er gibt der Unterbrechungs-Anforderung durch das *INTERRUPT-ENABLE-Signal* (INTE) statt, das nach Abarbeiten der Interrupt-Routine zurückgenommen wird. Daraufhin setzt der Prozessor die Programmausführung an der unterbrochenen Stelle fort, zu der er durch Rückladen des geretteten Programmzählerstandes gelangt.

Bei Eintreffen einer externen Programmunterbrechung wird die auf dem Datenbus liegende Information eingelesen und verarbeitet. Auf diese Weise ist es möglich, daß die unterbrechende Stelle eine Information auf den Datenbus schaltet (*Vektor*), durch die der Prozessor eine ganz spezifische Reaktion ausführt, mit der er individuell auf die jeweils unterbrechende Stelle eingeht.

Der *READY-Eingang* zeigt dem Prozessor an, daß nach Aussenden einer Adresse die zugehörige Information stabil auf dem Datenbus liegt. Solange dieses Signal nicht erscheint, verharrt der Prozessor im Wartezustand und meldet dies zurück (WAIT). Hierdurch kann das Zusammenarbeiten mit langsamen Speichern koordiniert werden, oder es läßt sich hierüber auch die schrittweise Befehlsausführung realisieren. Bei dem hier betrachteten MTS-Aufbau sind READY-Eingang und WAIT-Ausgang miteinander verbunden, um zur Geschwindigkeitsanpassung an die eingesetzten Speicher pro Maschinenzyklus einen Wartezyklus einzufügen.

2.1.4 Stromversorgung

Der 8080 benötigt noch die drei Versorgungsspannungen ±5 V und +12 V; da die negative Versorgung von –5 V nur mit maximal 1 mA belastet wird, kann sie problemlos durch die einfache Gleichrichtung eines Taktsignals gewonnen werden (*Bild 4*). Nach Herstellerangaben ist darauf zu achten, daß die 12-V-Versorgungsspannung nicht vor Anliegen der 5-V-Spannung eingeschaltet werden darf, und daß beim Abschalten die hohe Spannung als erste entfernt wird (oder gleichzeitig mit den 5 V).

Die Leistungsaufnahme des Mikroprozessors allein beträgt übrigens typisch 0,9 W bzw. maximal 1,25 W; zusammen mit den zum Betrieb unerläßlichen Hilfsbausteinen 8224 und 8228 erreicht der Leistungsbedarf im ungünstigsten Fall fast 3 W!

2.1.5 Interne Registerstruktur

Der Mikroprozessor 8080 besitzt außer dem zentralen Register A, dem Akkumulator, noch sechs weitere 8-bit-Register, auf die vom Programm zugegriffen werden kann. Durch die in Bild 3 gezeichnete paarweise Anordnung dieser Register B/C, D/E und H/L kommt zum Ausdruck, daß eine Reihe

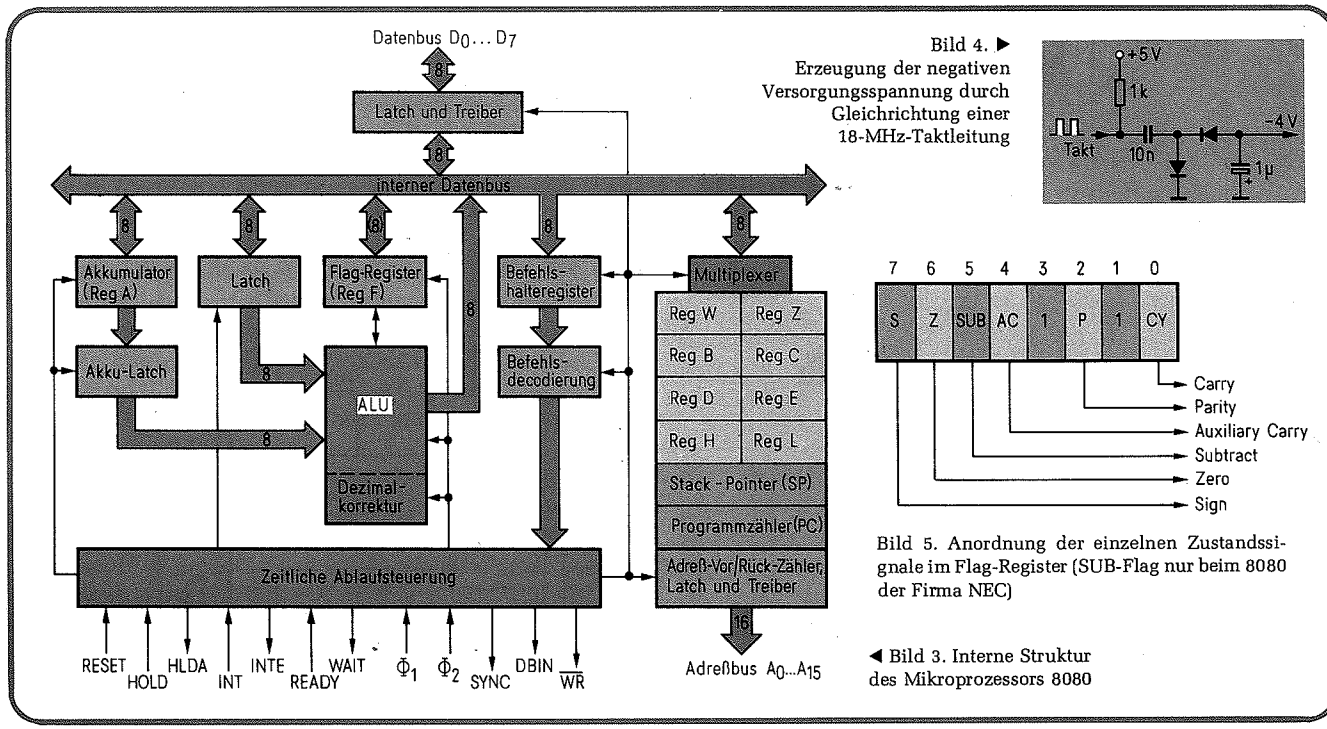


Bild 4. ▶ Erzeugung der negativen Versorgungsspannung durch Gleichrichtung einer 18-MHz-Taktleitung

Bild 5. Anordnung der einzelnen Zustandssignale im Flag-Register (SUB-Flag nur beim 8080 der Firma NEC)

◀ Bild 3. Interne Struktur des Mikroprozessors 8080

von Befehlen jeweils eins dieser Paare anspricht; darüber hinaus ist natürlich der Zugriff auf jedes Register einzeln möglich. Die Register W und Z dienen nur zur internen Zwischenspeicherung und sind für den Anwender nicht zugänglich.

Die beiden 16-bit-Register Programmzähler (PC) und Stack-Pointer (SP) dienen zur Speicher-Adressierung. Der Stand des Programmzählers weist immer auf diejenige Speicherstelle, in der der *nächste* auszuführende Befehl steht. Wenn das Programm nichts anderes vorschreibt, wird der Programmzähler kontinuierlich hochgezählt. Außerdem bestehen umfangreiche Möglichkeiten, den PC-Stand zu modifizieren und dadurch Programmsprünge auszuführen.

Mit Hilfe des Stack-Pointers kann man sich im Arbeitsspeicher ein Stapelregister von beliebiger Tiefe aufbauen. Der SP-Stand weist jeweils auf ein Paar von Speicherstellen hin, in denen eine 16-bit-Adresse oder zwei 8-bit-Datenworte abgelegt werden können. Bei jedem Einschreiben in ein solches 16-bit-Stapelregister wird der Stack-Pointer um zwei heruntergezählt, um in dem „Register-Stapel" das nächsttiefere Wortpaar zu adressieren. Umgekehrt erfolgt nach jedem Auslesen aus einem Stapelregister das automatische Hochzählen des Stack-Pointers. Durch diese Organisation lassen sich beispielsweise beliebig viele Unterprogramme ineinander verschachteln, weil die jeweilige Rücksprung-Adresse fortlaufend in dieser stapelförmigen Register-Anordnung abgelegt werden kann.

Im F-Register werden verschiedene Zustandssignale, sogenannte *Flags*, abgelegt (*Bild 5*). Entgegen einer häufig anzutreffenden falschen Vorstellung ist auch dieses Register 8 bit breit; dabei sind die Bits 1 und 3 fest auf H-Potential verdrahtet, und von den restlichen sechs können softwaremäßig nur die vier Bits „Übertrag" (Bit 0), „Parität" (Bit 2), „Null" (Bit 6) sowie „Vorzeichen" (Bit 7) abgefragt werden. Das Setzen und Zurücknehmen dieser Zustandssignale geschieht automatisch in Abhängigkeit vom Ergebnis bestimmter Operationen. Das Bit 0 im Flag-Register kann auch vom Programm gesetzt und gelöscht werden. Im Unterschied zu allen anderen 8080-Herstellern hat die japanische Firma NEC ein sechstes Zustandssignal (SUB) in das Flag-Register eingebaut, dessen Auswirkungen bei der Beschreibung des Befehlssatzes erläutert werden.

2.2 Hilfsbausteine 8224 und 8228

Der Baustein 8224 beinhaltet einen 18-MHz-Oszillator und die Logik zur Erzeugung des Zweiphasentaktes sowie einiger Steuersignale; er ist in bipolarer Schottky-Technologie aufgebaut und generiert nach dem Einschalten der Versorgungsspannung automatisch einen RESET-Impuls.

Aufgabe des Steuerbausteins 8228 ist im wesentlichen die Verwaltung des Datenbus'. Dazu schaltet er in Abhängigkeit der eingehenden Statusinformationen die internen bidirektionalen Tristate-Treiber in den jeweils erforderlichen Zustand. Außerdem erzeugt dieser Baustein bei externen Programmunterbrechungen automatisch einen Programmsprung zur dezimalen Adresse 56, solange die unterbrechende Stelle selbst keine Information auf den Datenbus schaltet. Dadurch können Interrupts auf einer einzigen Ebene ohne zusätzlichen Hardware-Aufwand verarbeitet werden, was insbesondere bei kleineren Systemen, wie beim vorliegenden, von Vorteil ist.

2.3 Speicher

Das Trainings-System ist mit 1 KByte EEPROMs des Typs µPD 454D bestückt [8]; das sind elektrisch löschbare und wiederprogrammierbare PROMs, die das ICS-Monitor-Programm enthalten. Dieses Programm bleibt fest gespeichert und ermöglicht die Kommunikation des Benutzers mit dem System. Als Arbeitsspeicher sind in der Grundausbaustufe 512 Worte RAM vorgesehen, jedoch ist auf der Karte bereits die Erweiterungsmöglichkeit bis auf 1 K RAM-Worte gegeben. Die Speicherorganisation und Chipanwahl spielen bei der Programmierung, beispielsweise beim direkten Speicherzugriff, eine wesentliche Rolle, so daß hierauf im folgenden näher einzugehen ist.

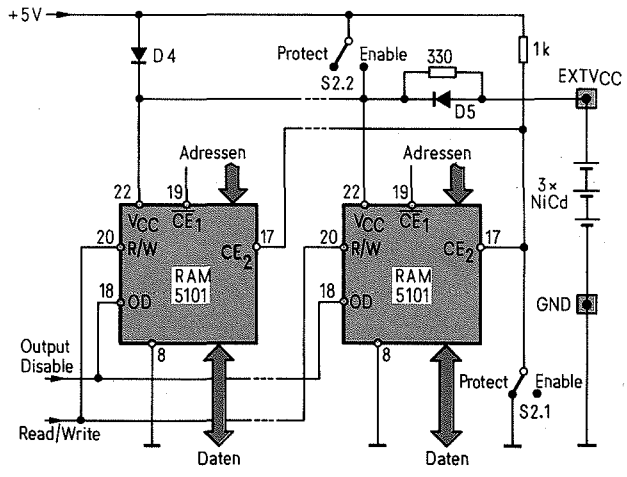


Bild 6. Stromversorgung mit externer Pufferung und Speicheranwahl der verwendeten CMOS-RAMs

Die RAMs µPD 5101-E sind in CMOS-Technologie ausgeführt und arbeiten statisch an einer einzelnen 5-V-Versorgungsspannung [9]. Sie nehmen extrem wenig Leistung auf, so daß sie auf einfache Weise mit kleinen NiCd-Akkumulatoren gepuffert werden können. Dadurch bleibt ihr Inhalt über Wochen erhalten, ohne daß eine Regenerierung erforderlich wird. Dazu muß vor Abschalten der Versorgungsspannung lediglich der Schalter für die RAM-Pufferung (siehe Bild 1) in die Stellung „PROTECT" gebracht werden. Nach Wiedereinschalten der Systemversorgung und dem Umschalten in die „ENABLE"-Stellung ist das RAM mit dem zuletzt geladenen Speicherinhalt wieder betriebsbereit. Für diesen Betrieb sind zusätzlich drei wiederaufladbare 1,2-V-Batterien und ein 330-Ω-Vorwiderstand zum Laden erforderlich, die sich problemlos auf der Platinen-Unterseite anbringen lassen (*Bild 6*).

2.3.1 Speicherorganisation

In *Bild 7* ist ausschnittweise die detaillierte Speicherorganisation dargestellt; wegen der besseren Übersichtlichkeit sind die Stromversorgungsanschlüsse fortgelassen worden.

Da das verwendete Mikroprozessor-System eine Wortlänge von 8 bit besitzt, erfolgt die Speicherung aller Daten 8-bit-weise. Davon unberührt bleibt die Tatsache, daß jedem dieser Worte eine eigene Adresse zugeordnet ist, unter der es angesprochen werden kann. Natürlich ist die Anzahl der Adreßbits nicht an die Datenwortlänge des Systems gebunden. Üblicherweise stattet man die 8-bit-Mikroprozessoren mit einem 16-bit-Adreßbus aus, weil sich dabei die Adresse aus zwei Datenworten zusammensetzen läßt, und der damit ansprechbare Speicherbereich von 64 KBytes auch für sehr umfangreiche Anwendungen ausreicht.

Die im MTS eingesetzten ROM-Bausteine sind als 256 x 8-bit-Speicher organisiert. Zum Aufbau eines 1-KByte-Speichers muß man folglich vier dieser Bausteine parallelschalten, was sich auf die Adreßeingänge und die Informationsausgänge bezieht. Zur Adressierung der 256 Worte in jedem Chip dienen 8 Adreßeingänge, die an die unteren 8 bit des Adreßbus' führen.

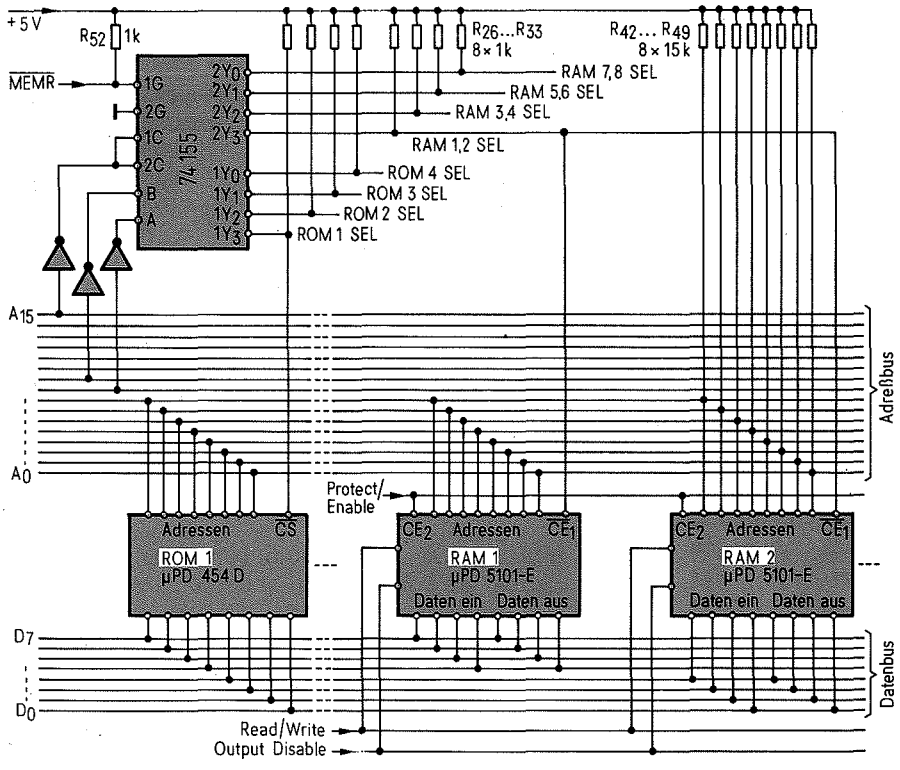


Bild 7. Speicherorganisation und Chipanwahl beim MTS-Computer

Der Arbeitsspeicher ist mit 256 x 4-bit-RAMs aufgebaut; zur Speicherung von 8-bit-Worten sind deshalb immer zwei RAMs derart zusammengeschaltet, daß das eine an die oberen vier und das andere an die unteren vier Bits des Datenbus' führen. Ein 1-K-RAM-Speicher erfordert demzufolge vier dieser Paare, also insgesamt 8 Bausteine, deren 8 Adreßeingänge wiederum parallel an die unteren 8 bit des Adreßbus' führen.

Um von den immer gleichzeitig adressierten Speicherbausteinen stets nur einen (beim ROM) bzw. stets nur ein Paar (beim RAM) anzusprechen, werden weitere Adreßbits vom Adreßbus zur Chipanwahl an einen Decodierer geführt, dessen Ausgänge je nach Eingangsinformation nur einen Baustein (bzw. nur ein Paar) aktivieren. Alle übrigen Bausteine bleiben inaktiv, d. h. ihre Ausgänge sind hochohmig und beeinflussen den Datenbus nicht.

Wie aus Bild 7 weiter hervorgeht, werden die Adreßbits A_8 und A_9 als neuntes und zehntes Adreßbit benutzt, um auf dem Umweg der Decodierung 1 KWorte zu adressieren. Die Auswahl, ob damit der RAM- oder ROM-Bereich gemeint ist, erfolgt über das höchstwertige Adreßbit A_{15}: Ist A_{15} auf H-Potential, wird der RAM-Bereich adressiert, andernfalls sprechen die unteren 10 Adreßbits das ROM an. Aus diesem Sachverhalt sind zwei Folgerungen abzuleiten: Erstens kann eine Speichererweiterung über den hier vorgesehenen Umfang von insgesamt 2 K hinaus nur durch *zusätzliche Hardware-Decodierung* (plus weitere Speicherbausteine) vorgenommen werden. Und zweitens geht aus dieser Hardware-Konfiguration unmittelbar die Adreß-Generierung für den Speicher hervor, wie sie in Bild 8 zusammengestellt ist. Dort ist ersichtlich, daß vollkommen verschiedene Adressen auf ein und dieselbe Speicherstelle zugreifen können, solange sich nur solche Adreßbits unterscheiden, die nicht signifikant sind (A_{10}...A_{14}). Außerdem läßt sich aus diesem Zusammenhang ableiten, daß dem ROM mit dem Betriebsprogramm die sedezimalen Adressen 0000...03FF zugeordnet sind, und daß sich der 1-K-RAM-Bereich von 8000...83FF erstreckt. In der Grundausbaustufe sind davon nur die oberen 512 Worte bestückt, was den Adressen 8200...83FF entspricht. Das unterste ansprechbare Byte im Arbeitsspeicher ist demzufolge unter der Adresse 8200 erreichbar, und nach dem Rücksetzen springt der Mikroprozessor über das Betriebsprogramm stets zu dieser Adresse.

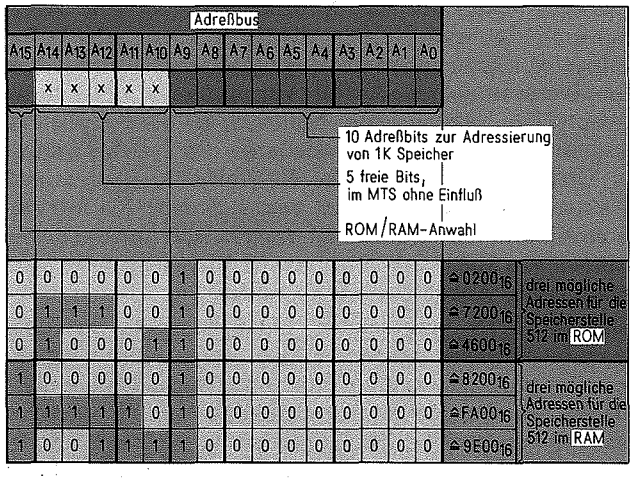


Bild 8. Belegung des Adreßbus' und Adreßgenerierung

2.4 Programmierbarer Interface-Baustein

Über den Programmierbaren Interface-Baustein 8255 ist der Datenverkehr zwischen Mikrocomputer und peripheren Komponenten möglich. Dazu besitzt dieser Baustein drei 8-bit-Kanäle (*Ports*), die vom Programm wahlweise auf Ein- oder Ausgabe-Betrieb geschaltet werden können [6]. Die zu übertragenden Informationen kommen vom Datenbus bzw. gelangen dorthin, so daß sich die Kommunikation zwischen Mikroprozessor und Außenwelt genauso einfach abspielt wie der Zugriff auf ein Register in der Zentraleinheit. Über einen Steuerbefehl wird dem Interface-Baustein die gewünschte Betriebsart mitgeteilt, die bis zum Eintreffen einer neuen Anweisung beibehalten wird.

Literatur

[1] *Gößler, R.* und *Schwerte, J.*: Auf dem Weg zur Mikroprozessor-Praxis. ELEKTRONIK 1976, H. 3, S. 74...85.

[2] 8085 Microcomputer Systems User's Manual. Vorläufiges Datenbuch der Firma Intel. November 1976.

[3] *Blomeyer-Bartenstein, H. P.*: Ein neues Mikrocomputer-Konzept. ELEKTRONIK 1976, H. 11, S. 83...87.

[4] Monitor-Programm der Firma Integrated Computer Systems aus deren ICS-Kurs 125.

[5] *Gößler, R.*: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43.

[6] *Gößler, R.*: Ein/Ausgabe-Baustein für Mikroprozessoren. ELEKTRONIK 1976, H. 11, S. 62...68.

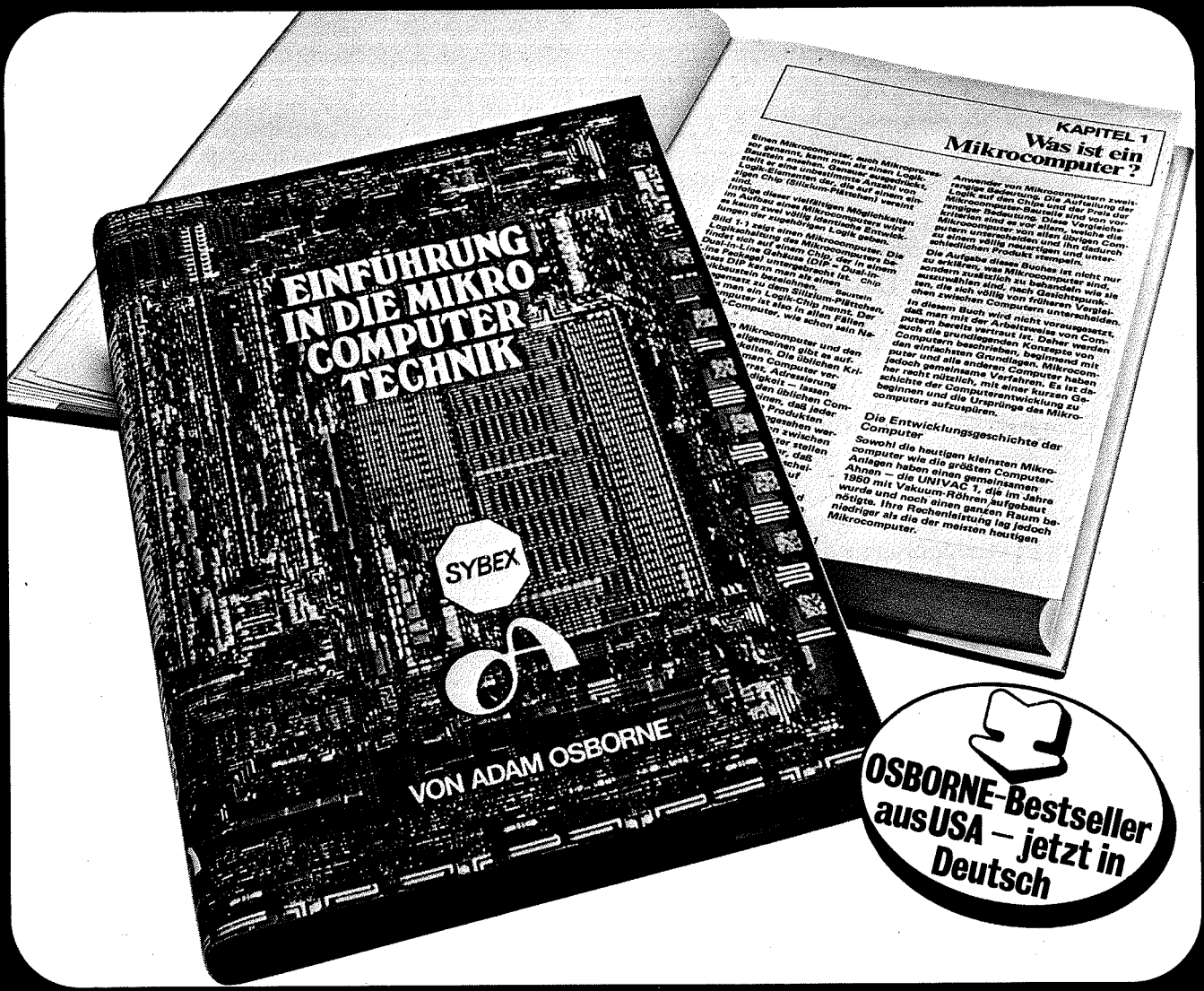
[7] Self-Study Microcomputer Hardware/Software Training Course. ICS-Kurs 126.

[8] µPD 454D. Datenblatt der Firma NEC Electronics, Düsseldorf.

[9] µPD 5101-E. Datenblatt der Firma NEC Electronics, Düsseldorf.

[10] Oktal- und Sedezimal-Code. ELEKTRONIK-Arbeitsblatt Nr. 103, H. 4/1977, S. 113...114.

MIKROCOMPUTER-TECHNIK AUS ERSTER QUELLE!



Wieviel Mikrocomputerwissen braucht man heute?

Adam Osborne gibt Ihnen in seinem US-Bestseller EINFÜHRUNG IN DIE MIKROCOMPUTER-TECHNIK in deutscher Sprache die Antwort. Von einer ausführlichen, behutsamen Einführung, über die Hardware zur richtigen Software, die interessantesten Mikroprozessoren der Firmen FAIRCHILD, NATIONAL SEMICONDUCTOR, INTEL, MOTOROLA, ROCKWELL und SIGNETICS – über Systeme, wirtschaftliche Aspekte, bis zum Lexikon finden Sie alles im neuen OSBORNE.

Ohne Zweifel ist es die bisher umfassendste, vollständigste und neutralste Darstellung der Mikrocomputer-Technik. Und das Ganze zu einem vernünftigen Preis: DM 66,–*.

Wenn Sie mehr wissen wollen, schicken wir Ihnen Informationsmaterial mit ausführlichen Inhaltsangaben.

Umfang: ca. 400 Seiten
Erscheinung: Herbst '77

Bestell-Coupon

Hiermit bestelle/n ich/wir
Stück

_____ Einführung in die Mikrocomputer-Technik von Adam Osborne, DM 66,–*.

_____ Lexikon der Mikroelektronik, DM 137,–*
Subskriptions-Preis bis 31. 10. 77: DM 109,–*

Datum

Unterschrift/Stempel

Coupon bitte deutlich ausfüllen und im Kuvert an nebenstehende Adresse schicken.

NEUERSCHEINUNG

LEXIKON DER MIKROELEKTRONIK (deutschsprachig).

Jeder, der mit Mikroelektronik zu tun hat – auch in benachbarten Gebieten der Technik – braucht dieses Nachschlagewerk.

Dieses neue umfassende Standardwerk über Mikroelektronik und -computertechnik mit über 5000 Stichwörtern und deren exakten Erläuterung schließt eine große Lücke in dem derzeitigen Fachliteraturangebot.

Herausgeber: IWT-Verlag; Co-Prod.: ICS, ca. 1000 Seiten. Subsk.-Preis bis 31. 10. 77 DM 109,–*.

te-wi

te-wi Verlag GmbH
technisch-wissenschaftliche Elektronik-Literatur
Waldfriedhofstraße 30
8000 München 70

*Preise in DM/Stck., inkl. 5.5% MwSt., zuzüglich Versandspesen.

Dipl.-Ing. Reinhard Gößler

Einführung in die Mikrocomputer-Programmierung (III)

Nach der Vorstellung der Mikrocomputer-Hardware in [1] geht der folgende Beitrag auf die Betriebssoftware des Mikrocomputer-Trainings-Systems ein, die von der Firma Integrated Computer Systems für diese Hardware-Konfiguration geschrieben wurde, wobei die speziellen Aspekte der Schulung und Einarbeitung im Vordergrund standen. Erst mit einem derartigen Betriebsprogramm, der „minimalen Intelligenz", ist der Computer in der Lage, Anweisungen vom Benutzer entgegenzunehmen und auszuführen. Von der Organisation dieser Software hängt in entscheidendem Maße die Effektivität im Einsatz ab, und der eigentliche Wert eines Systems ist an dieser Betriebssoftware zu messen. Das hier zugrunde liegende ICS-Monitor-Programm zeichnet sich durch einige Besonderheiten aus, durch die die Einarbeitung in die Grundlagen der Programmierung sehr effektiv möglich ist.

1 Terminologie

Von der Aufgabenstellung her sind grundsätzlich zwei verschiedene Mikrocomputer-Konfigurationen zu unterscheiden, nämlich solche, die in Anwender-Systemen *eingesetzt* werden, und solche, die zur *Entwicklung* von Anwender-Programmen dienen [2]. Beide brauchen ein fest gespeichertes Programm, um ihre Aufgabe ausführen zu können. Im ersten Fall ist dieses Programm problemorientiert, d. h. der Computer führt beispielsweise eine bestimmte, vorprogrammierte Steuerung durch; im zweiten Fall versetzt ihn das Programm in die Lage, solche Anweisungen vom Bediener entgegenzunehmen und auszuführen, die zur Entwicklung von Anwender-Programmen erforderlich sind. Anders ausgedrückt heißt dies, daß der Computer ein Programm braucht, das der Anwender dazu benutzen kann, seine eigenen Programme zu laden und auszutesten. Dieser Sachverhalt wird besonders von Anfängern oft nicht erkannt, und man geht fälschlicherweise davon aus, daß die Mikrocomputer-Hardware allein zum Laden von Anwender-Programmen geeignet ist.

Für das Programm, das die Kommunikation zwischen Computer und Bediener ermöglicht, hat sich die Bezeichnung *Monitor* [4] eingebürgert. Noch treffender wäre die Bezeichnung *Betriebsprogramm*, weil in dieser Software eine Vielzahl unterschiedlicher Programme zusammengefaßt ist, mit denen das Entwicklungssystem betrieben werden kann. Im Unterschied dazu versteht man unter einem *Betriebssystem* die umfassend ausgebaute Software einschließlich der sie bereitstellenden Hardware, also beispielsweise ein Floppy-Disk-Laufwerk mit gespeichertem Betriebsprogramm in einem voll ausgebauten Entwicklungssystem.

Die Beschreibung des hier betrachteten Monitors umfaßt sowohl dessen Handhabung als auch seine Organisation. Einzelheiten werden in dem Umfang vorgestellt, der im Rahmen einer gründlichen Einführung erforderlich ist. Eine bis ins Detail gehende Monitor-Beschreibung und dessen Programmausdruck sind in [3] zu finden.

2 Programmbeispiel

Um den Einsatz des Monitors am praktischen Beispiel zu demonstrieren, soll ein kurzes Programm geschrieben, geladen und ausgeführt werden. Das Programm soll den Akkumulator mit einem Anfangswert laden und diesen Wert in einer Schleife jeweils um Eins hochzählen, bis der Akkumulator-Inhalt vom Maximalwert FF_{16} auf Null springt; dann soll in den Akkumulator eine Konstante geladen werden, um das Durchlaufen des Nullzustandes anzuzeigen. Das zugehörige Flußdiagramm zeigt *Bild 1*, und in der *Tabelle 1* sind drei Befehle aufgeführt, mit denen das Programm realisiert werden soll. Dies geschieht im Vorgriff auf die Beschreibung des gesamten Befehlssatzes, um mit einem realen Beispiel arbeiten zu können. Zur Erläuterung sei hier so

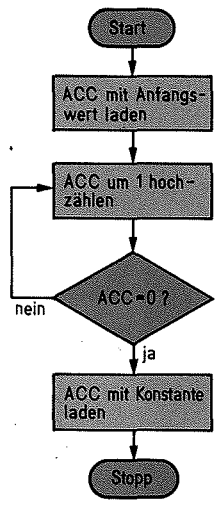


Bild 1. Flußdiagramm zum Programmbeispiel

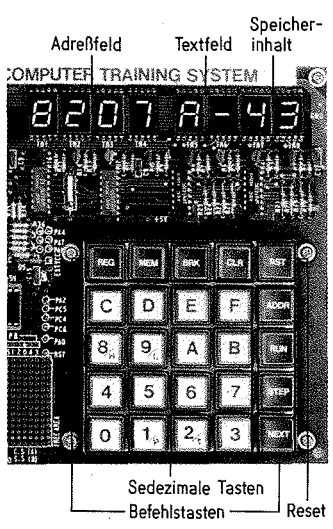


Bild 2. Bedeutung der Tasten und Anzeigefunktionen beim ICS-Mikrocomputer-Trainings-System

viel gesagt, daß es außer den Standard-Befehlen, die ein Byte im Speicher belegen, auch noch Zwei- und Dreiwort-Befehle gibt; diese werden bei der Programmausführung geschlossen verarbeitet. Der Zentraleinheit wird durch die Befehlsstruktur mitgeteilt, wieviele Worte der gerade auszuführende Befehl umfaßt und ob die zusätzlichen Worte als Daten oder Adresse zu verarbeiten sind.

Das Programmbeispiel soll, beginnend beim ersten Wort, im RAM abgelegt werden (*Tabelle 2*); aufgrund der in [1] geschilderten Zusammenhänge der MTS-Speicherorganisation trägt dieses erste RAM-Wort die sedezimale Adresse 8200. Die vier Befehle dieses Programms belegen im Arbeitsspeicher acht Worte, da einige Instruktionen mehr als ein Byte umfassen. Die Abfrage, ob der Akkumulator-Inhalt Null ist, erfolgt mit einem bedingten Sprungbefehl. Bei der Ausführung dieses Befehls zeigt das gesetzte Zero-Flag (Bit 6) im F-Register an, daß eine vorhergehende Operation zum Ergebnis Null geführt hat, und daß in diesem Fall die Programmausführung ohne Sprung fortgesetzt werden soll (die erfüllte Sprungbedingung wäre das nicht gesetzte Zero-Flag gewesen).

In den linken Spalten der Tabelle 2 sind die RAM-Adressen und ihr jeweiliger Inhalt dargestellt, wie er nach dem Laden des Programms aussehen soll. Die Zuordnung zwischen der Adresse eines Wortes und dem Inhalt des adressierten Wortes ist ein fundamentaler Zusammenhang in der Computer-Technik. Dieser Zusammenhang darf nicht falsch verstanden werden: Jedes Wort hat eine eigene Adresse, unter der es zu erreichen ist, und das gilt selbstverständlich auch dann, wenn das Wort selbst Teil einer Adresse ist.

Tabelle 1. Maschinenbefehle zur Realisierung des Programmbeispiels von Bild 1

Assembler-Code	Sedezimal-Code	Bedeutung
MVI A, dd	3E XX	move immediate into A data XX Direktes Laden des Akkumulators mit demjenigen Datenwort, das im Speicher auf die Instruktion „3E" folgt (Zweiwortbefehl)
INR A	3C	increment A (REG A) ersetzen durch (REG A) + 1
JNZ Addr	C2 ll hh	jump on no zero to address „hhll" Programmsprung zur Adresse „hhll", wenn das Zero-Flag nicht gesetzt ist; andernfalls Fortsetzung der Programmausführung beim nächsten Befehl (Dreiwortbefehl)
Schreibweise:	dd	sedezimale Abkürzung eines 8-bit-Datenwortes
	XX	beliebiger Inhalt eines 8-bit-Datenwortes
	hhll	16-bit-Adresse mit oberem (hh) und unteren (ll) Adreßbyte
	(REG A)	Inhalt von REG A

Tabelle 2. Assembliertes Programm zu Bild 1

RAM-Adresse	RAM-Inhalt (Sed.-Code)	Marke	Assembler-Code	Kommentar
8200	3E	START	MVI A, FD	ACC mit Anfangswert
8201	FD			‚FD' laden
8202	3C	LOOP	INR A	ACC um 1 hochzählen
8203	C2		JNZ LOOP	Sprung auf Marke „LOOP",
8204	02			wenn Zero-Flag nicht ge-
8205	82			setzt ist, andernfalls
				weiter bei ‚CONT'
8206	3E	CONT	MVI A, AA	ACC mit Konstante ‚AA'
8207	AA			laden
8208	00			

3 Monitor-Anweisungen

Der ICS-Mikrocomputer besitzt 8 Befehlstasten (helle Aufschrift) und 16 sedezimale Tasten (dunkle Aufschrift), über die Anweisungen und Daten eingegeben werden können (*Bild 2*). Die RESET-Taste führt unter Umgehung des Betriebsprogramms zu dem früher beschriebenen hardwaremäßigen Zurücksetzen des gesamten Systems. Die Kommunikation vom Computer zum Anwender erfolgt über die achtstellige Anzeigeeinheit, deren Bedeutung noch im einzelnen beschrieben wird. Wenn das Monitor-Programm aktiviert ist, was beispielsweise nach jedem Rücksetzen der Fall ist, wird die Schaltermatrix der Tastatur fortwährend abgetastet, um die Betätigung einer Taste abzufragen. Sobald eine definierte Eingabefolge beendet ist, führt der Monitor die entsprechende Reaktion aus. Es ist wichtig festzustellen, daß der Prozessor ständig aktiv ist, also auch in der Wartestellung, wenn er Eingaben vom Bediener erwartet. Auch dabei führt er ständig einen Befehl nach dem anderen aus, wenngleich es wegen der statischen Anzeige den Anschein hat, als ruhe die Befehlsausführung.

3.1 RESET (RST)

Bei Betätigen der RESET-Taste wird jede Aktivität des Prozessors unterbrochen, und der Programmzähler wird auf Null gesetzt; anschließend führt der Computer das an dieser Stelle beginnende Programm aus. Von der Adresse 0000 an ist das Monitor-Programm abgelegt, das bei der Abarbeitung zunächst verschiedene Anfangsbedingungen im System setzt. Dazu gehört beispielsweise die Initialisierung des Stack-Pointers (SP), auf dessen Bedeutung noch näher eingegangen wird. Außerdem erscheint im Adreßfeld der Anzeige die Adresse 8200, und der darin enthaltene Inhalt wird im Feld „Speicherinhalt" angezeigt.

3.2 Anzeigeeinheit und sedezimale Tastatur

In die im Adreßfeld der Anzeigeeinheit erscheinende Adresse kann der Anwender mit Hilfe des Monitors jeden gewünschten Inhalt laden, vorausgesetzt natürlich, daß es sich um eine gültige Adresse aus dem RAM-Bereich handelt. Das Monitor-Programm simuliert mit der Adreß-Anzeige einen Programmzählerstand, der für den Anwender bei der Eingabe und Ausführung seiner Programme gültig ist. Der tatsächliche Stand des Programmzählers in der Zentraleinheit ändert sich dagegen fortwährend, solange das Monitor-Programm ausgeführt wird; im Wartezustand resultiert aus der dynamischen Befehlsausführung eine statische Anzeige, die wiederum nicht zu dem Trugschluß verleiten darf, das System sei inaktiv.

Selbstverständlich erfolgt die Anzeige in sedezimaler Form, d. h. jede Stelle repräsentiert vier Bits. Infolge der begrenzten Darstellungsmöglichkeiten können die Buchstaben B und D auf der Siebensegment-Anzeigeeinheit nicht als Großbuchstaben wiedergegeben werden. Während man beim D noch auf den Kleinbuchstaben ausweichen kann, wäre bei einem kleinen b die Verwechslungsgefahr mit der Zahl 6 zu groß, so daß der ICS-Monitor das B als kleine Null mit einem Querstrich darüber darstellt.

Wenn eine Eingabesequenz nicht den festgelegten Bedingungen genügt, reagiert der Monitor mit einer Fehlermeldung, indem er „Err" ausgibt.

3.3 Adreß-Eingabe (ADDR)

Nach Betätigen der Taste ADDR bringt der Monitor die darauffolgende sedezimale Eingabe ins Adreßfeld der Anzeige; nachdem mindestens vier Stellen eingegeben worden sind, erscheint in den beiden Stellen rechts außen der Inhalt der adressierten Speicherstelle. Es sei nochmals darauf hingewiesen, daß die Adresse in der Anzeige nur einen Anwender-Programmzählerstand nachbildet, der vom Monitor dazu benutzt wird, auf diese Speicherstelle zuzugreifen und ihren Inhalt anzuzeigen oder zu modifizieren.

Folgen auf die Betätigung der ADDR-Taste mehr als vier sedezimale Stellen, werden immer die letzten vier als aktuelle Adresse aufgefaßt.

3.4 Speicher-Modifikation (MEM)

Der Inhalt der vom Adreßfeld der Anzeige adressierten Speicherstelle kann geändert werden, indem man die Taste MEM betätigt, gefolgt von den sedezimalen Tasten mit dem gewünschten neuen Speicherinhalt. Dieser wird in den beiden Stellen rechts außen angezeigt, ist aber während dieses Vorgangs noch nicht in den Arbeitsspeicher geladen worden. Der Einschreibvorgang wird erst dann eingeleitet, wenn der Monitor mit der Anweisung NEXT dazu aufgefordert wird.

3.5 Adresse hochzählen (NEXT)

Durch das Betätigen der Taste NEXT wird die angezeigte Adresse um Eins hochgezählt, so daß Inhalt und Adresse der nächsten Speicherstelle in der Anzeige erscheinen. Außerdem wird der zuvor über MEM eingegebene neue Speicherinhalt in die vorhergehende Adresse eingeschrieben. Wenn der Monitor einen Register-Inhalt anzeigt, schaltet die Taste NEXT zum nächsten Register weiter, das in alphabetischer Anordnung folgt, also beispielsweise von Register H auf L, von L auf A oder von A auf B.

3.6 Programmeingabe

Um das Programmbeispiel der Tabelle 2 in Maschinensprache in den Arbeitsspeicher zu laden, ist die Eingabesequenz der *Tabelle 3* einzuhalten; dort sind die Bediener-Eingaben und die daraus resultierende Anzeige einander gegenübergestellt. Nach Betätigen der RESET-Taste kommt man zur Adresse 8200, und deren zufällig vorhandener Inhalt wird angezeigt (Zeile 1). Durch die folgende Eingabe von MEM geht der Monitor in den „Schreib-Mode" über und bereitet das Laden der folgenden Information 3E nach 8200 vor (Zeile 2). Mit dem Drücken der Taste NEXT wird 3E nach 8200 geladen, und die folgende Speicherstelle 8201 wird adressiert (Zeile 3). Nun kann unmittelbar weiter eingeschrieben werden, ohne daß erneut MEM betätigt wird. Auf diese Weise läßt sich das gesamte Programm laden, wobei auch die Stelle 8208 noch eine definierte Information (00) beinhalten soll.

Nach dem Einschreiben erfolgt die Überprüfung, ob der gewünschte Inhalt auch tatsächlich geladen wurde. Dazu braucht man nur zur Adresse 8200 zurückzukehren und kann dann durch fortlaufende Betätigung der Taste NEXT den Speicherinhalt schrittweise abrufen.

Der Rücksprung auf die Anfangsadresse erfolgt durch Eingabe von ADDR mit anschließender Benennung der gewünschten Adresse. Wenn man zur Adresse 8200 springen will, kann man wegen der eben beschriebenen Monitor-Organisation auch über RESET dorthin gelangen. Der Vollständigkeit halber sei hier eingeschoben, daß im „Schreib-und-Lese-Mode" des Monitors (also nicht bei der schrittweisen Befehlsausführung) das bloße Betätigen der Taste ADDR dazu führt, daß in der Anzeige die Adresse 8200 erscheint.

Tabelle 3. Eingabesequenz beim Laden des Programms nach Tabelle 2

Zeile	Tastatur-Eingabe	Anzeige
1	RST	8200 XX
2	MEM 3 E	8200 3E
3	NEXT F D	8201 FD
4	NEXT 3 C	8202 3C
5	NEXT C 2	8203 C2
6	NEXT 0 2	8204 02
7	NEXT 8 2	8205 82
8	NEXT 3 E	8206 3E
9	NEXT A A	8207 AA
10	NEXT 0 0	8208 00
11	NEXT	8209 XX

3.7 Register-Anwahl (REG)

Die Möglichkeit, alle Datenregister in der CPU direkt anzuwählen und deren Inhalt anzuzeigen und modifizieren zu können, ist eine wesentliche Besonderheit des ICS-Monitor-Programms, weil damit die unmittelbare Auswirkung der einzelnen Befehle verfolgt werden kann; dieses Verhalten wird anschließend noch detailliert beschrieben.

Zur Anwahl eines Registers dient die Taste REG, gefolgt von der Register-Bezeichnung A, B, C, D, E, F, H bzw. L; in dieser Betriebsart interpretiert der Monitor die Taste 8 als Registerbezeichnung H, und Taste 9 liest er als L. Durch die unmittelbar folgende Eingabe einer sedezimalen Information überschreibt man den Register-Inhalt. Dies geschieht in diesem Fall unmittelbar, ohne daß man davor MEM oder danach NEXT drücken müßte. Im Textfeld der Anzeige erscheint der Register-Name, wie es auch in Bild 2 ersichtlich ist.

Eine direkte Möglichkeit, das Flag-Register F zu laden, besteht weder über den Monitor noch über einen Befehl; es ist aber beispielsweise auf dem Umweg über das Prozessor-Status-Wort (PSW) möglich, auf das im Zusammenhang mit dem Befehlssatz noch eingegangen wird.

3.8 Schrittweise Befehlsausführung (STEP)

Wenn der Betriebsarten-Wahlschalter STEP/AUTO in Stellung STEP steht, kann ein Anwender-Programm im Einzelschritt-Betrieb ausgeführt werden, indem die Taste STEP fortlaufend betätigt wird. Der Monitor führt dabei denjenigen Befehl aus, dessen Speicheradresse in der Anzeige erscheint und übernimmt anschließend wieder die Verwaltung des gesamten Systems. Dies wird hardwaremäßig dadurch realisiert, daß nach der Befehlsausführung eine Interrupt-Anforderung an die CPU gelangt; als Folge davon springt der Monitor auf eine festgelegte Anfangsadresse, von der aus er die Arbeit aufnimmt und Anweisungen des Bedieners erwartet.

Zur schrittweisen Ausführung des zuvor geladenen Programmbeispiels betätigt man zuerst die Taste RESET; der Prozessor springt auf die Startadresse 8200, in der der Befehl zum Laden des Akkumulators steht. Durch das Drücken der Taste STEP wird dieser Befehl ausgeführt, und der Prozessor

springt auf die Adresse 8202. Das in der Speicherstelle 8201 abgelegte Wort ist Bestandteil des Befehls aus 8200 und wird deshalb gemeinsam mit diesem ausgeführt. Beim nochmaligen Betätigen der STEP-Taste wird der Inhalt des Akkumulators um Eins hochgezählt, so wie es der Befehl in 8202 vorschreibt, anschließend weist der Anwender-PC-Stand auf 8203 und adressiert den darin abgelegten Sprungbefehl.

Solange der Akkumulator-Inhalt durch das vorherige Hochzählen nicht auf Null zurückgekippt ist, bewirkt die Ausführung dieses Befehls einen Programmsprung zur Adresse 8202, die als zweites und drittes Wort im Sprungbefehl enthalten ist. Dieser Sprung ist durch das Zurücksetzen des Anwender-PC-Standes auf 8202 zu verfolgen.

Die Beeinflussung des Akkumulator-Inhalts war bisher noch nicht sichtbar; mit der Eingabesequenz aus *Tabelle 4* ist dies möglich. Dazu wird nach dem Rücksetzen (Zeile 1) das Register A angewählt (Zeile 2), dessen Inhalt zu diesem Zeitpunkt noch unbestimmt ist. Mit dem ersten STEP wird der Akkumulator geladen, wie es der Befehl in 8200 vorschreibt. Anschließend wird der Akkumulator-Inhalt auf FE hochgezählt und zur Adresse 8202 (Zeilen 4 und 5) zurückgesprungen. Die Zeilen 6 und 7 zeigen einen erneuten Durchlauf der Programmschleife, und danach enthält der Akkumulator FF. Mit dem nächsten Hochzählen kippt der Akkumulator-Inhalt vom Maximalwert FF auf Null (Zeile 8). Nun wird der bedingte Sprung (Speicherstelle 8203 ff.) nicht mehr ausgeführt, und der Prozessor gelangt nach 8206 (Zeile 9), ohne daß der Inhalt von Register A dadurch beeinflußt wird. Mit der Ausführung des Lade-Befehls in 8206 wird in den Akkumulator die Konstante AA eingeschrieben (Zeile 10), um das Programmende anzuzeigen.

Tabelle 4. Schrittweise Befehlsausführung des Programmbeispiels mit Anzeige des Akkumulator-Inhalts

Zeile	Tastatur-Eingabe	Anzeige
1	RST	8200 3E
2	REG A	8200 A-XX
3	STEP	8202 A-FD
4	STEP	8203 A-FE
5	STEP	8202 A-FE
6	STEP	8203 A-FF
7	STEP	8202 A-FF
8	STEP	8203 A-00
9	STEP	8206 A-00
10	STEP	8208 A-AA

Tabelle 5. Schrittweise Befehlsausführung mit wechselnder Anzeige von Register- und Speicherinhalten

Zeile	Tastatur-Eingabe	Anzeige
1	RST	8200 3E
2	REG A	8200 A-XX
3	STEP	8202 A-FD
4	ADDR	8202 3C
5	STEP	8203 A-FE
6	ADDR	8203 C2
7	REG F	8203 F-8A
8	STEP	8202 F-8A
9	REG A	8202 A-FE
10	ADDR	8202 3C
11	STEP	8203 A-FF
12	REG F	8203 F-8E
13	ADDR	8203 C2
14	REG A	8202 A-FF
15	STEP	8202 A-FF
16	ADDR	8202 3C
17	STEP	8203 A-00
18	REG F	8203 F-4E
19	ADDR	8203 C2
20	STEP	8206 F-4E
21	REG A	8206 A-00
22	ADDR	8206 3E
23	STEP	8208 A-AA

Neben dieser fortlaufenden Überwachungsmöglichkeit eines Register-Inhalts bietet der Monitor noch eine Betriebsart an, bei der ständig zwischen Speicher- und Register-Inhalt hin und her geschaltet werden kann. Auf diese Weise kann man zuerst den Befehl selbst zur Anzeige bringen, um unmittelbar danach dessen Auswirkung auf das angesprochene Register zu verfolgen; hierin liegt die eigentliche Leistungsfähigkeit dieses Monitors, weil der Lerneffekt durch die ständige Verfolgungsmöglichkeit besonders groß ist. *Tabelle 5* zeigt einen Weg, das Programmbeispiel bei der Ausführung bis ins Detail zu verfolgen.

Die ersten drei Zeilen unterscheiden sich nicht von Tabelle 4. Dann wechselt durch Drücken der Taste ADDR die Anzeige des Register-Inhaltes über auf die Darstellung des Inhalts der adressierten Speicherstelle (Zeile 4), ehe man diesen Befehl mittels STEP ausführt (Zeile 5). Automatisch geht der Monitor zurück zur Darstellung des zuletzt angewählten Registers, und man sieht, daß das Hochzählen von FD auf FE erfolgt ist. Um die nächste Speicherstelle zu inspizieren, drückt man wiederum ADDR (Zeile 6); der dort enthaltene bedingte Sprungbefehl C2 wird ausgeführt, wenn das Zero-Flag nicht gesetzt ist. Die Eingabe REG F zeigt den Inhalt des Flag-Registers an (Zeile 7), in dem zu diesem Zeitpunkt drei Bits auf H-Potential sind: Es sind dies die Bits 7, 3 und 1, entsprechend der sedezimalen Anzeige 8A. Wie bereits beschrieben, liegen die Bits 1 und 3 fest auf 1 und haben keine Bedeutung. Das hier interessierende Zero-Flag ist demzufolge nicht gesetzt (Bit 6), und durch die Ausführung des Befehls springt das Programm auf 8202 (Zeile 8). Zeile 9 bringt wieder den Akkumulator-Inhalt zur Anzeige, und in Zeile 10 wird der als nächster anstehende Befehl angezeigt, ehe er in Zeile 11 ausgeführt wird. Dies bleibt ohne Einfluß auf das Zero-Flag in REG F (Zeile 12), in dem sich lediglich das Bit 2 (Parity-Flag) geändert hat: Der Inhalt des Flag-Registers hat sich von 8A auf 8E geändert, aber nach wie vor springt das Programm zurück zur Sprungadresse 8202 (Zeilen 13...15). Nach Anzeige und Ausführen des Hochzähl-Befehls (Zeilen 16 und 17) ist der Inhalt des Akkumulators auf Null gegangen, und das Zero-Flag wurde gesetzt (Zeile 18). Der in Zeile 19 angezeigte Sprungbefehl wird deshalb nicht ausgeführt, und das Programm gelangt zur Speicherstelle 8206 (Zeile 20). Am Inhalt des Akkumulators hat sich nichts geändert (Zeile 21), und es steht der Ladebefehl 3E zur Ausführung an (Zeile 22). Mit dem abschließenden STEP ist das Programmende erreicht, was am Inhalt des Akkumulators erkennbar ist (Zeile 23).

Auf ähnliche Weise lassen sich natürlich auch komplexere Programme ausführen, bei denen sich die Fehlersuche durch die beschriebene Monitor-Arbeitsweise bedeutend vereinfacht.

3.9 Automatischer Programmlauf (RUN)

Zur automatischen Programmausführung wird nach Eingabe der Startadresse die RUN-Taste betätigt; dazu muß der Wahlschalter STEP/AUTO in Stellung AUTO stehen. In dieser Betriebsart ist die Anzeige von Speicher- und Register-Inhalten nicht mehr möglich, weil ausschließlich das Anwenderprogramm die Systemsteuerung übernimmt und der Monitor nicht aktiviert ist. Als Hinweis sei hier nur erwähnt,

daß es darüber hinaus möglich ist, vom Anwenderprogramm her den Monitor aufzurufen und dessen Funktion in das eigene Programm einzubauen.

3.10 Löschtaste (CLR)

Mit dieser Taste lassen sich sedezimale Eingaben, die auf eine Befehlstaste folgen, wieder löschen; der angezeigte Speicher- oder Register-Inhalt bleibt dann unverändert erhalten. Außerdem kann man hierüber eingegebene Haltepunkte (*Breakpoints*) löschen.

3.11 Setzen von Haltepunkten (BRK)

Um zu überprüfen, ob die Zählschleife im Programmbeispiel richtig arbeitet, kann man einen Haltepunkt setzen, der definiert oft durchlaufen wird, und der das Programm dann automatisch anhält. Um beispielsweise den Befehl in der Speicherstelle 8202 zweimal zu durchlaufen und beim dritten Erreichen dort zu stoppen (ohne den Befehl ein drittes Mal auszuführen), ist die Sequenz der *Tabelle 6* einzugeben. Die erste Adresse spezifiziert die Stelle des Haltepunkts (Zeile 1), und die nach BRK eingegebene Zahl wird in einen Schleifenzähler geladen (Zeile 2). In Zeile 3 wird Register A zur Anzeige gebracht. Zeile 4 gibt die Startadresse vor, und in Zeile 5 wird das Programm gestartet. Wenige Mikrosekunden danach (also für den Bediener nicht wahrnehmbar) erscheint in der Anzeige der Inhalt des angewählten Registers A, nachdem die Speicheradresse 8202 mit dem zugeordneten Haltepunkt zweimal durchlaufen wurde. Erwartungsgemäß ist der Akkumulator dabei vom Anfangswert FD auf FF hochgezählt worden (Zeile 6).

Nach Setzen eines Haltepunkts muß trotz der Ausführung im „RUN-Mode" der Wahlschalter in Stellung STEP stehen, damit nach dem Leerzählen des Schleifenzählers ein Interrupt erzeugt werden kann und der Monitor wieder die Systemverwaltung übernimmt. Der ICS-Monitor kann bis zu acht verschiedene Breakpoints berücksichtigen.

3.12 Anzeige von Registerpaaren

Um den Inhalt von Registerpaaren gleichzeitig darzustellen, bedient man sich der Eingabesequenz nach *Tabelle 7*. Im Adreßfeld der Anzeigeeinheit erscheint dann der Inhalt der Benennung eingeblendet. Der Register-Inhalt wird hierbei als 16-bit-Adresse aufgefaßt, und der Inhalt der auf diese Weise adressierten Speicherstelle erscheint in der Anzeige rechts außen.

4 Monitor-Unterprogramme

Durch den modularen Aufbau des ICS-Monitors kann der Anwender eine Reihe von Unterprogrammen daraus aufrufen und sie in eigene Programme einbauen. Dazu gehören beispielsweise Routinen zum Einlesen von der Tastatur mit anschließendem Entprellen oder Unterprogramme zur Aufbereitung der anzuzeigenden Information. Außerdem besitzt der Monitor je ein Programm zum Ausgeben und Einlesen eines Speicherbereichs auf die bzw. von der Magnetbandkassette. Dazu ist eine geringe Hardware-Ergänzung erforderlich, die noch auf der Mikrocomputer-Platine Platz findet und detailliert in [3] beschrieben ist.

5 Interrupt-Verarbeitung

Durch eine minimale Hardware-Modifikation läßt sich das Mikrocomputer-System dahingehend erweitern, daß es externe Programm-Unterbrechungen auf zwei Ebenen verarbeitet. Diese Eigenschaft ist im Hinblick auf realistische Anwendungen besonders wichtig, weil sie in der Praxis na-

Tabelle 6. Eingabesequenz bei der Festlegung eines Haltepunktes (Schalter im STEP-Mode)

Zeile	Tastatur-Eingabe	Anzeige
1	ADDR 8 2 0 2	8202 3C
2	BRK 0 2	8202 BP.02
3	REG A	8200 A-XX
4	ADDR 8 2 0 0	8200 3E
5	RUN Programmausführung	
6		8202 A-FF

Tabelle 7. Eingabesequenz zur Anzeige von Registerpaaren

Registerpaar RP	Tastatur-Eingabe	Anzeige
B/C	ADDR B MEM	XXXX BC.XX
D/E	ADDR D MEM	XXXX DE.XX
H/L	ADDR H(8) MEM	XXXX HL.XX
Stack-Pointer	ADDR P(1) MEM	XXXX SP.XX
Stack-Top	ADDR T(2) MEM	XXXX ST.XX

Tabelle 8. RESTART-Instruktionen mit zugehörigen Sprungadressen und den Monitor-Reaktionen

Assembler-Code	Sedezimal-Code	Sprungziel (Sed.-Adr.)	Monitor-Reaktion
RST 0	C7	0000	Initialisierung
RST 1	CF	0008	nicht programmiert
RST 2	D7	0010	nicht programmiert
RST 3	DF	0018	nicht programmiert
RST 4	E7	0020	Software-Einsprungpunkt
RST 5	EF	0028	Sprung auf 8228
RST 6	F7	0030	Sprung auf 8230
RST 7	FF	0038	Interne Interrupt-Serviceroutine

hezu unerläßlich ist. Dazu muß lediglich die Leiterbahn zwischen dem Stift 23 des Steuerbausteins 8228 und dem 1-kΩ-Widerstand R 62 aufgetrennt werden; auf der Leiterplatte ist diese Verbindung direkt links oberhalb vom Mikroprozessor zu finden (unter dem Aufdruck „C 25").

Eine externe Programmunterbrechung erfolgt dergestalt, daß nach der Interrupt-Anforderung der Steuerbaustein 8228 am Anschlußstift 23 einen Impuls $\overline{\text{INTA}}$ (Interrupt Acknowledge) abgibt, und zeitgleich damit die auf dem Datenbus anliegende Information eingelesen und als Befehl verarbeitet wird. Dazu sind im 8080-Befehlssatz die RESTART-Instruktionen vorgesehen, bei deren Ausführung der Prozessor an die in *Tabelle 8* angegebenen Adressen springt. Dort beginnt er mit der Abarbeitung der Interrupt-Serviceroutine wie bei einem normalen Unterprogramm und springt anschließend zurück ins unterbrochene Hauptprogramm.

Aus der Tabelle 8 geht hervor, daß im Monitor an den Adressen 0028 und 0030 jeweils ein Sprungbefehl zum RAM-Bereich programmiert ist. Wenn also beispielsweise mit der externen Programmunterbrechung der Befehl RST 5 auf den Datenbus geschaltet wird, springt der Prozessor auf die Adresse 0028 und von dort zur RAM-Speicherstelle 8228, wo der Anwender die entsprechende Service-Routine ablegen kann.

Wegen des bestehenden Maschinen-Codes der RESTART-Instruktionen ist eine besonders einfache Hardwa-

re-Generierung möglich (*Bild 3*). Der Umschalter S 1 besitzt eine neutrale Mittelstellung, bei der die 8 Bits des Datenbus über die zusätzlich eingelöteten Pull-up-Widerstände auf H-Potential liegen. Damit wird der RST-7-Befehl nachgebildet (FF), den das System zur Abarbeitung selbst erzeugter Interrupts benötigt. Liegt S 1 in einer der Endstellungen, führt eine über S 2 ausgelöste externe Programmunterbrechung zur Erzeugung der Instruktionen RST 5 (Bit 4 auf LOW) bzw. RST 6 (Bit 3 auf LOW). Hierzu folgt später ein detailliertes Programmbeispiel.

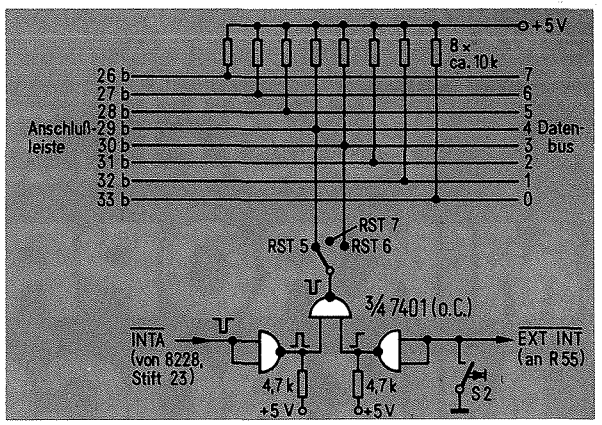


Bild 3. Hardware-Erweiterung zur Erzeugung und Verarbeitung externer Programmunterbrechungen auf zwei Ebenen

Literatur

[1] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 6, S. 64...70.

[2] *Gößler, R.*: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43 u. 71.

[3] Self-Study Microcomputer Hardware/Software Training Course. ICS-Kurs 126.

[4] *Kreidl, J.*: Arbeitsweise von Debug-Programmen. ELEKTRONIK 1977, H. 6, S. 99...101.

daß es darüber hinaus möglich ist, vom Anwenderprogramm her den Monitor aufzurufen und dessen Funktion in das eigene Programm einzubauen.

3.10 Löschtaste (CLR)

Mit dieser Taste lassen sich sedezimale Eingaben, die auf eine Befehlstaste folgen, wieder löschen; der angezeigte Speicher- oder Register-Inhalt bleibt dann unverändert erhalten. Außerdem kann man hierüber eingegebene Haltepunkte (*Breakpoints*) löschen.

3.11 Setzen von Haltepunkten (BRK)

Um zu überprüfen, ob die Zählschleife im Programmbeispiel richtig arbeitet, kann man einen Haltepunkt setzen, der definiert oft durchlaufen wird, und der das Programm dann automatisch anhält. Um beispielsweise den Befehl in der Speicherstelle 8202 zweimal zu durchlaufen und beim dritten Erreichen dort zu stoppen (ohne den Befehl ein drittes Mal auszuführen), ist die Sequenz der *Tabelle 6* einzugeben. Die erste Adresse spezifiziert die Stelle des Haltepunkts (Zeile 1), und die nach BRK eingegebene Zahl wird in einen Schleifenzähler geladen (Zeile 2). In Zeile 3 wird Register A zur Anzeige gebracht. Zeile 4 gibt die Startadresse vor, und in Zeile 5 wird das Programm gestartet. Wenige Mikrosekunden danach (also für den Bediener nicht wahrnehmbar) erscheint in der Anzeige der Inhalt des angewählten Registers A, nachdem die Speicheradresse 8202 mit dem zugeordneten Haltepunkt zweimal durchlaufen wurde. Erwartungsgemäß ist der Akkumulator dabei vom Anfangswert FD auf FF hochgezählt worden (Zeile 6).

Nach Setzen eines Haltepunkts muß trotz der Ausführung im „RUN-Mode" der Wahlschalter in Stellung STEP stehen, damit nach dem Leerzählen des Schleifenzählers ein Interrupt erzeugt werden kann und der Monitor wieder die Systemverwaltung übernimmt. Der ICS-Monitor kann bis zu acht verschiedene Breakpoints berücksichtigen.

3.12 Anzeige von Registerpaaren

Um den Inhalt von Registerpaaren gleichzeitig darzustellen, bedient man sich der Eingabesequenz nach *Tabelle 7*. Im Adreßfeld der Anzeigeeinheit erscheint dann der Inhalt der Benennung eingeblendet. Der Register-Inhalt wird hierbei als 16-bit-Adresse aufgefaßt, und der Inhalt der auf diese Weise adressierten Speicherstelle erscheint in der Anzeige rechts außen.

4 Monitor-Unterprogramme

Durch den modularen Aufbau des ICS-Monitors kann der Anwender eine Reihe von Unterprogrammen daraus aufrufen und sie in eigene Programme einbauen. Dazu gehören beispielsweise Routinen zum Einlesen von der Tastatur mit anschließendem Entprellen oder Unterprogramme zur Aufbereitung der anzuzeigenden Information. Außerdem besitzt der Monitor je ein Programm zum Ausgeben und Einlesen eines Speicherbereichs auf die bzw. von der Magnetbandkassette. Dazu ist eine geringe Hardware-Ergänzung erforderlich, die noch auf der Mikrocomputer-Platine Platz findet und detailliert in [3] beschrieben ist.

5 Interrupt-Verarbeitung

Durch eine minimale Hardware-Modifikation läßt sich das Mikrocomputer-System dahingehend erweitern, daß es externe Programm-Unterbrechungen auf zwei Ebenen verarbeitet. Diese Eigenschaft ist im Hinblick auf realistische Anwendungen besonders wichtig, weil sie in der Praxis na-

Tabelle 6. Eingabesequenz bei der Festlegung eines Haltepunktes (Schalter im STEP-Mode)

Zeile	Tastatur-Eingabe	Anzeige
1	ADDR 8 2 0 2	8202 3C
2	BRK 0 2	8202 BP 02
3	REG A	8200 A-XX
4	ADDR 8 2 0 0	8200 3E
5	RUN Programmausführung	
6		8202 A-FF

Tabelle 7. Eingabesequenz zur Anzeige von Registerpaaren

Registerpaar RP	Tastatur-Eingabe	Anzeige
B/C	ADDR B MEM	XXXX BC.XX
D/E	ADDR D MEM	XXXX DE.XX
H/L	ADDR H(8) MEM	XXXX HL.XX
Stack-Pointer	ADDR P(1) MEM	XXXX SP.XX
Stack-Top	ADDR T(2) MEM	XXXX ST.XX

Tabelle 8. RESTART-Instruktionen mit zugehörigen Sprungadressen und den Monitor-Reaktionen

Assembler-Code	Sedezimal-Code	Sprungziel (Sed.-Adr.)	Monitor-Reaktion
RST 0	C7	*0000*	Initialisierung
RST 1	CF	0008	nicht programmiert
RST 2	D7	0010	nicht programmiert
RST 3	DF	0018	nicht programmiert
RST 4	E7	0020	Software-Einsprungpunkt
RST 5	EF	0028	Sprung auf 8228
RST 6	F7	0030	Sprung auf 8230
RST 7	FF	0038	Interne Interrupt-Serviceroutine

hezu unerläßlich ist. Dazu muß lediglich die Leiterbahn zwischen dem Stift 23 des Steuerbausteins 8228 und dem 1-kΩ-Widerstand R 62 aufgetrennt werden; auf der Leiterplatte ist diese Verbindung direkt links oberhalb vom Mikroprozessor zu finden (unter dem Aufdruck „C 25").

Eine externe Programmunterbrechung erfolgt dergestalt, daß nach der Interrupt-Anforderung der Steuerbaustein 8228 am Anschlußstift 23 einen Impuls $\overline{\text{INTA}}$ (Interrupt Acknowledge) abgibt, und zeitgleich damit die auf dem Datenbus anliegende Information eingelesen und als Befehl verarbeitet wird. Dazu sind im 8080-Befehlssatz die RESTART-Instruktionen vorgesehen, bei deren Ausführung der Prozessor an die in *Tabelle 8* angegebenen Adressen springt. Dort beginnt er mit der Abarbeitung der Interrupt-Serviceroutine wie bei einem normalen Unterprogramm und springt anschließend zurück ins unterbrochene Hauptprogramm.

Aus der Tabelle 8 geht hervor, daß im Monitor an den Adressen 0028 und 0030 jeweils ein Sprungbefehl zum RAM-Bereich programmiert ist. Wenn also beispielsweise mit der externen Programmunterbrechung der Befehl RST 5 auf den Datenbus geschaltet wird, springt der Prozessor auf die Adresse 0028 und von dort zur RAM-Speicherstelle 8228, wo der Anwender die entsprechende Service-Routine ablegen kann.

Wegen des bestehenden Maschinen-Codes der RESTART-Instruktionen ist eine besonders einfache Hardwa-

re-Generierung möglich (*Bild 3*). Der Umschalter S 1 besitzt eine neutrale Mittelstellung, bei der die 8 Bits des Datenbus über die zusätzlich eingelöteten Pull-up-Widerstände auf H-Potential liegen. Damit wird der RST-7-Befehl nachgebildet (FF), den das System zur Abarbeitung selbst erzeugter Interrupts benötigt. Liegt S 1 in einer der Endstellungen, führt eine über S 2 ausgelöste externe Programmunterbrechung zur Erzeugung der Instruktionen RST 5 (Bit 4 auf LOW) bzw. RST 6 (Bit 3 auf LOW). Hierzu folgt später ein detailliertes Programmbeispiel.

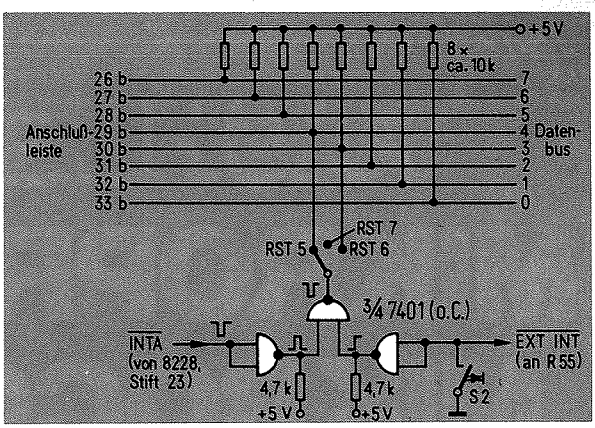


Bild 3. Hardware-Erweiterung zur Erzeugung und Verarbeitung externer Programmunterbrechungen auf zwei Ebenen

Literatur

[1] Gößler, R.: Einführung in die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 6, S. 64...70.

[2] Gößler, R.: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43 u. 71.

[3] Self-Study Microcomputer Hardware/Software Training Course. ICS-Kurs 126.

[4] Kreidl, J.: Arbeitsweise von Debug-Programmen. ELEKTRONIK 1977, H. 6, S. 99...101.

Neu in Deutschland!

FACHLEHRGANG MIKROPROZESSOR MIKROCOMPUTER

FÜR:
Entwickler, Techniker und Programmierer, die in 5 Monaten die Hardware und Software so in den Griff bekommen wollen, daß sie selbständig Mikrocomputer zusammenstellen und programmieren können.

STUDIENMETHODE:
Zu Hause sind jede Woche der Lehrgangsstoff und die Fragen einer schriftlichen Lektion durchzuarbeiten.
Der Lehrgangs-Teilnehmer kann einmal im Monat an einer mündlichen Wiederholung in einem unserer Lehrgangsorte teilnehmen.

SCHRIFTLICHES LEHRGANGS-MATERIAL:
● Was ist ein Computer? ● Was ist ein Mikrocomputer? ● Wie rechnet ein Computer? ● Schaltungen in einem Computer ● Der Zentralspeicher ● Einführung in die Programmierung ● CPU-Architektur ● Architektur eines Mikrocomputers ● Befehlsbeschreibung ● Assemblersyntax und Unterprogramme ● Adressierungstechniken ● Flußdiagramme ● Systemsoftware ● Vom Auftrag bis Ergebnis ● I/O-Interface ● Peripheriegeräte ● Programmbeispiele ● Verkehrsampelregelung ● Entwicklungssysteme.

LEHRGANGSORTE:
Düsseldorf, Hamburg, Hannover, Berlin, Frankfurt, Stuttgart, Nürnberg, München.

EXAMEN:
Sie können diesen Kursus mit einem Examen abschließen.

BETRIEBSKURSE:
Für Gruppen kann eine mündliche Begleitung des Lehrgangs im jeweiligen Betrieb durchgeführt werden. Das Institut klärt dazu die gesondert zu vereinbarenden organisatorischen Voraussetzungen.

FOLGE-LEHRGÄNGE:
Nach dem Lehrgang „Mikroprozessor/Mikrocomputer" können Folgekurse belegt werden, in denen mit Hilfe von Entwicklungssystemen Programme erstellt werden.

LEHRGANGSKOSTEN:
schriftlich DM 625,–
schriftlich + mündlich DM 875,–

← Wichtig! Bitte einsenden!

Senden Sie mir bitte die Einführungs-Broschüre über den Lehrgang „Mikroprozessor/Mikrocomputer".

Name_____

Straße_____

( ) Ort_____

*

Kurzinformation über unser Lehrinstitut:
Das ELD ist eine Tochtergesellschaft des größten holländischen Instituts für Elektronik-Lehrgänge, das staatlich anerkannt ist; Urkunde BVO/SFO – 129.448. Der Gründer des Instituts, A. J. Dirksen, ist Verfasser von 13 Elektronik-Fachbüchern, herausgegeben bei Muiderkring B. V., Bussum.

Elektronik Lehrinstitut Dirksen GmbH
Sommersstr. 20
4000 Düsseldorf 30
Tel. 0211/480143/44

Dipl.-Ing. Reinhard Gößler

Einführung in die Mikrocomputer-Programmierung (IV)

Aufbauend auf der Beschreibung der Mikrocomputer-Hardware [1] und der dafür entwickelten Betriebssoftware [2] schließt sich im folgenden Beitrag die Vorstellung des Befehlssatzes für den verwendeten Mikroprozessor 8080 an. Die tabellarisierte Zusammenstellung und detaillierte Beschreibung der einzelnen Befehle ist so ausführlich, wie sie bisher in der Literatur noch nicht zu finden war.

1 Kompatibilität verschiedener 8080-Typen

Für den Mikroprozessor 8080A gibt es zur Zeit sechs verschiedene Hersteller, einschließlich der Firma Siemens, die die Produktion Ende 1976 aufgenommen hat [4]. Gemeinhin geht man davon aus, daß diese Bausteine miteinander kompatibel sind, sich also ohne weiteres gegenständlich durch den Äquivalenztyp eines Zweitherstellers ersetzen lassen. Daß dies in der Praxis nicht immer der Fall ist, gehört zu jenen Unwägbarkeiten, die diese Materie manchmal undurchschaubar machen.

Bei der Ausarbeitung dieser Beitragsreihe lagen Mikroprozessoren vom Typ µPD8080A des japanischen Herstellers NEC zugrunde, die sich in einigen Punkten von der Ausführung des Erstherstellers Intel unterscheiden. Der wesentlichste Unterschied besteht im Aufbau des Flag-Registers, das beim Intel-Typ fünf Flag-Bits besitzt, während die Firma NEC insgesamt sechs Flags vorgesehen hat (*Bild 1*). Allein durch diese Hardware-Unterscheidung ist der Begriff „volle Kompatibilität" nicht mehr gerechtfertigt. Zur Klarstellung des Sachverhalts muß allerdings gesagt werden, daß die Modifikationen beim NEC-Typ insgesamt eine Verbesserung der Spezifikationen bedeuten und daß mittlerweile mit der Bezeichnung µPD8080AF auch eine voll kompatible Version angeboten wird.

Resultierend aus diesen Unterschieden kann der NEC-Mikroprozessor einen Befehl mehr ausführen als der Standardtyp 8080A: Durch das hinzugefügte SUBTRACT-Flag ist die Dezimalkorrektur über den Befehl DAA auch bei Subtraktionen wirksam, wo dann das CARRY- bzw. AUXILIARY-CARRY-Flag als BORROW eingehen. Um die Allgemeingültigkeit zu wahren, wird bei den nachfolgenden Betrachtungen auf diese Besonderheit nicht weiter eingegangen.

Ein weiterer markanter Unterschied zwischen den beiden Versionen besteht in der Ausführungszeit einiger Befehle, die teilweise sogar unterschiedlich abgearbeitet werden. Dieser Sachverhalt erklärt die scheinbaren Unstimmigkeiten zwischen der Anzahl von Taktzyklen und der erforderlichen Ausführungszeit, die im weiteren einander gegenübergestellt werden.

2 Befehlstabelle

In der *Tabelle 1* ist der gesamte Befehlssatz für die Mikroprozessor-Familie 8080 zusammengestellt. Die darin enthaltenen Abkürzungen erläutert *Tabelle 2*.

2.1 Assemblercode

In der ersten Spalte sind die mnemotechnischen Befehlsabkürzungen im Assemblercode aufgeführt. Diese sollte man sich zu eigen machen, selbst wenn man die Programmierung später auf unterster Ebene in Maschinensprache vornimmt. Ein im Assemblercode erstelltes Programm läßt sich nicht nur einfach erfassen, sondern es erleichtert die Umsetzung in die Maschinensprache beträchtlich, selbst wenn dieser Vorgang von Hand vorgenommen wird.

Die englische Beschreibung der einzelnen Befehle ist sprachlich nicht immer eindeutig; so wird hier beispielsweise nicht jedesmal streng zwischen einem Register und dessen Inhalt unterschieden, und außerdem differieren diese Befehlsbeschreibungen von einem Hersteller zum anderen. Daher ist in der Spalte rechts außen die detaillierte Auswirkung zu jedem Befehl separat aufgeführt.

Ein im Assemblercode aufgestelltes Programm kann man automatisch in die maschinenverständliche Form umsetzen lassen (*assemblieren*), wenn man auf einem Entwicklungssystem das dazu passende Assemblerprogramm benutzt. Dieses Entwicklungs-Hilfsprogramm setzt die mnemotechnischen Assembler-Abkürzungen in die binäre Form der Maschinensprache um und ermittelt gleichzeitig für symbolische Adressen den aktuellen Wert im Maschinenprogramm [5].

2.2 Maschinencode

Der eben angesprochene automatische Umsetzvorgang mit Hilfe eines Assembler-Programms erfordert einigen Hardware- und Speicheraufwand, der in dem hier betrachteten Minimalsystem nicht gegeben ist; deshalb muß auf dieser untersten Ebene die Programmierung direkt in Maschi-

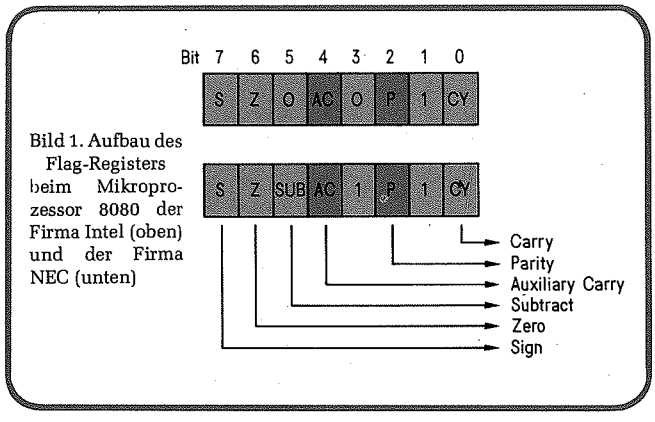


Bild 1. Aufbau des Flag-Registers beim Mikroprozessor 8080 der Firma Intel (oben) und der Firma NEC (unten)

Tabelle 1. Der vollständige Befehlsvorrat für den Mikroprozessor 8080 A

Assembler-Code	Bedeutung	Bytes	Maschinen-Code 1. Wort	2. Wort	3. Wort	Taktzyklen	Ausführungszeit auf dem MTS [µs]	Flag-beeinflussung Sign	Zero	Subtract	Aux. Carry	Parity	Carry	Auswirkung
1 Datentransport														
1.1 Laden von Konstanten														
1.1.1 Laden eines Registers														
MVI A, b	move immediate data	2	3E	XX	–	7	4,4	–	–	–	–	–	–	direktes Laden des angegebenen Registers mit ‚XX'
MVI B, b	to register	2	06	XX	–	7	4,4	–	–	–	–	–	–	
MVI C, b		2	0E	XX	–	7	4,4	–	–	–	–	–	–	
MVI D, b		2	16	XX	–	7	4,4	–	–	–	–	–	–	
MVI E, b		2	1E	XX	–	7	4,4	–	–	–	–	–	–	
MVI H, b		2	26	XX	–	7	4,4	–	–	–	–	–	–	
MVI L, b		2	2E	XX	–	7	4,4	–	–	–	–	–	–	
1.1.2 Laden eines Registerpaares														
LXI B, bb	load index immediate	3	01	XX	YY	10	6,3	–	–	–	–	–	–	direktes Laden des angegebenen Registerpaars mit ‚XXYY'
LXI D, bb		3	11	XX	YY	10	6,3	–	–	–	–	–	–	
LXI H, bb		3	21	XX	YY	10	6,3	–	–	–	–	–	–	
LXI SP, bb		3	31	XX	YY	10	6,3	–	–	–	–	–	–	
1.1.3 Laden einer Speicherstelle														
MVI m, b	move immediate data to memory	2	36	XX	–	10	6,3	–	–	–	–	–	–	Laden von „XX' in diejenige Speicherstelle, die durch (rp H,L) adressiert ist
1.2 Direktes Umladen von Daten														
1.2.1 Register mit Registerinhalt laden														
MOV A, A	move from register	1	7F	–	–	5	2,4	–	–	–	–	–	–	Inhalt des Quellenregisters umladen ins Zielregister
MOV A, B	to register	1	78	–	–	5	2,4	–	–	–	–	–	–	
MOV A, C		1	79	–	–	5	2,4	–	–	–	–	–	–	
MOV A, D		1	7A	–	–	5	2,4	–	–	–	–	–	–	
MOV A, E		1	7B	–	–	5	2,4	–	–	–	–	–	–	
MOV A, H		1	7C	–	–	5	2,4	–	–	–	–	–	–	
MOV A, L		1	7D	–	–	5	2,4	–	–	–	–	–	–	
MOV B, A		1	47	–	–	5	2,4	–	–	–	–	–	–	
MOV B, B		1	40	–	–	5	2,4	–	–	–	–	–	–	
MOV B, C		1	41	–	–	5	2,4	–	–	–	–	–	–	
MOV B, D		1	42	–	–	5	2,4	–	–	–	–	–	–	
MOV B, E		1	43	–	–	5	2,4	–	–	–	–	–	–	
MOV B, H		1	44	–	–	5	2,4	–	–	–	–	–	–	
MOV B, L		1	45	–	–	5	2,4	–	–	–	–	–	–	
MOV C, A		1	4F	–	–	5	2,4	–	–	–	–	–	–	
MOV C, B		1	48	–	–	5	2,4	–	–	–	–	–	–	
MOV C, C		1	49	–	–	5	2,4	–	–	–	–	–	–	
MOV C, D		1	4A	–	–	5	2,4	–	–	–	–	–	–	
MOV C, E		1	4B	–	–	5	2,4	–	–	–	–	–	–	
MOV C, H		1	4C	–	–	5	2,4	–	–	–	–	–	–	
MOV C, L		1	4D	–	–	5	2,4	–	–	–	–	–	–	
1.3 Indirektes Umladen von Daten														
1.3.1 Speicher mit Registerinhalt laden														
MOV m, A	move source data to	1	77	–	–	7	4,4	–	–	–	–	–	–	Inhalt des angegebenen Registers unter der Adresse abspeichern, die durch (rp H,L) adressiert ist
MOV m, B	memory	1	70	–	–	7	4,4	–	–	–	–	–	–	
MOV m, C		1	71	–	–	7	4,4	–	–	–	–	–	–	
MOV m, D		1	72	–	–	7	4,4	–	–	–	–	–	–	
MOV m, E		1	73	–	–	7	4,4	–	–	–	–	–	–	
MOV m, H		1	74	–	–	7	4,4	–	–	–	–	–	–	
MOV m, L		1	75	–	–	7	4,4	–	–	–	–	–	–	
STAX B	store accumulator	1	02	–	–	7	4,4	–	–	–	–	–	–	Inhalt des Akkumulators in derjenigen Speicherstelle ablegen, die durch (rp) adressiert ist
STAX D	contents in memory as addressed by rp	1	12	–	–	7	4,4	–	–	–	–	–	–	
1.3.2 Register mit Speicherinhalt laden														
MOV A, m	move memory contents	1	7E	–	–	7	4,4	–	–	–	–	–	–	Laden des angegebenen Registers mit dem Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist
MOV B, m	to register	1	46	–	–	7	4,4	–	–	–	–	–	–	
MOV C, m		1	4E	–	–	7	4,4	–	–	–	–	–	–	
MOV D, m		1	56	–	–	7	4,4	–	–	–	–	–	–	
MOV E, m		1	5E	–	–	7	4,4	–	–	–	–	–	–	
MOV H, m		1	66	–	–	7	4,4	–	–	–	–	–	–	
MOV L, m		1	6E	–	–	7	4,4	–	–	–	–	–	–	
LDAX B	load accumulator	1	0A	–	–	7	4,4	–	–	–	–	–	–	Akkumulator mit dem Inhalt derjenigen Speicherstelle laden, die durch (rp) adressiert ist
LDAX D	indirect	1	1A	–	–	7	4,4	–	–	–	–	–	–	
1.3.3 Registerpaar mit Registerpaar-Inhalt laden														
XTHL	exchange top of stack with HL register pair	1	E3	–	–	18	10,7	–	–	–	–	–	–	Inhalt von rp H,L mit dem Inhalt derjenigen Speicherstelle austauschen, die durch den Stack-Pointer adressiert ist
1.3.4 Speicher mit Registerpaar-Inhalt laden														
PUSH B	push register	1	C5	–	–	11	6,8	–	–	–	–	–	–	Übertragen des Registerpaar-Inhaltes in diejenigen Speicherstellen, die durch den Stack-Pointer adressiert werden
PUSH D	pair rp	1	D5	–	–	11	6,8	–	–	–	–	–	–	
PUSH H	on stack	1	E5	–	–	11	6,8	–	–	–	–	–	–	
PUSH PSW		1	F5	–	–	11	6,8	–	–	–	–	–	–	
1.3.5 Registerpaar mit Speicherinhalt laden														
POP B	pop register	1	C1	–	–	10	11,2	–	–	–	–	–	–	Inhalt derjenigen Speicherstellen, die vom Stack-Pointer adressiert werden, in das angegebene Registerpaar übertragen
POP D	pair rp	1	D1	–	–	10	11,2	–	–	–	–	–	–	
POP H	off stack	1	E1	–	–	10	11,2	–	–	–	–	–	–	
POP PSW		1	F1	–	–	10	11,2	●	●	●	●	●	●	

MOV C, L		1	4D	–	–	5	2,4	–	–	–	–	–	–	
MOV D, A		1	57	–	–	5	2,4	–	–	–	–	–	–	
MOV D, B		1	50	–	–	5	2,4	–	–	–	–	–	–	
MOV D, C		1	51	–	–	5	2,4	–	–	–	–	–	–	
MOV D, D		1	52	–	–	5	2,4	–	–	–	–	–	–	
MOV D, E		1	53	–	–	5	2,4	–	–	–	–	–	–	
MOV D, H		1	54	–	–	5	2,4	–	–	–	–	–	–	
MOV D, L		1	55	–	–	5	2,4	–	–	–	–	–	–	
MOV E, A		1	5F	–	–	5	2,4	–	–	–	–	–	–	
MOV E, B		1	58	–	–	5	2,4	–	–	–	–	–	–	
MOV E, C		1	59	–	–	5	2,4	–	–	–	–	–	–	
MOV E, D		1	5A	–	–	5	2,4	–	–	–	–	–	–	
MOV E, E		1	5B	–	–	5	2,4	–	–	–	–	–	–	
MOV E, H		1	5C	–	–	5	2,4	–	–	–	–	–	–	
MOV E, L		1	5D	–	–	5	2,4	–	–	–	–	–	–	
MOV H, A		1	67	–	–	5	2,4	–	–	–	–	–	–	
MOV H, B		1	60	–	–	5	2,4	–	–	–	–	–	–	
MOV H, C		1	61	–	–	5	2,4	–	–	–	–	–	–	
MOV H, D		1	62	–	–	5	2,4	–	–	–	–	–	–	
MOV H, E		1	63	–	–	5	2,4	–	–	–	–	–	–	
MOV H, H		1	64	–	–	5	2,4	–	–	–	–	–	–	
MOV H, L		1	65	–	–	5	2,4	–	–	–	–	–	–	
MOV L, A		1	6F	–	–	5	2,4	–	–	–	–	–	–	
MOV L, B		1	68	–	–	5	2,4	–	–	–	–	–	–	
MOV L, C		1	69	–	–	5	2,4	–	–	–	–	–	–	
MOV L, D		1	6A	–	–	5	2,4	–	–	–	–	–	–	
MOV L, E		1	6B	–	–	5	2,4	–	–	–	–	–	–	
MOV L, H		1	6C	–	–	5	2,4	–	–	–	–	–	–	
MOV L, L		1	6D	–	–	5	2,4	–	–	–	–	–	–	

1.2.2 Speicher mit Registerinhalt laden

STA addr	store accumulator contents in memory as addressed by instruction	3	32	YY	XX	13	8,3	–	–	–	–	–	–	Akkumulator-Inhalt unter Adresse ‚XXYY' abspeichern

1.2.3 Register mit Speicherinhalt laden

LDA addr	load accumulator direct with memory contents as addressed by instruction	3	3A	YY	XX	13	8,3	–	–	–	–	–	–	Inhalt der Speicherstelle ‚XXYY' in den Akkumulator laden

1.2.4 Registerpaar mit Registerpaar-Inhalt laden

XCHG	exchange contents of DE and HL register pairs	1	EB	–	–	4	2,4	–	–	–	–	–	–	Inhalte von rp D,E und rp H,L austauschen
SPHL	load stack pointer with contents of HL register pair	1	F9	–	–	5	2,4	–	–	–	–	–	–	Stack-Pointer mit (rp H,L) laden

1.2.5 Speicher mit Registerpaar-Inhalt laden

SHLD addr	store HL direct	3	22	YY	XX	16	10,3	–	–	–	–	–	–	Inhalt von rp H,L unter Adresse ‚XXYY' und ‚XXYY+1' abspeichern

1.2.6 Registerpaar mit Speicherinhalt laden

LHLD addr	load HL direct	3	2A	YY	XX	16	10,3	–	–	–	–	–	–	rp H,L mit dem Inhalt der Speicherstellen ‚XXYY' und ‚XXYY + 1' laden

1.4 Daten-Ein/Ausgabe

IN port	input	2	DB	XX	–	10	5,9	–	–	–	–	–	–	Datenübertragung zwischen dem Akkumulator und E/A-Kanal, XX'
OUT port	output	2	D3	XX	–	10	5,9	–	–	–	–	–	–	

2. Datenverarbeitung

2.1 Arithmetische Operationen

2.1.1 Additionsbefehle

INR A	increment	1	3C	–	–	5	2,9	●	●	o	□	●	–	Inhalt des angegebenen Registers um 1 erhöhen
INR B	register	1	04	–	–	5	2,9	●	●	o	□	●	–	
INR C		1	0C	–	–	5	2,9	●	●	o	□	●	–	
INR D		1	14	–	–	5	2,9	●	●	o	□	●	–	
INR E		1	1C	–	–	5	2,9	●	●	o	□	●	–	
INR H		1	24	–	–	5	2,9	●	●	o	□	●	–	
INR L		1	2C	–	–	5	2,9	●	●	o	□	●	–	
INR m	increment memory	1	34	–	–	10	6,3	●	●	o	□	●	–	Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert wird, um 1 erhöhen
INX B	increment	1	03	–	–	5	2,9	–	–	–	–	–	–	Inhalt des angegebenen Registerpaars um 1 erhöhen
INX D	register pair rp	1	13	–	–	5	2,9	–	–	–	–	–	–	
INX H		1	23	–	–	5	2,9	–	–	–	–	–	–	
INX SP		1	33	–	–	5	2,9	–	–	–	–	–	–	
ADI b	add immediate to accu	2	C6	XX	–	7	4,4	●	●	–	□	●	●	Addition von ‚XX' zum Akkumulator-Inhalt
ACI b	add immediate to accu with carry	2	CE	XX	–	7	4,4	●	●	–	□	●	●	Addition von ‚XX' und Carry-Flag zum Akkumulator-Inhalt
ADD A	add register	1	87	–	–	4	2,4	●	●	–	□	●	●	Addition von Akkumulator- und Register-Inhalt
ADD B	to accu	1	80	–	–	4	2,4	●	●	–	□	●	●	
ADD C		1	81	–	–	4	2,4	●	●	–	□	●	●	
ADD D		1	82	–	–	4	2,4	●	●	–	□	●	●	
ADD E		1	83	–	–	4	2,4	●	●	–	□	●	●	
ADD H		1	84	–	–	4	2,4	●	●	–	□	●	●	
ADD L		1	85	–	–	4	2,4	●	●	–	□	●	●	
ADD m	add memory to accu	1	86	–	–	7	4,4	●	●	–	□	●	●	Addition von Akkumulator-Inhalt und Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist
ADC A	add register to	1	8F	–	–	4	2,4	●	●	–	□	●	●	Addition von Akkumulator-Inhalt, Carry-Flag und Register-Inhalt
ADC B	accu with carry	1	88	–	–	4	2,4	●	●	–	□	●	●	
ADC C		1	89	–	–	4	2,4	●	●	–	□	●	●	
ADC D		1	8A	–	–	4	2,4	●	●	–	□	●	●	
ADC E		1	8B	–	–	4	2,4	●	●	–	□	●	●	
ADC H		1	8C	–	–	4	2,4	●	●	–	□	●	●	
ADC L		1	8D	–	–	4	2,4	●	●	–	□	●	●	
ADC m	add memory to accu with carry	1	8E	–	–	7	4,4	●	●	–	□	●	●	Addition von Akkumulator-Inhalt, Carry Flag und dem Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert wird
DAD B	add register pair	1	09	–	–	10	6,8	–	–	–	–	–	●	Addition von (rp H,L) und Inhalt des angegebenen Registerpaars
DAD D	to rp H & L	1	19	–	–	10	6,8	–	–	–	–	–	●	
DAD H		1	29	–	–	10	6,8	–	–	–	–	–	●	
DAD SP		1	39	–	–	10	6,8	–	–	–	–	–	●	

Assembler-Code	Bedeutung	Bytes	Maschinen-Code 1. Wort	2. Wort	3. Wort	Taktzyklen	Ausführungszeit auf dem MTS [µs]	Flag-beeinflussung Sign	Zero	Subtract	Aux. Carry	Parity	Carry	Auswirkung
2.1.2 Subtraktionsbefehle														
DCR A	decrement register	1	3D	–	–	5	2,9	●	●	o	□	●	–	Inhalt des angegebenen Registers um 1 vermindern
DCR B		1	05	–	–	5	2,9	●	●	o	□	●	–	
DCR C		1	0D	–	–	5	2,9	●	●	o	□	●	–	
DCR D		1	15	–	–	5	2,9	●	●	o	□	●	–	
DCR E		1	1D	–	–	5	2,9	●	●	o	□	●	–	
DCR H		1	25	–	–	5	2,9	●	●	o	□	●	–	
DCR L		1	2D	–	–	5	2,9	●	●	o	□	●	–	
DCR m	decrement memory	1	35	–	–	10	6,3	●	●	o	□	●	–	Inhalt derjenigen Speicherstelle um 1 vermindern, die von (rp H,L) adressiert wird
DCX B	decrement register pair	1	0B	–	–	5	2,9	–	–	–	–	–	–	Inhalt des angegebenen Registerpaars um 1 vermindern
DCX D		1	1B	–	–	5	2,9	–	–	–	–	–	–	
DCX H		1	2B	–	–	5	2,9	–	–	–	–	–	–	
DCX SP		1	3B	–	–	5	2,9	–	–	–	–	–	–	
SUI b	subtract immediate from accu	2	D6	XX	–	7	4,4	●	●	o	□	●	●	Akkumulator-Inhalt um ‚XX' vermindern
SBI b	subtract immediate from accu with borrow	2	DE	XX	–	7	4,4	●	●	o	□	●	●	Akkumulator-Inhalt um ‚XX' und Borrow vermindern
SUB A	subtract register from accu	1	97	–	–	4	2,4	●	●	o	□	●	●	Inhalt des angegebenen Registers vom Inhalt des Akkumulators subtrahieren
SUB B		1	90	–	–	4	2,4	●	●	o	□	●	●	
SUB C		1	91	–	–	4	2,4	●	●	o	□	●	●	
SUB D		1	92	–	–	4	2,4	●	●	o	□	●	●	
SUB E		1	93	–	–	4	2,4	●	●	o	□	●	●	
SUB H		1	94	–	–	4	2,4	●	●	o	□	●	●	
SUB L		1	95	–	–	4	2,4	●	●	o	□	●	●	
SUB m	subtract memory from accu	1	96	–	–	7	4,4	●	●	o	□	●	●	Inhalt des Akkumulators um den Inhalt derjenigen Speicherstelle vermindern, die durch (rp H,L) adressiert ist
SBB A	subtract register from accu with borrow	1	9F	–	–	4	2,4	●	●	o	□	●	●	Inhalt des angegebenen Registers und Borrow vom Inhalt des Akkumulators subtrahieren
SBB B		1	98	–	–	4	2,4	●	●	o	□	●	●	
SBB C		1	99	–	–	4	2,4	●	●	o	□	●	●	
SBB D		1	9A	–	–	4	2,4	●	●	o	□	●	●	
SBB E		1	9B	–	–	4	2,4	●	●	o	□	●	●	
SBB H		1	9C	–	–	4	2,4	●	●	o	□	●	●	
SBB L		1	9D	–	–	4	2,4	●	●	o	□	●	●	
SBB m	subtract memory from accu with borrow	1	9E	–	–	7	4,4	●	●	o	□	●	●	Inhalt des Akkumulators um Borrow und den Inhalt derjenigen Speicherstelle vermindern, die durch (rp H,L) adressiert ist

Assembler-Code	Bedeutung	Bytes	Maschinen-Code 1. Wort	2. Wort	3. Wort	Taktzyklen	Ausführungszeit auf dem MTS [µs]	Flag-beeinflussung Sign	Zero	Subtract	Aux. Carry	Parity	Carry	Auswirkung
CMP A	compare register with accu	1	BF	–	–	4	2,4	●	●	–	□	●	●	Vergleich zwischen Akkumulator-Inhalt und Inhalt des angegebenen Registers
CMP B		1	B8	–	–	4	2,4	●	●	–	□	●	●	
CMP C		1	B9	–	–	4	2,4	●	●	–	□	●	●	
CMP D		1	BA	–	–	4	2,4	●	●	–	□	●	●	
CMP E		1	BB	–	–	4	2,4	●	●	–	□	●	●	
CMP H		1	BC	–	–	4	2,4	●	●	–	□	●	●	
CMP L		1	BD	–	–	4	2,4	●	●	–	□	●	●	
CMP m	compare memory with accu	1	BE	–	–	7	4,4	●	●	–	□	●	●	Vergleich zwischen Akkumulator-Inhalt und Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist
2.2.5 Komplement-Bildung														
CMA	complement accu	1	2F	–	–	4	2,4	–	–	–	–	–	–	Invertieren des Akkumulator-Inhalts
2.2.6 Carry-Flag-Operationen														
STC	set carry	1	37	–	–	4	2,4	–	–	–	–	–	●	Carry-Flag setzen
CMC	complement carry	1	3F	–	–	4	2,4	–	–	–	–	–	●	Carry-Flag invertieren
2.3 Schiebe-Operationen														
RRC	rotate accu right	1	0F	–	–	4	2,4	–	–	–	–	–	●	Akkumulator-Inhalt zyklisch um 1 bit nach rechts verschieben
RLC	rotate accu left	1	07	–	–	4	2,4	–	–	–	–	–	●	Akkumulator-Inhalt zyklisch um 1 bit nach links verschieben
RAR	rotate accu right through carry	1	1F	–	–	4	2,4	–	–	–	–	●		Akkumulator-Inhalt unter Einbeziehung des Carry-Flags zyklisch um 1 bit nach rechts verschieben
RAL	rotate accu left through carry	1	17	–	–	4	2,4	–	–	–	–	–	●	Akkumulator-Inhalt unter Einbeziehung des Carry-Flags zyklisch um 1 bit nach links verschieben
3 Beeinflussung des Programmablaufs														
3.1 Programmsprünge														
3.1.1 Unbedingte Sprünge														
JMP addr	jump unconditional	3	C3	YY	XX	10	7,3	–	–	–	–	–	–	Programmsprung zur Adresse ‚XXYY'
PCHL	rp H & L to program counter	1	E9	–	–	5	2,4	–	–	–	–	–	–	Programmfortsetzung an der Speicherstelle, deren Adresse in rp H,L steht
3.1.2 Bedingte Sprünge														
JC addr	jump on carry	3	DA	YY	XX	10	6,3	–	–	–	–	–	–	Programmsprung zur Adresse

2.1.3 BCD-Umsetzung

DAA	decimal adjust accu	1	27	–	–	4	2,4	●	●	–	□	●	●	BCD-Korrektur des Akkumulator-Inhalts

2.2 Logische Operationen
2.2.1 Logisch-UND

ANI b	and immediate with accu	2	E6	XX	–	7	4,4	●	●	–	□	●	●	UND-Verknüpfung zwischen Akkumulator-Inhalt und ‚XX'
ANA A	and register with accu	1	A7	–	–	4	2,4	●	●	–	□	●	●	UND-Verknüpfung zwischen Akkumulator-Inhalt und Inhalt des angegebenen Registers
ANA B		1	A0	–	–	4	2,4	●	●	–	□	●	●	
ANA C		1	A1	–	–	4	2,4	●	●	–	□	●	●	
ANA D		1	A2	–	–	4	2,4	●	●	–	□	●	●	
ANA E		1	A3	–	–	4	2,4	●	●	–	□	●	●	
ANA H		1	A4	–	–	4	2,4	●	●	–	□	●	●	
ANA L		1	A5	–	–	4	2,4	●	●	–	□	●	●	
ANA m	and memory with accu	1	A6	–	–	7	4,4	●	●	–	□	●	●	UND-Verknüpfung zwischen Akkumulator-Inhalt und dem Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist

2.2.2 Logisch-ODER

ORI b	or immediate with accu	2	F6	XX	–	7	4,4	●	●	–	□	●	●	ODER-Verknüpfung zwischen Akkumulator-Inhalt und ‚XX'
ORA A	with accu	1	B7	–	–	4	2,4	●	●	–	□	●	●	ODER-Verknüpfung zwischen Akkukmulator-Inhalt und Inhalt des angegebenen Registers
ORA B		1	B0	–	–	4	2,4	●	●	–	□	●	●	
ORA C		1	B1	–	–	4	2,4	●	●	–	□	●	●	
ORA D		1	B2	–	–	4	2,4	●	●	–	□	●	●	
ORA E		1	B3	–	–	4	2,4	●	●	–	□	●	●	
ORA H		1	B4	–	–	4	2,4	●	●	–	□	●	●	
ORA L		1	B5	–	–	4	2,4	●	●	–	□	●	●	
ORA m	or memory with accu	1	B6	–	–	7	4,4	●	●	–	□	●	●	ODER-Verknüpfung zwischen Akkumulator-Inhalt und dem Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist

2.2.3 Exklusiv-ODER

XRI b	exclusive or immediate with accu	2	EE	XX	–	7	4,4	●	●	–	□	●	●	Exklusiv-ODER-Verknüpfung zwischen Akkumulator-Inhalt und ‚XX'
XRA A	exclusive or register with accu	1	AF	–	–	4	2,4	●	●	–	□	●	●	Exklusiv-ODER-Verknüpfung zwischen Akkumulator-Inhalt und Inhalt des angegebenen Registers
XRA B		1	A8	–	–	4	2,4	●	●	–	□	●	●	
XRA C		1	A9	–	–	4	2,4	●	●	–	□	●	●	
XRA D		1	AA	–	–	4	2,4	●	●	–	□	●	●	
XRA E		1	AB	–	–	4	2,4	●	●	–	□	●	●	
XRA H		1	AC	–	–	4	2,4	●	●	–	□	●	●	
XRA L		1	AD	–	–	4	2,4	●	●	–	□	●	●	
XRA m	exclusiv or memory with accu	1	AE	–	–	7	4,4	●	●	–	□	●	●	Exklusiv-ODER-Verknüpfung zwischen Akkumulator-Inhalt und Inhalt derjenigen Speicherstelle, die durch (rp H,L) adressiert ist

2.2.4 Vergleich zweier Bytes

CPI b	compare immediate with accu	2	FE	XX	–	7	4,4	●	●	–	□	●	●	Vergleich zwischen Akkumulator-Inhalt und ‚XX'

JZ addr	jump on zero	3	CA	YY	XX	10	6,3	–	–	–	–	–	–	‚XXYY', wenn das betreffende
JM addr	jump on minus	3	FA	YY	XX	10	6,3	–	–	–	–	–	–	Flag gesetzt ist
JPE addr	jump on parity even	3	EA	YY	XX	10	6,3	–	–	–	–	–	–	
JNC addr	jump on no carry	3	D2	YY	XX	10	6,3	–	–	–	–	–	–	Programmsprung zur Adresse
JNZ addr	jump on no zero	3	C2	YY	XX	10	6,3	–	–	–	–	–	–	‚XXYY', wenn das betreffende
JP addr	jump on positive	3	F2	YY	XX	10	6,3	–	–	–	–	–	–	Flag nicht gesetzt ist
JPO addr	jump on parity odd	3	E2	YY	XX	10	6,3	–	–	–	–	–	–	

3.2 Sprung ins Unterprogramm
3.2.1 Unbedingter Sprung

CALL addr	call unconditional	3	CD	YY	XX	17	10,7	–	–	–	–	–	–	Sprung in das bei Adresse ‚XXYY' beginnende Unterprogramm

3.2.2 Bedingte Sprünge

CC addr	call on carry	3	DC	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	Sprung in das bei Adresse
CZ addr	call on zero	3	CC	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	‚XXYY' beginnende Unterpro-
CM addr	call on minus	3	FC	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	gramm, wenn das betreffende
CPE addr	call parity even	3	EC	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	Flag gesetzt ist
CNC addr	call on no carry	3	D4	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	Sprung in das bei Adresse
CNZ addr	call on no zero	3	C4	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	‚XXYY' beginnende Unterpro-
CP addr	call on positive	3	F4	YY	XX	17[1])	9,3[2])	–	–	–	–	–	–	gramm, wenn das betreffende
CPO addr	call on parity odd	3	E4	YY	XX	17[1])	8,3[2])	–	–	–	–	–	–	Flag nicht gesetzt ist

3.3 Rücksprung aus dem Unterprogramm
3.3.1 Unbedingter Rücksprung

RET	return	1	C9	–	–	10	6,8	–	–	–	–	–	–	Rücksprung zu der Speicherstelle, auf die der Stack-Pointer weist

3.3.2 Bedingte Rücksprünge

RC	return on carry	1	D8	–	–	11[3])	6,8[4])	–	–	–	–	–	–	Rücksprung zu der Speicher-
RZ	return on zero	1	C8	–	–	11[3])	6,8[4])	–	–	–	–	–	–	stelle, auf die der Stack-
RM	return on minus	1	F8	–	–	11[3])	6,8[4])	–	–	–	–	–	–	Pointer weist, wenn das be-
RPE	return on parity even	1	E8	–	–	11[3])	6,8[4])	–	–	–	–	–	–	treffende Flag gesetzt ist
RNC	return on no carry	1	D0	–	–	11[3])	6,8[4])	–	–	–	–	–	–	Rücksprung zu der Speicher-
RNZ	return on no zero	1	C0	–	–	11[3])	6,8[4])	–	–	–	–	–	–	stelle, auf die der Stack-
RP	return on positive	1	F0	–	–	11[3])	6,8[4])	–	–	–	–	–	–	Pointer weist, wenn das be-
RPO	return on parity odd	1	E0	–	–	11[3])	6,8[4])	–	–	–	–	–	–	treffende Flag nicht gesetzt ist

3.4 Externe Programmunterbrechungen
3.4.1 Interrupt-Beeinflussung

EI	enable interrupt	1	FB	–	–	4	2,4	–	–	–	–	–	–	Interrupt ermöglichen
DI	disable interrupt	1	F3	–	–	4	2,4	–	–	–	–	–	–	Interrupt sperren

3.4.2 Interrupt-Sprungtabelle

RST 0	restart 0	1	C7	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0000
RST 1	restart 1	1	CF	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0008
RST 2	restart 2	1	D7	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0010
RST 3	restart 3	1	DF	1	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0018
RST 4	restart 4	1	E7	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0020
RST 5	restart 5	1	EF	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0028
RST 6	restart 6	1	F7	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0030
RST 7	restart 7	1	FF	–	–	11	6,8	–	–	–	–	–	–	Sprung zur Adresse 0038

3.5 Programmstop und -fortsetzung

HLT	halt	1	76	–	–	(7)	(4,4)	–	–	–	–	–	–	Programmstop, Interrupt
NOP	no operation	1	00	–	–	4	2,4	–	–	–	–	–	–	Programmzähler um 1 erhöhen

Tabelle 2. Bedeutung der in Tabelle 1 verwendeten Abkürzungen

addr	16-bit-Speicheradresse
b	Byte
bb	zwei aufeinanderfolgende Bytes
m	Speicherstelle
port	Bezeichnung eines E/A-Kanals
rp	Registerpaar
(rp)	Inhalt eines Registerpaares
rp B	Registerpaar B und C
rp D	Registerpaar D und E
rp H	Registerpaar H und L
XX	beliebiger Inhalt eines Bytes
XXYY	beliebiger Inhalt zweier aufeinanderfolgender Bytes
PC	Programmzähler
PSW	Prozessor-Status-Wort (Inhalt von Reg A und Reg F)
SP	Stack-Pointer
ST	Stack-Top
●	für alle 8080-Typen gültig
o	gilt nur für den Intel-8080
□	gilt nur für den NEC-8080
[1])	11 Zyklen bei nichterfüllter Bedingung
[2])	6,8 µs bei nichterfüllter Bedingung
[3])	5 Zyklen bei nichterfüllter Bedingung
[4])	2,9 µs bei nichterfüllter Bedingung

nensprache erfolgen. Um diesen Vorgang zu erleichtern, ist das im MTS fest abgespeicherte ICS-Betriebsprogramm zur Entgegennahme sedezimaler Informationen vorbereitet. Darunter ist zu verstehen, daß zur Eingabe eines 8-bit-Wortes nicht acht einzelne Schalter zur Vorwahl von logisch „0" bzw. „1" betätigt werden müssen, sondern daß dazu die Betätigung zweier Tasten einer sedezimalen Tastatur genügt. Dieser Vorgang ist die unerläßliche Voraussetzung für die manuelle Programmerstellung; im Gegensatz dazu ist die binäre Eingabe so fehlerintensiv und mühselig, daß ihre Tauglichkeit auf Programme mit wenigen Schritten beschränkt bleibt.

Neben der Spalte „Maschinencode" ist die Anzahl der Bytes angegeben, die der jeweilige Befehl umfaßt; es gibt im 8080-System Ein-, Zwei- und Dreiwortbefehle, die im Speicher dann nacheinander die entsprechende Anzahl von Speicherstellen belegen. Es ist wichtig zu wissen, daß auch die Mehrwortbefehle geschlossen verarbeitet werden. Aus der Befehlsstruktur erkennt die Zentraleinheit, wieviele Worte der gerade auszuführende Befehl umfaßt. Bei Mehrwortbefehlen veranlaßt die CPU deshalb das Auslesen von einem bzw. zwei weiteren Worten aus dem Programmspeicher, um den Befehl insgesamt zu erfassen.

2.3 Ausführungszeiten

Im Gegensatz zu reinen Software-Lösungen spielt bei Prozeßsteuerungen die Ausführungszeit eines Programms eine ganz entscheidende Rolle. Hiervon hängt oft primär die Realisierbarkeit eines Projektes ab, wofür die in [6] beschriebene Problemlösung ein ausgezeichnetes Beispiel ist. Darum sind in der Tabelle 1 detaillierte Angaben über Befehlsausführungszeiten enthalten, um beispielsweise programmierte Zeitverzögerungen realisieren oder definierte Angaben über Reaktionszeiten auf externe Programmunterbrechungen machen zu können.

Die Angabe von Taktzyklen bezieht sich auf die Intel-Version des Typs 8080; mit Kenntnis der Taktfrequenz läßt sich hieraus die *minimale* Befehlsausführungszeit errechnen, wenn man eine Taktperiodendauer (= Taktzykluszeit) von 488 ns zugrunde legt; diese Zeitangabe resultiert aus der durch Neun geteilten Oszillatorfrequenz von 18,432 MHz, wie sie auf dem MTS vorzufinden ist.

Die tatsächliche Ausführungszeit auf dem Mikrocomputer-Trainings-System unterscheidet sich aus drei Gründen von den eben genannten Werten: Erstens führt die hier eingesetzte NEC-Version des 8080 einige Befehle schneller und andere langsamer aus als der Intel-Typ; zweitens wird durch eine besondere Schaltungsmaßnahme bei jedem Speicherzugriff der CPU ein Wartezyklus eingefügt, um die langsamen Speicher mit der Zentraleinheit zu koordinieren; und drittens entsteht dadurch eine Verlangsamung der Ausführungsgeschwindigkeit, daß die CPU durch den ständig stattfindenden direkten Speicherzugriff der Anzeigeeinheit „gebremst" wird.

2.4 Flag-Beeinflussung

Die eigentliche „Intelligenz" eines Computers besteht darin, Entscheidungen treffen zu können, die in Abhängigkeit bestimmter Ergebnisse unterschiedliche Reaktionen hervorrufen; dies ist nicht zu verwechseln mit der hohen Arbeitsgeschwindigkeit, die als zweite wesentliche Eigenschaft die Leistungsfähigkeit eines Computers bestimmt. Entscheidungen werden in einem Computer-Programm über Abfragen realisiert, die einen bedingten Sprung zur Folge haben; je nach erfüllter bzw. ausgebliebener Abfragebedingung wird die Programmausführung an anderer Stelle fortgesetzt, um die gewünschte Reaktion auszuführen. Als Indikator für die verschiedenen Bedingungen fungieren die *Flags*, denen wegen der eben geschilderten Zusammenhänge eine ganz entscheidende Bedeutung im Gesamtsystem zukommt.

Soweit logische und arithmetische Operationen Einfluß auf die Flags haben, ist dies in der Tabelle 1 angegeben. Diese Einflußnahme geschieht in der in *Tabelle 3* zusam-

Tabelle 3. Bedeutung der einzelnen Flag-Bits und deren Setzbedingung

Flag	Bedeutung	Setzbedingung
Sign	Vorzeichen	Das Ergebnis einer Operation führt zu einer „1" im höchstwertigen Bit
Zero	Null	Das Ergebnis einer Operation ist Null
Subtract [1])	Subtraktion	Die zuletzt ausgeführte Operation war eine Subtraktion
Auxiliary Carry	Zwischenübertrag	Die Ausführung einer Operation führt zu einem Übertrag von Bit 3 nach Bit 4 (wird in Verbindung mit dem DAA-Befehl benutzt)
Parity	Parität	Das Ergebnis einer Operation enthält eine gerade Anzahl von Bits, die auf logisch-1 sind
Carry	Übertrag	Eine Operation führt zu einem Übertrag (Addition) bzw. zu einem negativen Übertrag (Subtraktion) über Bit 7 hinaus

[1]) Nur bei den 8080-Mikroprozessoren der Firma NEC

mengefaßten Form. Es ist hierbei der unterschiedliche Aufbau des Flag-Registers bei der Intel- bzw. NEC-Version des 8080 zu beachten, wie er aus *Bild 1* hervorgeht. Ein Flag-Bit bleibt stets so lange gesetzt, bis die nächste Operation ausgeführt wird, die es zurücksetzen kann; wie aus Tabelle 1 hervorgeht, beeinflussen längst nicht alle Befehle die Zustandssignale im Flag-Register. Außerdem bestehen bei den verschiedenen 8080-Versionen einige Unterschiede hinsichtlich der Flag-Beeinflussung.

3 Befehlsgruppen

Von den insgesamt 256 Möglichkeiten, die durch ein 8-bit-Wort dargestellt werden können, sind im 8080-Befehlssatz 244 ausgenutzt worden; diese lassen sich nach unterschiedlichen Gesichtspunkten zusammenfassen, was in der hier gewählten Weise zu insgesamt 91 Grundbefehlen führt, die sich generell auf drei Befehlsgruppen aufteilen:

- Datentransportbefehle,
- Datenverarbeitungsbefehle,
- Befehle zur Beeinflussung des Programmablaufs.

Soweit einzelne Befehle einer näheren Erläuterung bedürfen, geschieht dies im folgenden Abschnitt. Hier sei nur auf eine Besonderheit hingewiesen, die von der Auswirkung her keinen Sinn ergibt: Die Befehle MOV A, A, MOV B, B usf. besagen, daß der Inhalt des betreffenden Registers in eben dieses Register transportiert werden soll, was demzufolge am Registerinhalt selbst nichts ändert. Diese Redundanz im Befehlsvorrat entsteht durch den Befehlsaufbau, wie er für die Register-Umladebefehle MOV in *Bild 2* skizziert ist. Hierbei geben die beiden Bits 7 und 6 an, daß es sich um einen Ladebefehl von Register zu Register handelt, und die restlichen beiden 3-bit-Gruppen spezifizieren das Quell- und das Zielregister; da in beiden Fällen jedes der sieben Register angesprochen werden kann, entsteht die angesprochene Überlappung.

4 Befehlszyklus

Unter den teilweise irreführenden Angaben irgendwelcher Zykluszeiten besitzt der *Befehlszyklus* eine wohldefinierte Bedeutung; er faßt die Summe der einzelnen Steuersignale zusammen, die zum Holen und Ausführen eines Befehls erforderlich sind [7]. Dementsprechend unterscheidet man im Befehlszyklus die Hol- und die Ausführungsphase. Zur Verdeutlichung der nachfolgend geschilderten Abläufe dient Bild 3 aus [1].

4.1 *Holzyklus*

Die Ausführung eines *jeden* Befehls beginnt mit der Holphase, während der der Befehl aus dem Speicher ausgelesen wird. Dazu schaltet die interne Ablaufsteuerung den aktuellen Programmzählerstand auf den Adreßbus und generiert gleichzeitig diejenigen Steuersignale, die zum Auslesen der adressierten Speicherstelle erforderlich sind. Das auf diese Weise angesprochene Befehlswort gelangt über den Datenbus in das Befehlshalteregister in der CPU, wo es anschließend decodiert wird.

Es ist eine wesentliche Eigenschaft *aller* Computer, daß während dieser Phase, also noch *vor* der eigentlichen Befehlsausführung, der Programmzähler automatisch um Eins hochgezählt wird, um die nächste Speicherstelle zu adressieren. Das Erhöhen zu diesem Zeitpunkt hat einen plausiblen Grund: Wenn beispielsweise in der folgenden Phase der Befehlsausführung ein Sprung ins Unterprogramm durchgeführt werden soll, kann der aktuelle Programmzähler-

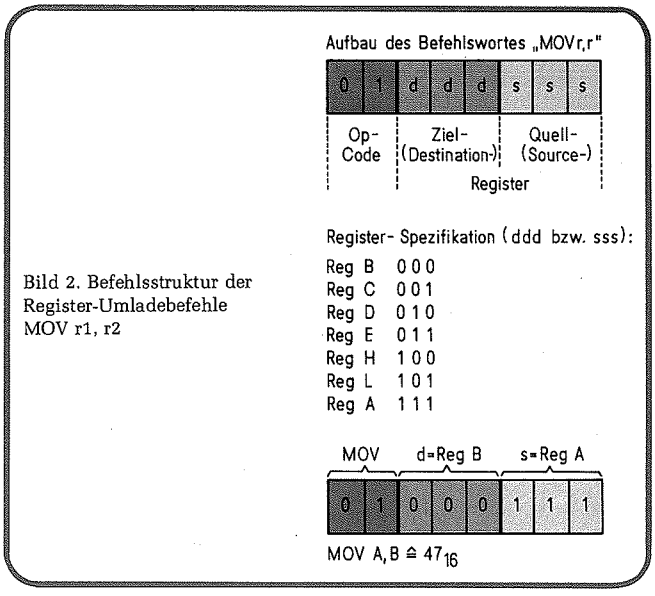


Bild 2. Befehlsstruktur der Register-Umladebefehle MOV r1, r2

stand direkt als Rücksprungadresse gerettet werden. Würde dagegen das Erhöhen des Programmzählerstandes erst parallel zur Befehlsausführung erfolgen, müßte bei einem solchen Programmsprung das Hochzählen separat erfolgen, um die spätere Rücksprungadresse bereitzustellen.

4.2 *Ausführungszyklus*

Das im Befehlshalteregister abgelegte Befehlswort wird durch die interne Decodierung entschlüsselt, und die Ablaufsteuerung erzeugt als Folge davon diejenigen Steuersignale, die zur Ausführung des jeweiligen Befehls erforderlich sind. Dazu gehören beispielsweise das Stellen des Multiplexers vor der Register-Anordnung oder die interne Zwischenspeicherung von Daten und Adressen. Außerdem wird hierbei, sofern ein Mehrwortbefehl vorliegt, das Auslesen des bzw. der nächsten Befehlsworte vorbereitet.

Wenn in den Bytes 2 und 3 eines Dreiwortbefehls eine 16-bit-Adresse enthalten ist, gelangt diese Information wortweise in die internen Hilfsregister W und Z, um von dort parallel auf den Adreßbus geschaltet zu werden, wenn der Befehl ausgeführt wird. Eine direkte Zugriffsmöglichkeit auf diese beiden Register besteht für den Anwender nicht.

5 Adressierungsarten

Ein Großteil der Computer-Programmierung besteht darin, Daten von einer Stelle im System zur anderen zu transportieren, ohne sie dabei in irgendeiner Form zu verändern. Dies erfordert ebenso wie der Zugriff auf einen Operanden die Angabe eines Ausgangs- bzw. Zielpunktes, was durch unterschiedliche Adressierungsarten geschehen kann. Es lassen sich insgesamt zehn verschiedene Möglichkeiten der Adressierung unterscheiden [8], die hier aus Gründen der Übersichtlichkeit in vier Gruppen zusammengefaßt werden:

- Unmittelbare Adressierung,
- Direkte Adressierung,
- Register-Adressierung,
- Indirekte Adressierung.

5.1 *Unmittelbare Adressierung*

Hierbei enthält die Instruktion selbst die Daten, die weiterverarbeitet werden sollen; diese sind als zweites bzw. als zweites und drittes Byte Bestandteil des Befehls, der zur Kennzeichnung der unmittelbaren Adressierung die Angabe „immediate" enthält.

Der Befehl „MVI E, 4D" (Zweiwortbefehl: 1E 4D) bewirkt beispielsweise das Laden des Registers E mit dem 8-bit-Operanden 4D; der Befehl selbst enthält den Operanden und weist damit unmittelbar auf ihn hin.

Entsprechendes gilt für 16-bit-Operanden, die in der Regel als Adresse verarbeitet werden. Der Befehl „LXI H, 004D" (Dreiwortbefehl: 21 00 4D) beispielsweise lädt das Registerpaar H und L mit der Information 4D und 00. Es ist hierbei zu beachten, daß das untere Adreßbyte (00) im Befehl vor dem oberen Adreßbyte (4D) abgespeichert wird. Diese Aufteilung wird im 8080-System bei der Handhabung von Adressen generell angewandt, obwohl sie der sinnfälligen Schreibweise genau entgegensteht. Dies ist in der Praxis nicht nur ein Schönheitsfehler, sondern ein nicht zu übersehender Mangel, der um so schwerer wiegt, als sich weitere Prozessoren daran orientieren, um 8080-kompatibel zu sein (z. B. 8085, Z-80). Als Begründung hierfür ist vom Ersthersteller zu erfahren, daß sich dadurch der interne Hardware-Aufbau vereinfacht, was natürlich kein stichhaltiges Argument darstellt.

5.2 Register-Adressierung

Im 8-bit-Befehlswort selbst ist das Quell- bzw. Zielregister (oder beide) spezifiziert, das einen Operanden bereitstellt bzw. aufnimmt. So erfolgt beispielsweise im oben angeführten Befehl „MVI E, 4D" die Adressierung des Registers E durch das erste Byte des Befehls, wofür auch die Bezeichnung „implizit" als im Befehl selbst enthalten üblich ist.

5.3 Direkte Adressierung

Diese Adressierungsart erfordert generell einen Dreiwortbefehl, bei dem die Bytes 2 und 3 eine Adresse enthalten, unter der ein Operand abgelegt bzw. von der ein Operand geholt werden soll. Der Befehl „STA 0206" (Dreiwortbefehl: 32 06 02) führt beispielsweise dazu, daß der Inhalt des Akkumulators als Operand behandelt und unter der Adresse 0206 abgelegt wird; die Speicheradresse wird *direkt* angegeben.

Tabelle 4. Programmbeispiel zur Demonstration der Stack-Arbeitsweise

	Programm (im RAM)			Arbeitsregister (in CPU)				Stack-Register (im RAM)		
Befehl	Adresse	Inhalt	Assembler-code	(Reg A)	(Reg B)	(Reg C)	(SP)	(8300)	(82FF)	(82FE)
1	8200 8201 8202	31 00 83	LXI SP, 8300	XX	XX	XX	8300	XX	XX	XX
2	8203 8204	3E 4D	MVI A, 4D	4D	XX	XX	8300	XX	XX	XX
3	8205	47	MOV B, A	4D	4D	XX	8300	XX	XX	XX
4	8206	3C	INR A	4E	4D	XX	8300	XX	XX	XX
5	8207	3C	INR A	4F	4D	XX	8300	XX	XX	XX
6	8208	4F	MOV C, A	4F	4D	4F	8300	XX	XX	XX
7	8209	C5	PUSH B	4F	4D	4F	82FE	XX	4D	4F
8	820A	AF	XRA A	00	4D	4F	82FE	XX	4D	4F
9	820B	47	MOV B, A	00	00	4F	82FE	XX	4D	4F
10	820C	4F	MOV C, A	00	00	00	82FE	XX	4D	4F
11	820D	C1	POP B	00	4D	4F	8300	XX	4D	4F

5.4 Indirekte Adressierung

Bei dieser Adressierungsart ist die Adresse für einen Operanden in einem Registerpaar enthalten, auf das der Befehl hinweist. Der Befehl „STAX B" beispielsweise bewirkt, daß der Akkumulator-Inhalt in derjenigen Speicherstelle abgelegt wird, deren Adresse zu diesem Zeitpunkt im Registerpaar B steht; dabei enthält das Register B das obere und Register C das untere Adreßbyte.

6 Stack-Operation

In [1] ist bereits die Arbeitsweise der Stack-Register in einem 8080-System beschrieben worden. Diese Register sind im Unterschied zu anderen Prozessoren nicht Bestandteil der Zentraleinheit, sondern sie werden durch RAM-Speicherstellen im Arbeitsspeicher gebildet. Die Spitze des Register-Stapels wird durch das Initialisieren des Stack-Pointers SP definiert, in den die entsprechende Adresse, beispielsweise mit Hilfe des Befehls „LXI SP, addr", eingeschrieben wird.

Zur Demonstration dieser Arbeitsweise dient das Programmbeispiel der *Tabelle 4*; es zeigt, wie der Stack-Pointer als Adreßregister für die zugehörigen RAM-Speicherstellen fungiert, die den Stapelspeicher bilden. Das Einschreiben in den Stack und das Auslesen von dort geschieht immer Zwei-Wort-weise, um damit 16-bit-Adressen handhaben zu können; dies ist für die Speicherung und den Abruf von Rücksprungadressen von Bedeutung.

Im vorliegenden Beispiel werden nach der Initialisierung des Stack-Pointers (Befehl 1) die Register B und C mit der Information 4D bzw. 4F geladen (Befehle 2...6), was über den Umweg des Akkumulator-Ladens (Befehl 2) und -Hochzählens (Befehle 4 und 5) geschieht. Durch den Befehl „PUSH B" werden die Inhalte des Registerpaares B und C im Stack abgelegt, und zwar in diejenigen beiden Speicherstellen, die unter der aktuellen Adresse des SP liegen (Befehl 7). Im Verlauf der Befehlsausführung wird der Stack-Pointer um Zwei erniedrigt, um auf diese Weise die neue Spitze des Stapelspeichers zu adressieren.

Zur Verdeutlichung der Rückspeicherung werden die Register A...C erst gelöscht (Befehle 8...10), bevor mit der POP-B-Instruktion das Registerpaar B und C geladen wird; hierein gelangen jetzt die Inhalte derjenigen Speicherstellen, die durch (SP) und (SP + 1) adressiert werden (Befehl 11). Im Verlauf der Befehlsausführung wird der Stack-Pointer um Zwei hochgezählt, um nach dem Auslesen wiederum die aktuelle Spitze des Stapelspeichers zu adressieren.

Literatur

[1] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (II). ELEKTRONIK 1977, H. 6, S. 64...70.

[2] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (III). ELEKTRONIK 1977, H. 7, S. 53...58.

[3] *Gößler, R.*: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43 u. 71.

[4] Mikroprozessoren und Mikrocomputer. Kursunterlagen 102 der Firma Integrated Computer Systems.

[5] *Gößler, R.* und *Schwerte, J.*: Auf dem Weg zur Mikroprozessor-Praxis. ELEKTRONIK 1976, H. 3, S. 74...86.

[6] *Gößler, R.*: Mikrocomputer steuert Zündfunkeneinstellung beim Auto. ELEKTRONIK 1977, H. 4, S. 48...51.

[7] Self-Study Microcomputer Hardware/Software Training Course. Kurs 126 der Firma Integrated Computer Systems.

[8] Mikroprozessor-Software: Hilfsmittel und Techniken. Kursunterlagen 156 der Firma Integrated Computer Systems.

Dipl.-Ing. Reinhard Gößler

Einführung in die Mikrocomputer-Programmierung (V)

Zu Beginn dieser Beitragsreihe wurden nach allgemeinen Grundlagen die Mikrocomputer-Hardware und das speziell dafür geschriebene Monitorprogramm der amerikanischen Lehrfirma Integrated Computer Systems vorgestellt [1, 2]; die daran anschließende Beschreibung des Befehlssatzes für den Mikroprozessor 8080 [3] vervollständigte die Grundlagen der Mikrocomputer-Programmierung, und darauf aufbauend bringt der folgende Beitrag konkrete Programmbeispiele, die anhand praktischer Einsatzfälle erläutert werden. Somit bietet diese Beitragsreihe dem Leser die Möglichkeit, sich im Selbststudium in die Mikrocomputer-Technik einzuarbeiten. Die Aufsätze enthalten trotz der Kürze des gesamten Lehrgangs alle für die praktische Arbeit erforderlichen Informationen.

1 Programmbeispiele

Nach Vorstellung der Mikrocomputer-Hardware und der dazugehörenden Betriebssoftware als grundlegende *Hilfsmittel* für die Programmierung geht der folgende Beitrag auf einige praktische Programmbeispiele ein. Diese sind nicht als ergänzender Anhang zu verstehen, sondern sie verfolgen drei wesentliche Ziele: Erstens geht oftmals nur aus dem praktischen Anwendungsbeispiel die genaue Auswirkung bestimmter Befehle hervor, wie etwa die Ein- und Ausgabe von Daten mit entsprechender Vorbereitung des E/A-Bausteins. Zweitens muß der Neuling auf diesem Gebiet erst einige grundlegende Techniken und Kniffe kennenlernen, ehe er darauf aufbauend eigene Programme realisieren kann; als Beispiel hierfür folgt eine modular aufgebaute Zeitverzögerung, die von einem entsprechenden Programm ausgeführt wird. Und schließlich zeigt ein Programmbeispiel für den direkten Speicherzugriff (DMA = *direct memory access*), in welch engem Maße Hard- und Software beim Mikrocomputer-Einsatz miteinander verbunden sind.

Die Beispiele sind derart gewählt worden, daß sie einerseits die Grundelemente für die Mikrocomputer-Handhabung vervollständigen und andererseits effektvolle Anwendungen zeigen. Hier standen didaktische Gesichtspunkte im Vordergrund, um eine Motivation für die weitergehende Beschäftigung zu bieten; dies geht natürlich auf Kosten einer optimalen Computer-Auslastung, die in der Lern- und Ausbildungsphase im Hintergrund bleibt.

Alle Programme sind in der hier dokumentierten Form für das in [1] vorgestellte Mikrocomputer-Trainings-System auf der Basis des Mikroprozessors 8080A geschrieben worden. Sie können selbstverständlich auch auf anderen 8080-Systemen laufen, wenn die zur Anpassung erforderlichen Software-Modifikationen berücksichtigt werden. Darüber hinaus lassen sich diese Beispiele natürlich auch auf andere Mikrocomputer umsetzen, um dort dieselben Funktionen auszuführen. Allerdings sind weder Autor noch Redaktion zeitlich dazu in der Lage, Hilfestellung bei der Entwicklung von Anwenderprogrammen zu geben. Hier können im allgemeinen die zuständigen Vertriebsfirmen eine Brücke schlagen.

2 Direkter Speicherzugriff (DMA)

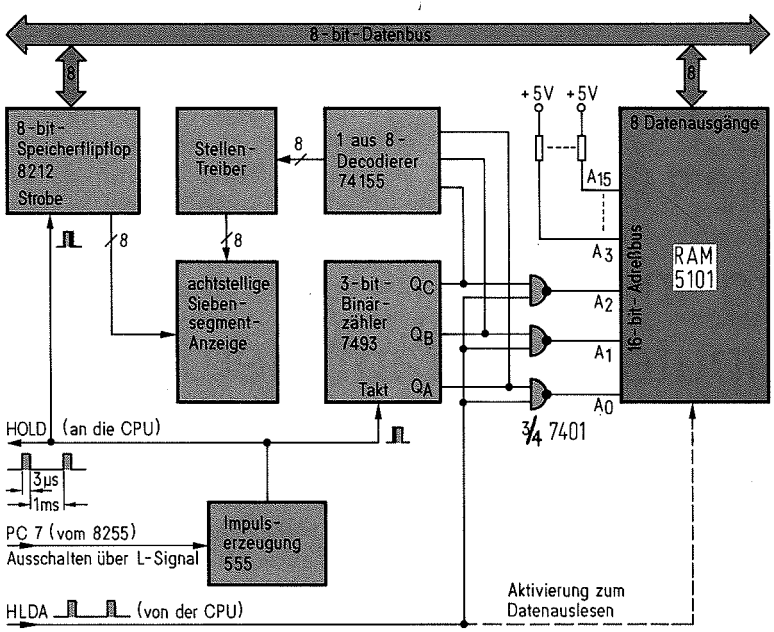
Die direkte Zugriffsmöglichkeit zum Arbeitsspeicher des Computers besitzt in der Praxis eine große Bedeutung, weil dann der Datenaustausch (nahezu) ohne Belastung der CPU und mit maximal möglicher Übertragungsgeschwindigkeit erfolgen kann; die Übertragungsrate wird hierbei nur durch die Zugriffszeit der verwendeten Speicher bestimmt. Zur Abwicklung des Datenverkehrs im DMA gibt es bereits intelligente Steuerbausteine [5], doch geht aus der hier betrachteten einfachen Hardware-Lösung der prinzipielle Funktionsablauf noch deutlicher hervor.

Bei der Betrachtung der Prozessor-Steuerleitungen in [1] ist bereits gesagt worden, daß sich der direkte Speicherzugriff beim MTS über die Leitungen HOLD und HLDA abspielt; nach Aktivierung des HOLD-Eingangs durch einen H-Impuls stoppt der Prozessor seine augenblickliche Aktivität am Ende des laufenden Maschinenzyklus', also noch vor Beendigung des gerade ausgeführten Befehls. Gleichzeitig schaltet er Daten- und Adreßbus in den hochohmigen Zustand, um die Systemverwaltung an eine externe Stelle zu übergeben; diesen Zustand zeigt die Zentraleinheit durch das HLDA-Signal (HOLD-ACKNOWLEDGE) an.

2.1 Hardware-Realisierung

Die im Mikrocomputer-Trainings-System (MTS) vorgesehene Hardware ist so aufgebaut, daß sie direkt auf die obersten acht RAM-Speicherstellen zugreifen kann. Der Inhalt dieser Speicherstellen wird ausgelesen und separat zwischengespeichert und dient dann zur Ansteuerung der achtstelligen Anzeige. Demzufolge legen die Informationen in diesen RAM-Zellen fest, welche Segmente in der Anzeige aufleuchten, um die gewünschte Darstellung zu erzielen; die Aktivierung der einzelnen Stellen erfolgt sequentiell im Multiplex-Betrieb.

Auslösende Stelle für den direkten Speicherzugriff ist ein freilaufender Impulsgenerator, der mit einer Wiederholrate



◀Bild 1. Blockdarstellung der Hardware für den direkten Speicherzugriff (DMA)

von ca. 1 ms Impulse erzeugt, die etwa 3...5 µs lang sind (*Bild 1*). Ansteigende Flanke, H-Zustand und abfallende Flanke dieser Impulse bewirken jeweils separate Reaktionen, die im Zusammenspiel zur Aktivierung der Anzeige führen. Es sei nochmals herausgestellt, daß diese Impulserzeugung ohne Koordination mit der Zentraleinheit arbeitet, daß ihre Impulse also zu einem beliebigen Zeitpunkt im Arbeitsablauf der Zentraleinheit auftreten.

Den zentralen Block in Bild 1 bildet der RAM, der an den Adreß- und den Datenbus angeschlossen ist. Zur Vereinfachung der Darstellung wird der Arbeitsspeicher hier als eine Einheit angesehen, die über 16 Adreßeingänge und 8 Datenausgänge verfügt; der detaillierte Schaltungsaufbau ist in den *Bildern 2, 6* und *7* in [1] zu finden.

Mit der *positiven Flanke des DMA-Impulses* wird ein 3-bit-Binärzähler weitergezählt, dessen drei Ausgänge zyklisch umlaufend die Binärinformationen 000, 001...110, 111 aufweisen. Dieser Zählerstand gelangt gleichzeitig an einen Decodierer und über Open-Collector-Gatter an die untersten drei Bits des Adreßbusses.

Der *H-Zustand des DMA-Impulses* bringt die CPU in den HOLD-Mode, in dem Daten- und Adreßbus des Systems von außen zugänglich sind; entsprechende Pull-up-Widerstände am Adreßbus sorgen dafür, daß dann alle Adreßbits auf HIGH-Potential liegen, entsprechend der sedezimalen Adresse FFFF. Mit dem CPU-Rückmeldesignal HLDA wird nun der Stand des Binärzählers an die untersten Adreßbits geschaltet, so daß in Abhängigkeit vom aktuellen Zählerstand eine der acht obersten RAM-Stellen adressiert wird. Parallel dazu erfolgt die Aktivierung des RAMs, um den Inhalt der adressierten Speicherstelle auszugeben und auf dem Datenbus bereitzustellen.

Mit der *fallenden Flanke des DMA-Impulses* übernimmt das 8-bit-Speicherflipflop 8212 die Information vom Datenbus. Gleichzeitig setzt die Zentraleinheit ihre gestoppte Aktivität fort und nimmt das HLDA-Signal zurück. Von diesem Augenblick an geht die Busverwaltung wieder auf die CPU über, und bis zum Eintreffen des nächsten DMA-Impulses bleibt diejenige Stelle in der Anzeige aktiviert, die von dem vorgeschalteten Decodierer in Übereinstimmung mit der zugehörigen RAM-Zelle angesprochen wird.

In zyklischer Folge wiederholt sich dieser Ablauf, so daß nacheinander die RAM-Speicherstellen FFF8...FFFF ausgelesen werden. Mit dem in [1] geschilderten und dort in Bild 8 dargestellten Zusammenhang folgt, daß diese Adressen im MTS auch als 83F8...83FF dargestellt werden können; diese Zuordnung wird bei den nachfolgenden Programmbeispielen zugrunde gelegt.

2.2 Programmierung der Anzeige

Mit der eben beschriebenen Anordnung ist es möglich, jede gewünschte Kombination von Segmenten zum Leuchten zu bringen. Der Hardware-Aufbau ist dergestalt realisiert worden, daß die RAM-Stelle mit der Adresse 83F8 der Anzeigenstelle links außen (Digit 1) zugeordnet ist; die nächsten RAM-Zellen korrespondieren entsprechend mit den Digits 2...8 (*Bild 2*).

In jedem dieser acht Bytes steuern die einzelnen Bits, wenn sie auf HIGH gesetzt sind, die zugehörigen Segmente an, so wie es in *Bild 3* schematisch gezeichnet ist. Wenn beispielsweise im RAM unter der Adresse 83F9 die sedezimale Information 06 steht, führt das zur Darstellung einer „1" (Segmente b und c leuchten) in der zweiten Anzeigenstelle von links (entsprechend der Adresse 83F9). Allerdings genügt es hierzu nicht, diese Information „06" mit Hilfe des Monitors in die gewünschte Speicherstelle zu laden, wie es beispielsweise in [2] beschrieben wurde; denn nach diesem Ladevorgang würde der Monitor wieder die Steuerung über das gesamte System übernehmen und dabei Speicheradressen und deren Inhalt zur Anzeige bringen. *Die vom Anwender in diesen RAM-Bereich geladenen Informationen erscheinen nur dann in der Anzeige, wenn der Monitor anschließend nicht wieder aktiviert wird.* Das kann man beispielsweise dadurch erreichen, daß im Anschluß an ein entsprechendes Ladeprogramm vom Prozessor eine Endlos-Schleife durchlaufen wird oder daß man die Zentraleinheit mit dem HALT-Befehl stoppt. In beiden Fällen springt der Prozessor nach Abarbeitung des Anwender-Ladeprogramms nicht zurück ins Monitorprogramm, und die gewünschte Darstellung erscheint in der Anzeige.

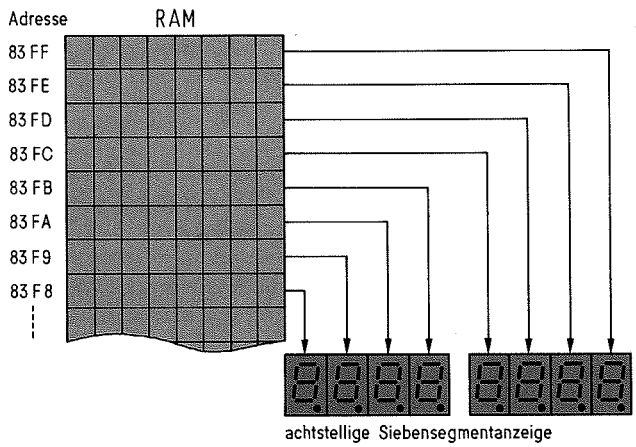
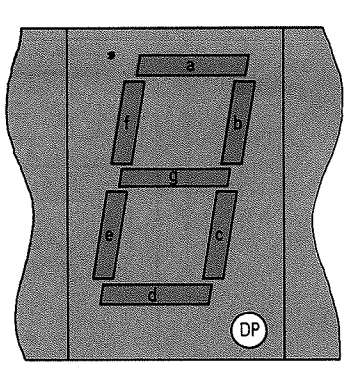
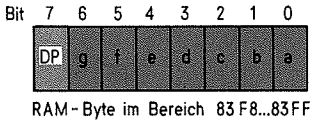


Bild 2. Zuordnung der obersten acht RAM-Speicherstellen zu den einzelnen Digits der Siebensegmentanzeige





RAM-Byte im Bereich 83F8...83FF

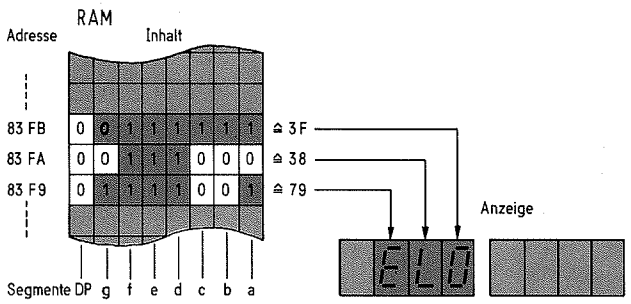


Bild 4. RAM-Programmierung zur Darstellung der Buchstaben ELO in der Siebensegmentanzeige

◀ Bild 3. Zuordnung der einzelnen Bits zu den Segmenten der Anzeige

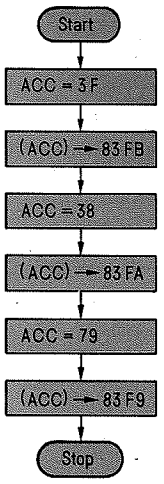


Bild 5. ▶
Flußdiagramm zum Programmbeispiel aus Bild 4

Um dies an einem praktischen Beispiel zu demonstrieren, soll ein kurzes Programm geschrieben werden; es soll in der Lage sein, die Buchstaben „ELO" in den Stellen 2, 3 und 4 der Anzeige erscheinen zu lassen. Aus den bisher geschilderten Zusammenhängen ergibt sich die erforderliche Zuordnung der RAM-Zellen 83F9, 83FA und 83FB mit den entsprechenden Inhalten 79, 38 und 3F (*Bild 4*); das Löschen der gesamten Anzeige vor dem Neueinschreiben besorgt der Monitor automatisch, so daß der Anwender hierfür nicht zu sorgen braucht. Das in *Tabelle 1* aufgeführte und in *Bild 5* dargestellte Programm lädt auf dem Umweg über den Akkumulator die RAM-Speicherstellen mit den in Bild 4 ermittelten Informationen; anschließend stoppt das Programm (Befehl 7), ohne daß der Monitor dadurch wieder die Systemsteuerung übernimmt, und in der Anzeige erscheinen an der gewünschten Stelle die jeweiligen Buchstaben. Die Programmausführung muß hierbei im AUTO-Mode erfolgen, weil im Einzelschrittbetrieb der Monitor wieder auf die Anzeige zugreifen und dabei die entsprechenden RAM-Zellen überschreiben würde.

Als Übungsbeispiel bietet sich nun die Programmierung anderer Initialien an beliebiger Stelle der Anzeige an, um daran die Handhabung des Systems kennenzulernen. Das Programmieren einer Laufschrift, was prinzipiell selbstverständlich auch möglich ist, soll noch so lange zurückgestellt werden, bis weitere Programmtechniken bekannt sind.

Tabelle 1. Programmierung der Buchstaben „ELO" in der Siebensegmentanzeige

Befehl	RAM-Adresse	RAM-Inhalt	Label	Assembler-code	Kommentar
1	8200 8201	3E 3F	DISPL	MVI A, 3F	Konstante ‚3F' für ‚0' in Reg A laden
2	8202 8203 8204	32 FB 83		STA 83FB	(ACC) nach 83FB abspeichern (4. Anzeige-Digit)
3	8205 8206	3E 38		MVI A, 38	Konstante ‚38' für ‚L' in Reg A laden
4	8207 8208 8209	32 FA 83		STA 83FA	(ACC) nach 83FA abspeichern (3. Anzeige-Digit)
5	820A 820B	3E 79		MVI A, 79	Konstante ‚79' für ‚E' in Reg A laden
6	820C 820D 820E	32 F9 83		STA 83F9	(ACC) nach 83F9 abspeichern (2. Anzeige-Digit)
7	820F	76		HLT	Stoppen der CPU

3 Verarbeitung von Unterprogrammen

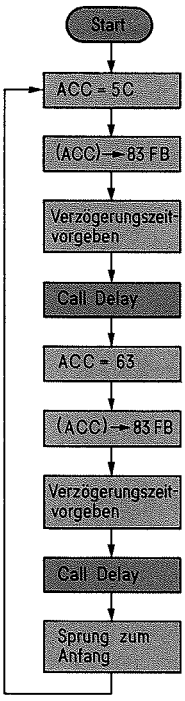
Am Beispiel eines einfachen Programms werden nachfolgend die fünf wesentlichen Elemente der Mikrocomputer-Programmierung vorgestellt:

- Prinzipielle Vorgehensweise
- Verarbeitung von Unterprogrammen
- Verschachtelte Schleifen
- Bedingte Sprünge
- Programmierbare Zeitverzögerungen

Vordergründig ist deshalb nicht der Effekt zu sehen, den das Programmbeispiel bewirkt; von übergeordneter Bedeutung ist vielmehr die prinzipielle Vorgehensweise bei der Problemrealisierung. Die dabei eingesetzten Programmtechniken sind Standardfunktionen, die gewissermaßen als Grundbegriffe zum Handwerkszeug des Programmierers gehören.

Es soll ein Programm geschrieben werden, das zu einem Blinkeffekt in der Anzeige führt; die Blinkfrequenz soll dabei über eine programmierbare Zeitverzögerung in Form eines Unterprogramms bestimmt werden. Das Programmbeispiel wird in der vierten Anzeigenstelle von links eine kleine Null zwischen der unteren und oberen Position hin- und herbewegen, d. h., daß abwechselnd die Segmente c, d, e und g bzw. a, b, g und f aufleuchten. Entsprechend den in den Bildern 2 und 3 dargestellten Zusammenhängen wird das vierte Digit der Anzeige über das RAM-Wort 83FB angesprochen; die kleine Null in der unteren Position erscheint, wenn in dieser RAM-Stelle die sedezimale Information „5C" steht, und dementsprechend führt die Information „63" zur Anzeige einer kleinen Null in der oberen Position. Zwischen dem Umschalten der Anzeige liegt eine vorgebbare Verzögerungszeit, die die Wiederholrate des Ablaufs bestimmt.

Bild 6 zeigt das Flußdiagramm hierzu, das als Endlosschleife aufgebaut ist. Nach Vorgabe der gewünschten Verzögerungszeit erfolgt in den Blöcken „CALL DELAY" der



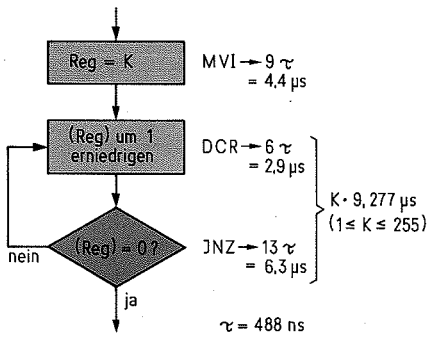


Bild 7. Grundbaustein einer softwaremäßigen Zeitverzögerung

◀ Bild 6. Flußdiagramm für eine blinkende Anzeige

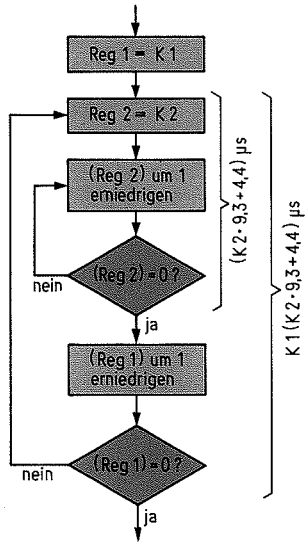


Bild 8. Verschachtelung zweier Zählschleifen zur Erzielung längerer Laufzeiten

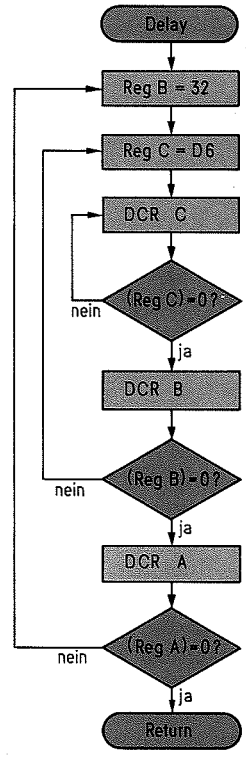


Bild 9. ▶ Flußdiagramm zum Unterprogramm DELAY

Aufruf des Unterprogramms DELAY, das eine definierte Laufzeit besitzt, die mit dem vom Programm vorgegebenen Wert multipliziert wird. Dieses Unterprogramm wird anschließend von der Speicherstelle 8300 an abgelegt werden, so daß es über den Dreiwortbefehl CALL 8300 (CD 00 83) aufgerufen wird. Dabei rettet die Zentraleinheit automatisch die Rücksprungadresse, zu der sie nach Abarbeitung des Unterprogramms zurückkehrt; dieser Rücksprung erfolgt, wenn am Ende des Unterprogramms der Befehl „RETURN" (RET) erreicht wird.

3.1 Programmierbare Zeitverzögerungen

Der prinzipielle Aufbau einer softwaremäßigen Zeitverzögerung geht aus *Bild 7* hervor. Nach Laden eines Registers mit dem Anfangswert K wird der Registerinhalt fortwährend um Eins erniedrigt, bis er auf Null heruntergezählt ist. Mit der Ausführungszeit für jeden Befehl und der Anzahl K der Schleifendurchläufe läßt sich die Laufzeit t dieses Programmteils ermitteln:

$$t = K \cdot 9{,}277\ \mu s + 4{,}4\ \mu s \qquad (1)$$

Berücksichtigt man weiter, daß ein 8-bit-Register als Zähler dienen soll, kann K im Bereich von 1...255 liegen, und für die Laufzeit t ergibt sich ein Bereich von 13,7 µs...2,37 ms, gestuft in Vielfachen von 9,277 µs.

Da die hiermit erreichbare Verzögerungszeit noch unter der Wahrnehmbarkeitsgrenze des Menschen liegt, ist nach Maßnahmen zu suchen, die diese Zeit verlängern. Ein möglicher Ausweg wäre der Einsatz eines Registerpaares als 16-bit-Zähler, bei dem man die Konstante K maximal als 65535 vorgeben könnte; damit wäre dann eine Verzögerungszeit von 608 ms erreichbar. Dies läßt sich nach dem Schema von Bild 7 allerdings nicht verwirklichen, weil es im Befehlssatz keine Nullabfrage für ein Registerpaar gibt. Deshalb ist es zweckmäßig, die Schleife aus Bild 7 in eine übergeordnete Zählschleife einzubetten (*Bild 8*). Auf diese Weise erreicht man eine Multiplikation der Basiszeit mit dem Faktor K 1, womit ebenfalls eine maximale Laufzeit von 608 ms erzielt werden kann.

Bei geeigneter Wahl der Konstanten K 1 und K 2 läßt sich eine „glatte" Basiszeit von beispielsweise 100 ms realisieren. Durch eine einfache Ergänzung erreicht man die Multiplikation dieses Wertes mit derjenigen Zahl, die beim Sprung ins Unterprogramm im Akkumulator steht. Das Unterprogramm wird damit zu einem modularen Gebilde, dessen Eigenschaften (gesamte Laufzeit) durch einen Parameter

Tabelle 2. Unterprogramm DELAY mit vorwählbarer Laufzeit
t = (ACC) · 100 ms

Befehl	RAM-Adresse	RAM-Inhalt	Label	Assembler-code	Kommentar
1	8300 8301	06 32	DELAY	MVI B, 32	Konstante $K1 = 50_{10} \triangleq 32_{16}$ in Reg B laden
2	8302 8303	0E D6	LOOPB	MVI C, D6	Konstante $K2 = 214_{10} \triangleq D6_{16}$ in Reg C laden
3	8304	0D	LOOPC	DCR C	(Reg C) erniedrigen
4	8305 8306 8307	C2 04 83		JNZ LOOPC	Sprung zu LOOPC, wenn (Reg C) ≠ 0
5	8308	05		DCR B	(Reg B) erniedrigen
6	8309 830A 830B	C2 02 83		JNZ LOOPB	Sprung zu LOOPB, wenn (Reg B) ≠ 0
7	830C	3D		DCR A	(Reg A) erniedrigen
8	830D 830E 830F	C2 00 83		JNZ DELAY	Sprung zu DELAY, wenn (Reg A) ≠ 0
9	8310	C9	END	RET	sonst: Rücksprung ins Hauptprogramm

Tabelle 3. Programmbeispiel für eine blinkende Anzeige mit Aufruf des Unterprogramms DELAY

Be-fehl	RAM-Adresse	RAM-Inhalt	Label	Assembler-code	Kommentar
1	8200 8201	3E 5C	BLINK	MVI A, 5C	Konstante ‚5C' für untere Null laden
2	8202 8203 8204	32 FB 83		STA 83FB	(ACC) nach 83FB abspeichern (4. Digit v. links)
3	8205 8206	3E 02		MVI A, 02	Zeitverzögerung vorgeben (2mal 100 ms)
4	8207 8208 8209	CD 00 83		CALL DELAY	Unterprogramm DELAY aufrufen
5	820A 820B	3E 63		MVI A, 63	Konstante ‚63' für obere Null laden
6	820C 820D 820E	32 FB 83		STA 83FB	(ACC) nach 83FB abspeichern (4. Digit v. links)
7	820F 8210	3E 04		MVI A, 05	Zeitverzögerungvorgeben (4mal 100 ms)
8	8211 8212 8213	CD 00 83		CALL DELAY	Unterprogramm DELAY aufrufen
9	8214 8215 8216	C3 00 82		JMP BLINK	Sprung zum Anfang

(Akkumulator-Inhalt) bestimmt werden. Dazu müssen die beiden Zählschleifen aus Bild 8 in eine dritte Schleife eingebettet werden, die den Akkumulator-Inhalt leerzählt; für die Konstanten K 1 und K 2 sind die dezimalen Werte 50 bzw. 214 einzusetzen (sedezimal 32 bzw. D6), um eine Basiszeit von 100 ms zu erreichen. Das vollständige Flußdiagramm für dieses Unterprogramm zeigt *Bild 9*, und in *Tabelle 2* ist das zugehörige Maschinenprogramm zusammengestellt.

Um das Programm aus Bild 6 einzugeben, bedient man sich der Sequenz der *Tabelle 3*. In den Befehlen 1 bzw. 5 wird die anzuzeigende Information geladen, um anschließend in diejenige RAM-Speicherstelle transportiert zu werden, die der gewünschten Stelle der Anzeige zugeordnet ist (Befehle 2 bzw. 6). Die Befehle 3 bzw. 7 laden vor dem Aufruf des Verzögerungs-Unterprogramms DELAY diejenige Konstante in den Akkumulator, mit der die Basislaufzeit von 100 ms multipliziert werden soll; die Vorgabe der Zahlen 2 bzw. 4 führt zu einem unsymmetrischen Tastverhältnis mit einer Periodendauer von 0,6 s. Es sei darauf hingewiesen, daß die rechnerisch ermittelte Laufzeit auf dem MTS durch den ständig stattfindenden direkten Speicherzugriff geringfügig verlängert wird.

4 Daten-Ein- und -Ausgabe

Die Kommunikation zwischen Computer und peripheren Bausteinen erfolgt über drei bidirektionale 8-bit-Kanäle (Port A...C), die hardwaremäßig im E/A-Baustein 8255 untergebracht sind. Vor der Ein/Ausgabe muß diesem Baustein mitgeteilt werden, in welcher Betriebsart er arbeiten soll.

Tabelle 4. Bestimmung der Übertragungsrichtung der einzelnen E/A-Kanäle mit Hilfe eines Befehlswortes

E: Eingabe A: Ausgabe

Befehlswort (sedezimal)	Kanal A	Kanal B	Kanal C (Bit 0...3)	Kanal C (Bit 4...7)
80	A	A	A	A
88	A	A	A	E
81	A	A	E	A
89	A	A	E	E
82	A	E	A	A
8A	A	E	A	E
83	A	E	E	A
8B	A	E	E	E
90	E	A	A	A
98	E	A	A	E
91	E	A	E	A
99	E	A	E	E
92	E	E	A	A
9A	E	E	A	E
93	E	E	E	A
9B	E	E	E	E

Dazu dient ein Befehlswort, das vom Akkumulator in ein internes Register des E/A-Bausteins eingeschrieben wird und dort die Betriebsart festlegt. Insgesamt sind 16 verschiedene Betriebsarten möglich, in denen die E/A-Kanäle eingesetzt werden können; Kanal C teilt sich dabei in zwei 4-bit-Kanäle auf, deren Übertragungsrichtung unterschiedlich festgelegt werden kann (*Tabelle 4*).

Das Einschreiben des Befehlswortes erfolgt mit der OUT-Instruktion, gefolgt von einem Steuerbyte, welches dem Steuerregister im E/A-Baustein mitteilt, daß es sich bei dem auszulesenden Akkumulator-Inhalt nicht um Daten, sondern um ein Befehlswort handelt; das Steuerbyte zur Kennzeichnung eines Befehlswortes hat in den Bits 0 und 1 eine „1" (*Bild 10*).

Dieselbe OUT-Instruktion wird eingesetzt, wenn der Akkumulator-Inhalt über einen E/A-Kanal ausgegeben werden soll. Nach Festlegung der Betriebsart geht die Datenausgabe so vor sich, daß das Steuerbyte nach dem Befehl OUT mit seinen Bits 0 und 1 einen der Übertragungskanäle A, B oder C anwählt (Bild 10). Der OUT-Befehl ist also ein Zweiwortbefehl, dessen zweites Byte dem E/A-Baustein mitteilt, ob aus dem Akkumulator Daten oder ein Befehlswort ausgelesen werden sollen.

Wenn beispielsweise Kanal A und C in Richtung Ausgabe und Kanal B in Richtung Eingabe geschaltet sein sollen, muß in das Steuerregister des E/A-Bausteins die Information „82" eingeschrieben werden (*Tabelle 5*). Dazu wird zunächst der Akkumulator mit der Konstanten 82 geladen (Befehl 1), und der Ausgabe-Befehl (Befehl 2) sorgt in Verbindung mit dem Steuerbyte in 8203 dafür, daß der Akkumulator-Inhalt als Befehlswort in das Steuerregister eingeschrieben wird (Bits 0 und 1 des Steuerbytes sind HIGH, siehe auch Bild 10).

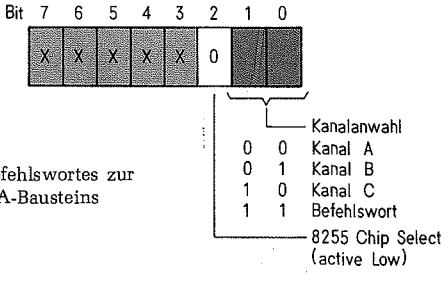


Bild 10. Aufbau des Befehlswortes zur Programmierung des E/A-Bausteins

Tabelle 5. Programmierung des E/A-Bausteins und anschließende Ein- und Ausgabe von Daten

Befehl	RAM-Adresse	RAM-Inhalt	Label	Assembler-Code	Kommentar
1	8200 8201	3E 82	INIT	MVI A, 82	Initialisierung des E/A-Bausteins: Kanäle A und C auf Ausgabe und Kanal B auf Eingabe (Befehlswort 82)
2	8202 8203	D3 03		OUT	
	⋮	⋮		⋮	
3	XXXX XXXX	3E 01	AUSG1	MVI A,01	Konstante ‚01' ins Reg A laden
4	XXXX XXXX	D3 00		OUT PORT A	Ausgabe des (ACC) über Kanal A
	⋮	⋮		⋮	
5	XXXX XXXX	D3 10	AUSG2	OUT PORT C	Ausgabe des (ACC) über Kanal C
6	XXXX XXXX	DB 01	EING1	IN PORT B	Eingabe von Kanal B nach Reg A
	⋮	⋮		⋮	

Nach dieser Initialisierung bleibt der E/A-Baustein so lange in der gewählten Betriebsart, bis ein neues Befehlswort eingeschrieben wird. Die Daten-Ein- und Ausgabe kann nun an beliebiger Stelle im Programm erfolgen, wobei das zweite Byte jedes Ein- bzw. Ausgabebefehls den Übertragungskanal anspricht, der den Datenverkehr mit dem Akkumulator aufnehmen soll.

Aufgrund des Hardware-Aufbaus im MTS muß Bit 2 des Befehlswortes auf LOW sein, um den Baustein 8255 über dessen Chip-select-Eingang zu aktivieren. Abschließend sei noch erwähnt, daß die angesprochenen Ein/Ausgabe-Möglichkeiten nur einen Teil der gesamten Leistungsfähigkeit wiedergeben; es würde aber den hier gesteckten Rahmen sprengen, alle Ein/Ausgabe-Varianten im Detail vorzustellen.

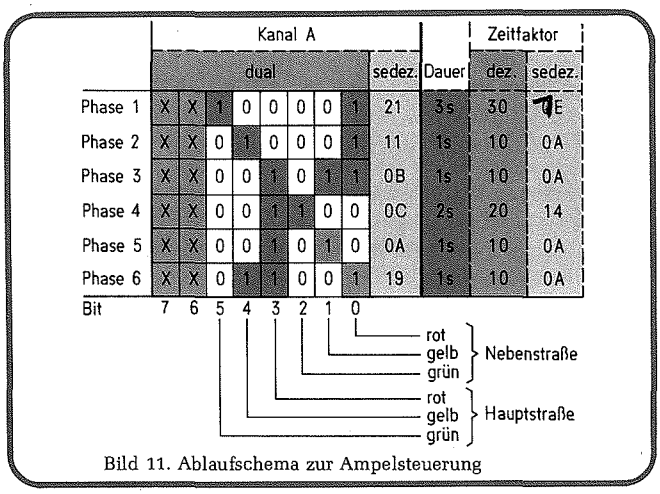


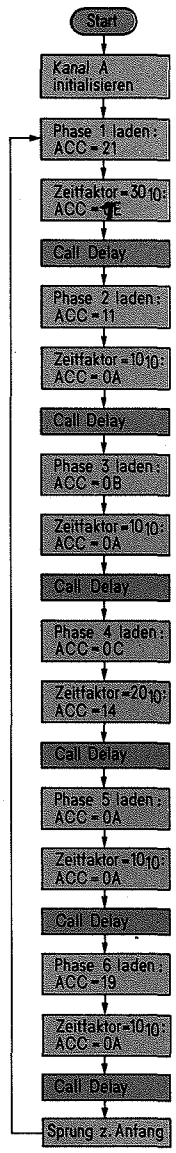
Bild 11. Ablaufschema zur Ampelsteuerung

4.1 Programmbeispiel Ampelsteuerung

Eines der Standardbeispiele bei der Mikrocomputer-Programmierung ist die Lichtzeichensteuerung bei einer Verkehrsampel; da dieses Beispiel schon mit geringem Hardware-Aufwand aufgebaut werden kann und sich außerdem gut zur Demonstration einer einfachen Ablaufsteuerung eignet, soll es hier im Detail vorgestellt werden.

Die Ansteuerung der insgesamt 6 verschiedenen Lämpchen (je drei für die Haupt- und Nebenstraßen) erfolgt von jeweils einem Bit des Ausgabekanals A (*Bild 11*). In der Darstellung sind die einzelnen Phasen aufgeführt, und parallel erscheint die Zeitdauer. Das Programm besteht aus einer einfachen Sequenz, bei der stets zuerst die Steuerinformation für die Lämpchen geladen und ausgegeben wird, gefolgt vom Aufruf des Unterprogramms DELAY zur Erzeugung der gewünschten Verzögerungszeit (*Bild 12*); das vollständige Programm enthält die *Tabelle 6*.

In diesem Programm verdient ein Detail nähere Erwähnung; im Befehl 21 wird der Zeitfaktor 0A (dezimal 10) geladen, obwohl dieser Wert bereits mit dem Befehl 19 in den Akkumulator gebracht wurde. Diese Redundanz sollte unbedingt bestehen bleiben, damit das Programm bei Änderung des Zeitfaktors an dieser Stelle durch Umprogrammieren von nur einer Stelle angepaßt werden kann; das Einfügen eines neuen Befehls dagegen wäre wesentlich aufwendiger.



Zur praktischen Demonstration des Ampel-Beispiels eignen sich Leuchtdioden in den Farben Rot, Gelb und Grün, die zur Ansteuerung separate Stromtreiber benötigen (*Bild 13*). Das Nachvollziehen dieses einfachen Beispiels führt bei dem begrenzten und leicht überschaubaren Umfang zum Vertrautwerden mit der gesamten Materie. Insbesondere eine eventuelle Fehlersuche gestaltet sich hierbei relativ problemlos, so daß es sich empfiehlt, dieses Anwendungsbeispiel wegen des damit verbundenen Lerneffekts selbst nachzuvollziehen.

◀ Bild 12. Flußdiagramm zum Programmbeispiel Ampelsteuerung

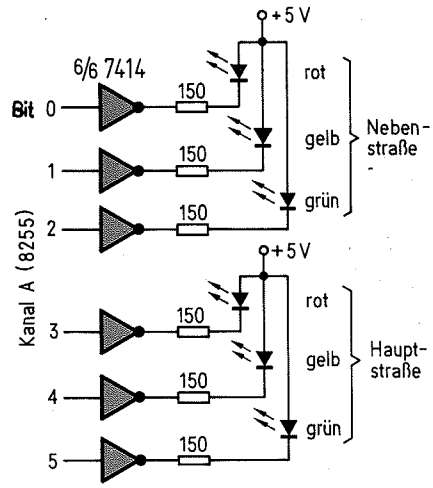


Bild 13. Hardware-Ergänzung zur Ansteuerung der Leuchtdioden

Tabelle 6. Programmbeispiel Ampelsteuerung

Befehl	RAM-Adresse	RAM-Inhalt	Label	Assemblercode	Kommentar
1	8200 8201	3E 82	AM- PEL	MVI A, 82	Befehlswort für E/A-Baustein laden
2	8202 8203	D3 03		OUT	Befehlswort an E/A-Baustein ausgeben
3	8204 8205	3E 21	PHAS1	MVI A, 21	Byte für Phase 1 laden
4	8206 8207	D3 00		OUT PORT A	(ACC) ausgeben
5	8208 8209	3E 0E		MVI A, 0E	Zeitfaktor laden (30mal 100 ms)
6	820A 820B 820C	CD 00 83		CALL DELA Y PORT A	Unterprogramm aufrufen
7	820D 820E	3E 11	PHAS2	MVI A, 11	Byte für Phase 2 laden
8	820F 8210	D3 00		OUT	(ACC) ausgeben
9	8211 8212	3E 0A		MVI A, 0A	Zeitfaktor laden (10mal 100 ms)
10	8213 8214 8215	CD 00 83		CALL DELAY PORT A	Unterprogramm aufrufen
11	8216 8217	3E 0B	PHAS3	MVI A, 0B	Byte für Phase 3 laden
12	8218 8219	D3 00		OUT PORT B	(ACC) ausgeben
13	821A 821B	3E 0A		MVI A, 0A	Zeitfaktor laden (10mal 100 ms)
14	821C 821D 821E	CD 00 83		CALL DELAY PORT A	Unterprogramm aufrufen
15	821F 8220	3E 0C	PHAS4	MVI A, 0C	Byte für Phase 4 laden
16	8221 8222	D3 00		OUT	(ACC) ausgeben
17	8223 8224	3E 14		MVI A, 14	Zeitfaktor laden (20mal 100 ms)
18	8225 8226 8227	CD 00 83		CALL DELAY	Unterprogramm aufrufen
19	8228 8229	3E 0A	PHAS5	MVI A, 0A	Byte für Phase 5 laden
20	822A 822B	D3 00		OUT PORT A	(ACC) ausgeben
21	822C 822D	3E 0A		MVI A, 0A	Zeitfaktor laden
22	822E 822F 8230	CD 00 83		CALL DELAY	Unterprogramm aufrufen
23	9231 8232	3E 19	PHAS6	MVI A, 19	Byte für Phase 6 laden
24	8233 8234	D3 00		OUT PORT A	(ACC) ausgeben
25	8235 8236	3E 0A		MVI A, 0A	Zeitfaktor laden
26	8237 8238 8239	CD 00 83		CALL DELAY	Unterprogrammn aufrufen
27	823A 823B 823C	C3 00 82	REPT	JMP AMPEL	Sprung zum Anfang

5 Kassetten-Interface

Der Monitor besitzt zwei Unterprogramme SEROT und SERIN, mit denen die serielle Aus- bzw. Eingabe von Daten möglich ist. Durch eine geringfügige Ergänzung kann man mit Hilfe dieser Programme Speicherbereiche auf ein Magnetbandgerät überschreiben und umgekehrt, so daß auf diese Weise ein einfacher und preisgünstiger externer Massenspeicher zur Verfügung steht.

5.1 Datenformat

Die asynchrone Datenübertragung auf einer einzelnen Leitung erfolgt byteweise, beginnend mit Bit 0 eines jeden Wortes. Vor jedem Byte wird ein Startbit (LOW) eingefügt, und nach der Übertragung von Bit 7 folgen drei Stopbits (HIGH). Start- und Stopsignal dienen zur Synchronisation bei der empfangenen Stelle; sie werden dort wieder abgetrennt, um die bloße 8-bit-Information zu erhalten und ablegen zu können (*Bild 14*).

Um die Kompatibilität mit der Fernschreibmaschine herzustellen, erfolgt die Übertragung mit 110 bit/s; daraus ergibt sich eine Übertragungsdauer von 109 ms pro Byte, da jedes Datenwort einschließlich der Start- und Stopbits aus zwölf Bits besteht. Diese Übertragungsgeschwindigkeit ist äußerst niedrig und reicht allenfalls für kurze Programmaufzeichnungen aus; bereits die Übertragung von 1 KByte dauert nahezu zwei Minuten, so daß man bei umfangreicheren Datenblöcken auf schnellere Peripheriegeräte ausweichen wird [4].

5.2 Serielle Ausgabe

Das Programm SEROT benutzt Bit 0 von Kanal C als seriellen Ausgangskanal, an dem die in Bild 14 oben gezeich-

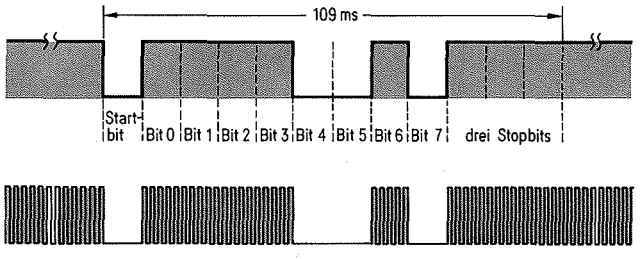


Bild 14. Aufbau des Datenwortes bei der asynchronen seriellen Informationsübertragung

Tabelle 7. Eingabesequenz zum Aufruf des Unterprogramms SEROT

REG	D	01	XXXX	D-01
NEXT		20	XXXX	E-20
NEXT	NEXT	82	XXXX	H-82
NEXT		00		L-00
ADDR		0375	0375	F3
Bandgerät in Stellung „Aufnahme" starten				
RUN				
Datenübertragung				
am Ende erscheint			039E	L-20

nete Digitalinformation ansteht. Dieses Signal wird mit dem Ausgang eines Rechteckoszillators verknüpft, und es entsteht ein getastetes Tonfrequenzsignal, wie es in der unteren Darstellung von Bild 14 zu sehen ist. Das Hinzufügen der Start- und Stopbits übernimmt das Programm selbsttätig. Vor dem Aufruf des Programms, das bei Adresse 0375 beginnt, muß der Anwender lediglich die Startadresse des zu übertragenden Speicherbereichs und die Blocklänge eingeben; die beiden Bytes der Startadresse werden in die Register H und L eingegeben, und die binäre Anzahl der zu übertragenden Worte schreibt man in die Register D und E ein.

Wenn beispielsweise der Speicherbereich von 8200 bis 8320 (= 0120 Worte) ausgegeben werden soll, verfährt man nach der Sequenz der *Tabelle 7*: Die Wortanzahl 0120 wird in das Registerpaar D, und die Startadresse 8200 in das Registerpaar H eingeschrieben. Vor dem Programmstart bei Adresse 0375 ist das aufnahmebereite Bandgerät zu starten, damit zunächst einige Sekunden des ungetasteten Tonfrequenzsignals aufgezeichnet werden. Im Verlauf dieses Vorspanns muß beim späteren Wiedereinlesen das Einleseprogramm SERIN gestartet werden.

Am Ende der Übertragung leuchtet die Anzeige wieder auf, und in den Registern H und L steht die Endadresse des übertragenen Datenblocks, während die Register D und E auf Null leergezählt sind.

5.3 Serielle Eingabe

Zum Einlesen serieller Informationen, die zuvor mit dem Ausgabeprogramm SEROT aufgezeichnet worden sind, dient das Programm SERIN. Bevor man es über Adresse 03B1 aufrufen kann, muß die Startadresse für den einzulesenden Speicherbereich eingegeben werden (*Tabelle 8*); die Endadresse erfährt das Programm beim Einlesen selbst, weil diese zuvor mit ausgegeben worden ist.

5.4 Anpaßschaltung

In *Bild 15* ist eine Anpaßschaltung gezeichnet, die als Interface zwischen den seriellen E/A-Kanälen des Bausteins 8255 und einem Magnetbandgerät dient. Sie besteht aus einem getasteten Oszillator (oben) und einer Hüllkurvengleichrichtung (unten) zur Modulation bzw. Demodulation des Tonfrequenzsignals. Mit zwei integrierten Standardbausteinen und einigen passiven Komponenten läßt sich diese Schaltung problemlos nachbauen.

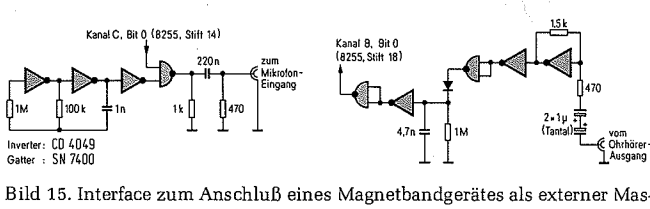


Bild 15. Interface zum Anschluß eines Magnetbandgerätes als externer Massenspeicher

Tabelle 8. Eingabesequenz zum Aufruf des Unterprogramms SERIN

REG	H	82	XXXX	H-82
NEXT		00	XXXX	L-00
ADDR		03B1	03B1	F3
Bandgerät in Stellung „Wiedergabe" starten				
RUN				
Datenübertragung				
am Ende erscheint			03EF	L-20

6 Eigene Programmierpraxis

Ganz bewußt stand im Rahmen dieser Einführungsreihe die enge Verknüpfung zwischen Hard- und Software im Vordergrund. Zum Einsatz eines Mikrocomputers reichen bloße Programmierkenntnisse genauso wenig aus wie die ausschließliche Hardware-Erfahrung.

So ist es auch längst nicht damit getan, die angeführten Beispiele „trocken" zu verstehen, ohne sie in der Praxis Schritt für Schritt nachzuvollziehen; obwohl die Beispiele sehr einfach aussehen, unterläuft bei ihrer Realisierung doch der eine oder andere Fehler, der typisch ist für die praktische Arbeit am Mikrocomputer. Diese ersten Fehler lassen sich aber nur dann rasch erkennen und ausmerzen, wenn das Programm selbst keine Probleme aufwirft.

Die ersten eigenen „Gehversuche" bei der Programmierung verlaufen zweckmäßig so, daß man vorhandene Programme erweitert oder modifiziert, daß man also anfangs nur kleine Schritte unternimmt. So ist beispielsweise folgende Änderung des Programmbeispiels aus Tabelle 3 aufwendiger, als es zunächst den Anschein hat: Wenn sich die Null nicht auf einer Stelle zwischen unterer und oberer Position hin- und herbewegen soll, sondern der Blinkeffekt zwischen zwei verschiedenen Stellen stattfinden soll, genügt nicht die Änderung der Adresse im Befehl 2 bzw. 6; vielmehr muß beim Neueinschreiben in eine andere Anzeigenstelle die Information an der vorherigen Stelle gelöscht werden, um den Blinkeffekt zu erreichen.

Derartige Gedankenfehler treten leicht auf, und sie führen zu dem wesentlichen Grundsatz, den man als obersten Leitgedanken bei der Programmierung vor Augen haben sollte:

● *Sorgfältige, bis ins Detail gehende Planung ist die unerläßliche Voraussetzung für eine erfolgreiche Programmerstellung.*

Natürlich ist die eigene Erfahrung im praktischen Umgang mit dem Computer durch nichts zu ersetzen; und so kann diese Beitragsreihe auch nur eine Hilfestellung bei der Einarbeitung geben. Sie enthält allerdings alle erforderlichen Grundlageninformationen in ausführlicher und umfassender Zusammenstellung, so daß der Praktiker hiermit ein wertvolles Hilfsmittel geboten bekommt.

Literatur

[1] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (II). ELEKTRONIK 1977, H. 6, S. 64...70. Teil I dieser Reihe erschien in H. 4, S. 79...84 und erklärte die Erstellung von Flußdiagrammen und andere Grundlagen.

[2] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (III). ELEKTRONIK 1977, H. 7, S. 53...58.

[3] *Gößler, R.*: Einführung in die Mikrocomputer-Programmierung (IV). ELEKTRONIK 1977, H. 8, S. 67...74.

[4] *Gößler, R.*: Entwicklungshilfsmittel für die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 5, S. 36...43.

[5] *Gößler, R.*: Ein/Ausgabe-Bausteine für Mikroprozessoren. ELEKTRONIK 1976, H. 11, S. 62...68.

[6] Self-Study Microcomputer Hardware/Software Training Course. Einführungslehrgang der Firma Integrated Computer Systems.

[7] Die praktischen Ausbildungen am Mikrocomputer. Kurs 125 der Firma Integrated Computer Systems.

In der Artikelserie „Einführung in die Mikroprozessor-Programmierung" wurde auf die enge Verflechtung von Hardware und Software auf diesem Gebiet hingewiesen. Doch während erfahrungsgemäß dem Schaltungsentwickler die Hardware kaum Schwierigkeiten bereitet, ist die Software zum größten Teil Neuland für ihn. Zur Vertiefung der bereits behandelten Grundlagen betrachtet der folgende Beitrag deshalb den Mikroprozessor anhand anschaulicher Beispiele nur aus der Sicht des Programmierers. Der zugrunde gelegte Prozessor (Serie 6500) ist einer der modernsten auf dem Markt.

Dipl.-Ing. E. Flögel

Mikrocomputer-Programmierpraxis

1 Das Prozessor-System

Der Mikroprozessor stellt im Prinzip nichts anderes als die auf kleinstem Raum untergebrachte Zentraleinheit (CPU) eines Computers dar. Um zu einem Mikroprozessor-System zu gelangen, muß man diese CPU mit einem Speicher für Programm und Daten und einem Ein/Ausgabe-Baustein erweitern (*Bild 1*). Alle drei Bausteine sind durch Daten-, Adreß- und Steuerleitungen miteinander verbunden. Für die Programmierung ist aber die Kenntnis der Hardware nicht unbedingt notwendig. Man benötigt vielmehr ausreichende Information über die logische Struktur der CPU, den Befehlsumfang, die Möglichkeiten der Adressierung und die Ausführungszeiten der einzelnen Befehle.

1.1 Die logische Struktur der CPU

Unter „logischer Struktur" soll hier die Darstellung aller dem Programmierer zugänglichen Teile des Prozessors verstanden werden. Den weiteren Betrachtungen wird das Prozessor-System 6500 von der Firma MOS-Technology zugrunde gelegt (*Bild 2*). Dies bedeutet keine Einschränkung, denn andere Prozessoren sind nach ähnlichen Gesichtspunkten aufgebaut.

Der Akkumulator stellt die zentrale Schnittstelle in der Datenübertragung dar. Sollen z. B. Daten aus einer Speicherzelle in eine andere übertragen werden, so werden diese erst in den Akkumulator übernommen und dann in die neue Zelle geschrieben. Ebenso werden alle arithmetischen Operationen über den Akkumulator ausgeführt. Der Inhalt einer Speicherzelle wird zum Inhalt des Akkumulators addiert und dort als Ergebnis gespeichert. Die beiden Register X und Y, auch Index-Register genannt, können ebenfalls Daten übertragen. Sie dienen aber auch zur Adressenmodifizierung (indizierte Adressierung) oder können als Zählregister verwendet werden. Mit dem doppeltlangen Programmzähler ist es möglich, insgesamt 64 KByte an Speicherzellen zu adressieren. Das Stack-Register stellt den Stapelzeiger (*Stack Pointer*) dar. Es dient zur Adressierung eines besonderen Speicherbereiches, dem Stapelspeicher (*Stack*). Das letzte der Register ist das Prozessorstatus-Register. Die einzelnen Bits dieses Registers werden durch Operationen, deren Ergebnisse oder durch Befehle gesetzt oder gelöscht. Die Bedeutung der einzelnen Bits ist in *Bild 3* dargestellt.

1.2 Der Befehlssatz

Die Befehle des Systems 6500 werden in einer symbolischen Schreibweise angegeben, wie sie beim Schreiben von Programmen angewendet wird, die anschließend mit Hilfe des Assemblers in die Maschinensprache übersetzt werden. Um einen besseren Überblick über den gesamten Befehls-

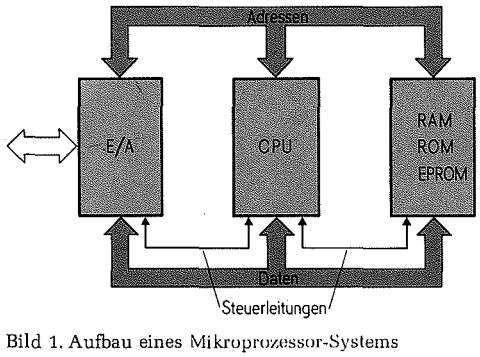


Bild 1. Aufbau eines Mikroprozessor-Systems

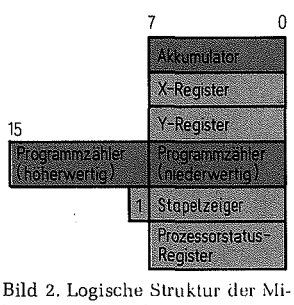


Bild 2. Logische Struktur der Mikroprozessoren des Systems 6500

C	CARRY	= 1	Übertrag aus Bit 7
Z	ZERO	= 1	Ergebnis Null
I	IRQ	= 1	Interrupt über IRQ nicht möglich
D	DECIMAL	= 1	dezimale Arithmetik
B	BRK	= 1	BRK-Befehl ist erfolgt
V	OVERFLOW	= 1	Übertrag aus Bit 6
N	NEGATIV	= 1	Ergebnis negativ

Bild 3. Bedeutung der einzelnen Bits im Prozessorstatus-Register

vorrat zu haben, wird dieser in acht Teile gegliedert (*Tabelle*).

a) Transportbefehle
Z. B. LDA (*Load Accumulator*). Lade den Akkumulator mit dem Inhalt einer Speicherzelle; STX (*Store X-Reg.*) Speichere Inhalt des X-Registers in eine Zelle.

b) Arithmetische Befehle
Z. B. ADC (*Add with Carry*). Addiere zum Inhalt des Akkumulators den Inhalt einer Speicherzelle und den Übertrag. INY (Increment Y-Reg.) Erhöhe den Inhalt des Y-Registers um Eins.

c) Logische Befehle
Z. B. AND. Akkumulator und Speicherinhalt werden durch die UND-Funktion bitweise miteinander verknüpft.

d) Vergleichsbefehle
Z. B. CMP (*Compare Memory with Accumulator*). Es wird

Tabelle. Befehlsvorrat und Adressierungsarten des Prozessorsystems 6500 (X Statusregister wird verändert)

			ADRESSIERUNGSARTEN													Status-register					
Befehle	symb. Code	Wirkung	UNM	ABS	ABS X	ABS Y	S0	S0, X	S0, Y	(IND, X)	(IND, Y)	REL	IND	ACCU	IMPL	N	Z	C	I	D	V
Transport-	LDA	M → A	A9	AD	BD	B9	A5	B5		A1	B1					X	X	–	–	–	–
	LDX	M → X	A2	AE		BE	A6		B6							X	X	–	–	–	–
	LDY	M → Y	A0	AC	BC		A4	B4								X	X	–	–	–	–
	STA	A → M		8D	9D	99	85	95		81	91					–	–	–	–	–	–
	STX	X → M		8E			86		96							–	–	–	–	–	–
	STY	Y → M		8C			84	94								–	–	–	–	–	–
	TAX	A → X													AA	X	X	–	–	–	–
	TAY	A → Y													A8	X	X	–	–	–	–
	TXA	X → A													8A	X	X	–	–	–	–
	TYA	Y → A													98	X	X	–	–	–	–
	TXS	X → S													9A	–	–	–	–	–	–
	TSX	S → X													BA	X	X	–	–	–	–
	PLA	S+1→S, Ms→A													68	X	X	–	–	–	–
	PHA	A→Ms, S–1→S													48	–	–	–	–	–	–
	PLP	S+1→S, Ms→P													28						
	PHP	P→Ms, S–1→S													08	–	–	–	–	–	–
Arithmetische	ADC	A+M+C → A	69	6D	7D	79	65	75		61	71					X	X	X	–	–	X
	SBC	A–M–C → A	E9	ED	FD	F9	E5	F5		E1	F1					X	X	X	–	–	X
	INC	M+1 → M		EE	FE		E6	F6								X	X	–	–	–	–
	DEC	M–1 → M		CE	DE		C6	D6								X	X	–	–	–	–
	INX	X+1 → X													E8	X	X	–	–	–	–
	DEX	X–1 → X													CA	X	X	–	–	–	–
	INY	Y+1 → Y													C8	X	X	–	–	–	–
	DEY	Y–1 → Y													88	X	X	–	–	–	–
Logische	AND	A ∧ M → A	29	2D	3D	39	25	35		21	31					X	X	–	–	–	–
	ORA	A ∨ M → A	09	0D	1D	19	05	15		01	11					X	X	–	–	–	–
	EOR	A ∨̸ M → A	49	4D	5D	59	45	55		41	51					X	X	–	–	–	–
Vergleichs-	CMP	A–M	C9	CD	DD	D9	C5	D5		C1	D1					X	X	X	–	–	–
	CPX	X–M	E0	EC			E4									X	X	X	–	–	–
	CPY	Y–M	C0	CC			C4									X	X	X	–	–	–
	BIT	A ∧ M		2C			24									7	X	–	–	–	6
Verzweigungs-	BCC	BRANCH ON C=0										90				–	–	–	–	–	–
	BCS	BRANCH ON C=1										B0				–	–	–	–	–	–
	BEQ	BRANCH ON Z=1										F0				–	–	–	–	–	–
	BNE	BRANCH ON Z=0										D0				–	–	–	–	–	–
	BMI	BRANCH ON N=1										30				–	–	–	–	–	–
	BPL	BRANCH ON N=0										10				–	–	–	–	–	–
	BVC	BRANCH ON V=0										50				–	–	–	–	–	–
	BVS	BRANCH ON V=1										70				–	–	–	–	–	–
	JMP			4C									6C			–	–	–	–	–	–
	JSR			20												–	–	–	–	–	–
Schiebe-	ASL	(Bild 5)		0E	1E		06	16						0A		X	X	X	–	–	–
	LSR	(Bild 5)		4E	5E		46	56						4A		0	X	X	–	–	–
	ROL	(Bild 5)		2E	3E		26	36						2A		X	X	X	–	–	–
	ROR	(Bild 5)		6E	7E		66	76						6A		X	X	X	–	–	–
Status-Register-	CLC	C=0													18	–	–	0	–	–	–
	CLD	D=0													D8	–	–	–	–	0	–
	CLI	I=0													58	–	–	–	0	–	–
	CLV	V=0													B8	–	–	–	–	–	0
	SEC	C=1													38	–	–	1	–	–	–
	SED	D=1													F8	–	–	–	–	1	–
	SEI	I=1													78	–	–	–	1	–	–
Versch.	NOP	NO OPER.													EA	–	–	–	–	–	–
	RTS	RETURN F. SUB													60	–	–	–	–	–	–
	RTI	RETURN F. INT													40						
	BRK	BREAK													00	–	–	–	1	–	–

der Inhalt einer Speicherzelle mit dem Inhalt des Akkumulators verglichen und das Ergebnis im Statusregister festgehalten (*Bild 4*). Beide Inhalte werden aber nicht verändert.

e) Verzweigungsbefehle
Z. B. BEQ (*Branch Equal Zero*). Eine Programmverzweigung wird ausgeführt, wenn das Zero-Bit des Statusregisters gesetzt (=1) ist. BCC (*Branch on Carry Clear*) Programmverzweigung, wenn Carry-Bit nicht gesetzt (=0) ist.

f) Schiebebefehle
Z. B. ASL (*Arithmetic Shift Left*). Der Inhalt einer Zelle wird eine Stelle nach links geschoben. Bit 7 wird ins Carry-Bit übertragen. Bit 0 wird Null gesetzt (*Bild 5*).

g) Statusregister-Befehle
Z. B. SED (*Sed Decimal Mode*). Das Bit D des Statusregisters wird gesetzt (=1). CLC (Clear Carry). Das Carry-Bit wird gelöscht (=0).

h) Verschiedene Befehle
Z. B. NOP (*No Operation*) Leerbefehl; RTS (*Return from Subroutine*) Rücksprung aus Unterprogramm.

1.3 Die Adressierungsarten

Beim System 6500 sind folgende Adressierungen [2] möglich:

a) Unmittelbare Adressierung (immediate)
Die dem Operationscode folgende Speicherzelle enthält die zu verarbeitenden Daten. LDX =FF*: Lade X Register unmittelbar mit FF.

b) Absolute Adressierung (extended)
Auf den Operationscode folgt die vollständige Adresse der gewünschten Speicherzelle. JMP F3 02: Springe unbedingt nach Zelle 02F3.

c) Indizierte Adressierung (indexed)
STA NAME, X.: NAME ist die Adresse einer Speicherzelle. Zu dieser Adresse wird der Inhalt des X-Registers addiert und so die effektive Adresse gebildet, in die der Akkumulator-Inhalt abgespeichert wird. Eine Indizierung mit dem Y-Register ist ebenfalls möglich.

* Für den symbolischen Code wird die für einen eigenen Assembler festgelegte Schreibweise verwendet.

d) Absolute Adressierung der Seite 0 (Zero Page)
Zur Adressierung der Speicherplätze 00...FF wird nur ein Byte benötigt. Sämtliche Befehle, die sich auf diesen Bereich beziehen, sind also nur 2 Byte lang. Häufig benutzte Daten wird man deshalb dorthin legen, um im Programm Speicherplätze einzusparen. Auch hier sind Indizierungen mit den Indexregistern möglich.

e) Indirekte Adressierung
Es ist durchaus möglich, daß in einem Programm ein Sprungziel nicht fest programmiert werden kann, da es im Programmablauf erst als arithmetischer Ausdruck berechnet wird, oder durch spezielle Eingabewerte geändert werden muß. Für solche Fälle kann man die indirekte Adressierung benutzen. Hier ist z. B. die bei einem Sprungbefehl angegebene Adresse nicht das Sprungziel, sondern die Adresse einer Speicherzelle, deren Inhalt das richtige Sprungziel darstellt. Bild 6 soll dies verdeutlichen. In Zelle 0250 steht der indirekte Sprung nach 0201. In dieser Zelle ist der erste (niederste) Adreßteil 00 in 0202 der zweite (höhere) Adreßteil 03 gespeichert, so daß das eigentliche Sprungziel die Zelle 0300 ist. Ändert man nun den Inhalt der Zellen 201, 202, so erhält man jeweils andere Sprungziele. Diese einfache Art der indirekten Adressierung ist beim System 6500 nur beim Sprungbefehl implementiert. Für einige Befehle gibt es zwei erweiterte indirekte Adressierungen, die indiziert indirekte und die indirekt indizierte Adressierung.

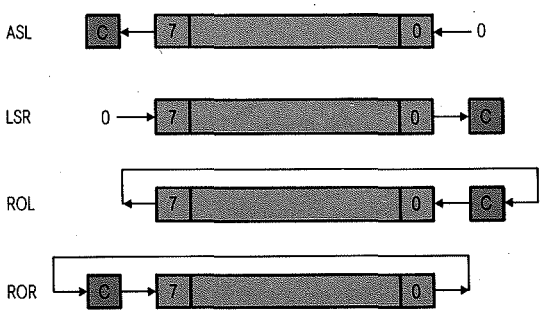
f) Indiziert indirekte Adressierung
Die indiziert indirekte Adressierung benutzt immer das Indexregister X, und die Basisadressen beziehen sich immer auf die Seite 0 (Zero Page). Mit *Bild 7* läßt sich diese Art der Adressierung am besten verdeutlichen. Im Programmablauf ist der Befehl LDA (00,X) gespeichert. Dies bedeutet: Lade den Akkumulator mit dem Inhalt der Zelle, deren Adresse in den Zellen 00 + X, 01 + X gespeichert ist. Ist X = 0, so wird der Inhalt der Zelle 0300, bei X = 2 der Inhalt von 0210 und bei X = 4 der Inhalt von 02FF geholt. Hier wird also die Basisadresse durch den Inhalt des X-Registers verändert.

g) Indirekt indizierte Adressierung
Wie bei f) beziehen sich die Basisadressen immer auf die Seite 0. Hier wird aber zur Indizierung das Indexregister Y benutzt. Der Inhalt dieses Registers wird nicht zur Basisadresse, sondern zu der dort hinterlegten Adresse hinzu-

	N	C	Z
(A) < (SP)	X	0	0
(A) = (SP)	0	1	1
(A) > (SP)	X	1	0

◀ Bild 4. Setzen der Statusregister-Bits N, C und Z durch Vergleichsbefehle: (A) = Akkumulatorinhalt, (SP) = Inhalt der angesprochenen Speicherzelle

▼ Bild 5. Durchführung der Schiebebefehle



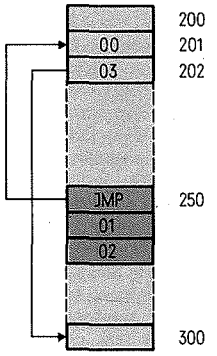


Bild 6. Indirekte Adressierung

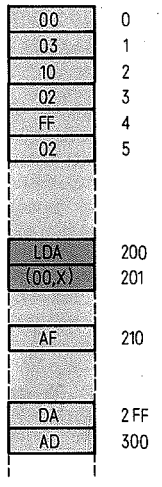


Bild 7. ▶ Indiziert indirekte Adressierung

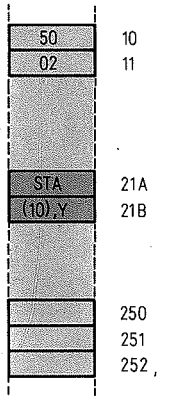


Bild 8. Indirekt indizierte Adressierung

addiert. Der Befehl STA (10), Y speichert also den Inhalt des Akkumulators in der Zelle 0250 ab, wenn Y = 0 ist. Mit Y = 1 in Zelle 0251 usw. (*Bild 8*).

h) Relative Adressierung

Beim System 6500 ist eine relative Adressierung nur bei den Verzweigungsbefehlen (*Branch*) implementiert. Hier erfolgt die Adressenmodifizierung relativ zum augenblicklichen Stand des Befehlszählers. Die Zellen 244 und 245 sollen den Befehl BNE +5 (Springe, wenn nicht Null, um 5 weiter) enthalten (*Bild 9*). Bei der Berechnung des effektiven Sprungzieles muß man vom augenblicklichen Stand des Befehlsfolgezählers ausgehen. Dieser zeigt bei der Entschlüsselung schon auf Zelle 246, das Sprungziel ist somit 246+5=24B. Dasselbe trifft auch bei Rückwärtsverzweigungen zu. Der in 24D und 24E gespeicherte Sprung BPL–8 (Springe, wenn positiv um 8 zurück) führt auf die Adresse 247 (24F–8=247).

i) Implizierte Adressierung (implied)

Bei einigen Befehlen ist keine Adressierung notwendig, da diese schon im eigentlichen Operationscode enthalten ist. Hierzu gehören Befehle wie TAX (Übertrage Akku in X-Register), INX, DEY usw.

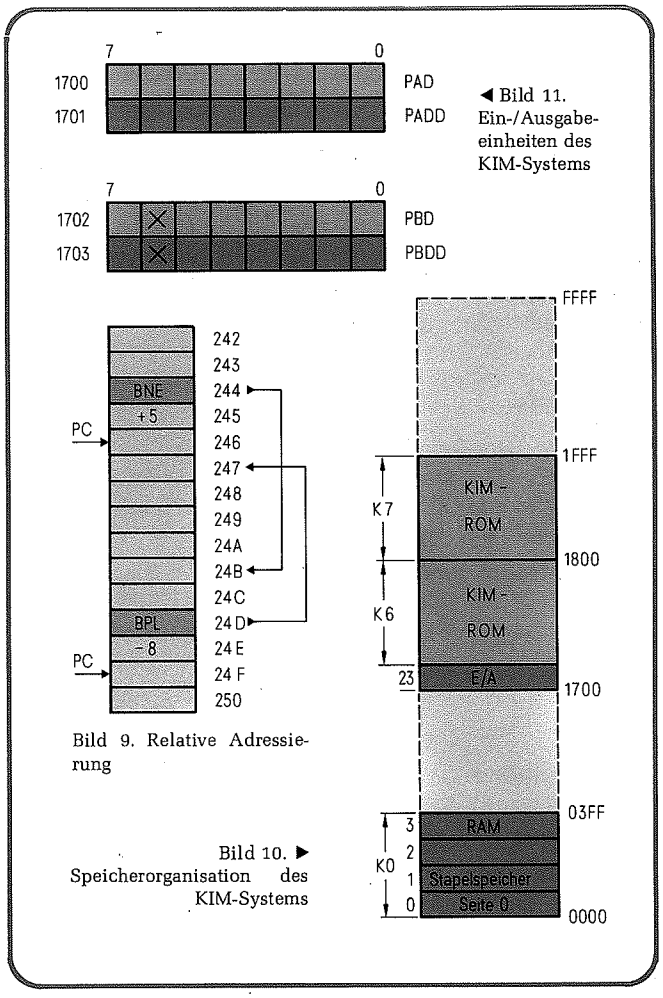
k) Akkumulator bezogene Adressierung (Accu referenced)

Befehle, die direkt den Akkumulator betreffen, benötigen ebenfalls keine weitere Adressierung. Dies sind beim System 6500 nur die Schiebebefehle. Eine Übersicht über die Arten der Adressierung der einzelnen Befehle liefert die Tabelle.

1.4 Befehlszykluszeiten

Beim Erlernen der Programmierung spielt die Ausführungszeit der Befehle noch keine Rolle. Dies wird anders beim Einsatz von Mikroprozessoren in der Regelungs- und Steuerungstechnik. Also dort, wo Echtzeitprobleme vorliegen. Hier wird man versuchen, durch geschickte Programmierung z. B. die Ausführungszeit von Programmschleifen, die häufig durchlaufen werden, möglichst klein zu halten. Dies ist ein Grund, sich mit der Programmierung in Assembler- oder Maschinensprache zu befassen, denn jedes in einer höheren Programmiersprache entwickelte und in den Maschinencode übersetzte Programm ist langsamer. Es würde hier zu weit führen, für jeden Befehl die Zykluszeit anzugeben. Allgemein kann man sagen, daß die kürzesten Zeiten (2 µs) bei allen Befehlen mit unmittelbarer Adressierung, die längsten Zeiten (7 µs) bei Befehlen mit indizierter Adressierung auftreten.

Dipl.-Ing. Ekkehard Flögel wurde in Bärn (Sudeten) geboren. Nach dem Studium der Nachrichtentechnik im Karlsruhe wurde er 1962 wissenschaftlicher Mitarbeiter an der Abteilung Schwingungstechnik des Institutes für Mechanik der Universität Karlsruhe. Sein Aufgabengebiet zerfällt in zwei Teile: erstens die Betreuung der gesamten Elektronik für die umfangreichen experimentellen Untersuchungen und Modellversuche im Labor des Institutes, zweitens die Bearbeitung eines Teilgebietes der Grundlagenforschung in der Schwingungstechnik. Beide Aufgabengebiete machten sehr früh den Einsatz von Computern notwendig, wobei in den letzten Jahren versucht wurde, Aufgaben der Labormeßtechnik durch den Einsatz von Mikroprozessoren zu vereinfachen.
Hobbys: Skifahren, Segeln
Privattelefon: (07 21) 81 75 62
ELEKTRONIK-Leser seit Anbeginn



◀ Bild 11. Ein-/Ausgabeeinheiten des KIM-Systems

Bild 9. Relative Adressierung

Bild 10. ▶ Speicherorganisation des KIM-Systems

1.5 Das KIM-System

Ein weiterer wichtiger Punkt ist die Peripherie eines Mikroprozessors, d. h. welche Möglichkeiten vorhanden sind, Programme und Daten einzugeben und fertige Programme extern zu speichern. Hier bietet die KIT-Version KIM 1 des Prozessor-Systems 6500 eine gute Lösung [3]. Programme und Daten können über ein Tastenfeld eingegeben und auf einer 7-Segment-Anzeigeeinheit kontrolliert werden. Ein Interface für einen 8-Kanal-Fernschreiber oder ein Datensichtgerät ist ebenfalls vorhanden. Fertige Programme lassen sich auf einer Tonbandkassette mit einem handelsüblichen Kassettenrecorder speichern. *Bild 10* zeigt die Speicherorganisation des KIM-Systems. Die Betriebsprogramme für die Tastenfeldeingabe und die Aufzeichnung auf Kassette sind in zwei 1-KByte-ROM-Bausteinen gespeichert. Für die Programmentwicklung steht noch 1 KByte RAM zur Verfügung. Ferner sind zwei Ein/Ausgabeeinheiten vorhanden. Sie verhalten sich wie adressierbare Zellen und sind bidirektional, d. h., eine Leitung kann als Eingang oder als Ausgang geschaltet sein. Diese Umschaltung übernimmt ein zu jeder Einheit gehörendes Datenrichtungs-Register, eine ebenfalls adressierbare Zelle (*Bild 11*). PAD (*Port A Data*) ist die E/A-Zelle, PADD (*Port A Data Direction*) das zugehörige Datenrichtungs-Register, ebenso erklären sich PBD und PBDD. Während bei Tor A alle 8 bit als Ein- oder Ausgang verwendet werden können, ist von Tor B Bit 6 anderweitig belegt, so daß hier nur 7 bit zur Verfügung stehen.

Fortsetzung folgt

2 Ablauf der Programmerstellung

Am Beispiel eines programmierten Rechteckgenerators soll der vollständige Arbeitsablauf der Programmerstellung für einen Mikroprozessor gezeigt werden. Dieser Ablauf wird von der Programmidee über das Flußdiagramm, die Umsetzung in den symbolischen Code bis zur Übersetzung in den Maschinen-Code verfolgt. Das ist der Weg, den man beschreiten muß, wenn keine Möglichkeit besteht, solche Programme auf größeren Rechnern zu simulieren und zu erstellen. Aber auch wenn dies zutrifft, ist man dennoch oft gezwungen, auf diese unterste Ebene der Programmierung herabzusteigen.

2.1 Programmierter Rechteckgenerator

Der Mikroprozessor soll dazu benutzt werden, an einem Ausgang ein in der Frequenz einstellbares Rechteck abzugeben. Als Ausgang wählen wir Bit 0 von Tor A. Tor A stellt im KIM-System ja eine Zelle PAD mit der Adresse 1700 dar. Um Bit 0 als Ausgang zu programmieren, muß im Datenrichtungs-Register (PADD) Bit 0 mit einer 1 markiert werden. Addiert man zum Inhalt der Zelle PAD eine 1, so wird Bit 0 jeweils 1 oder 0 werden. Um nun zwischen zwei Additionen eine Zeitverzögerung einzuführen, wird in dieser Zeit auf eine vorgegebene Zahl gezählt. Durch Ändern dieser Zahl wird die Periode der Rechteckschwingung eingestellt.

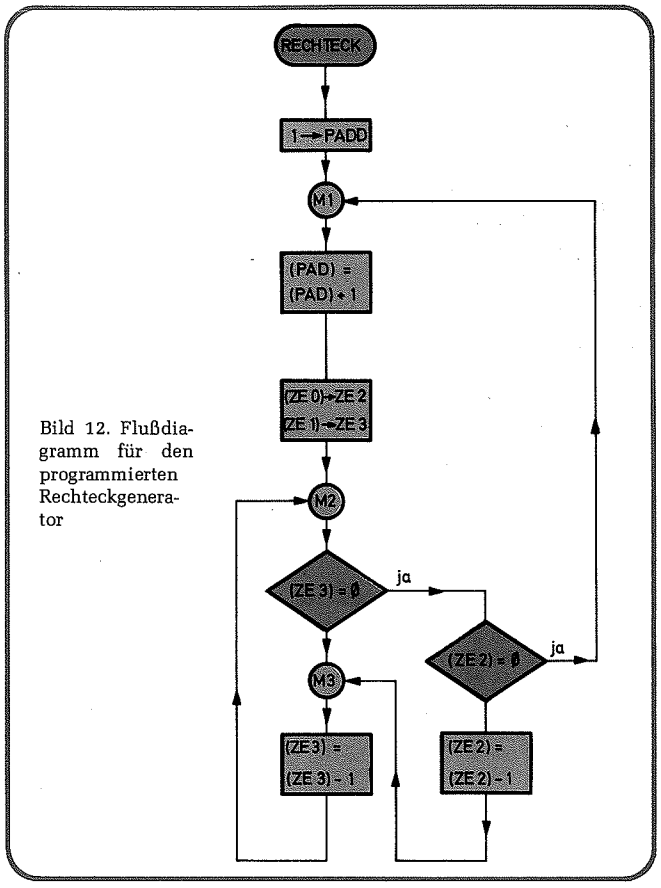


Bild 12. Flußdiagramm für den programmierten Rechteckgenerator

2.2 Flußdiagramm

Bild 12 zeigt das Flußdiagramm: Nach dem Start wird nach PADD eine Eins geschrieben und somit Bit 0 von Tor A als Ausgang markiert. Danach wird der Inhalt von PAD um eins erhöht. Je nach vorherigem Zustand ist Bit 0 von Tor A nun eins oder null. In den Zellen ZE0 und ZE1 ist die Zahl abgespeichert, die die Verzögerungszeit zwischen zwei Additionen von PAD bestimmt. Und zwar ist in ZE0 der höherwertige Teil (MSB) in ZE1 der niederwertige Teil (LSB) enthalten. In der nächsten Anweisung wird der Inhalt von ZE0 nach ZE2, der Inhalt von ZE1 nach ZE3 übernommen. Es folgt die erste Abfrage, ob der Inhalt von ZE3 gleich null ist. Falls dies nicht zutrifft, wird der Inhalt von ZE3 um eins erniedrigt. Ist der Inhalt von ZE3 gleich null, folgt die nächste Abfrage, ob der Inhalt von ZE2 null ist. Ist dies nicht der Fall, so wird der Inhalt von ZE2 um eins erniedrigt und anschließend auch der Inhalt von ZE3. Dieser war vorher 00 und ist jetzt FF. Das Programm springt nach M2 zurück und erniedrigt ZE3 um eins bis ZE3 wiederum null ist, danach wird ZE2 um eins erniedrigt, so lange, bis die Abfrage „Inhalt von ZE2 gleich null" erfüllt ist. Dann beginnt das Programm bei M1 aufs Neue. PAD wird um eins erhöht, (ZE0), (ZE1) nach ZE2 und ZE3 geschrieben, und es beginnt die zweite Halbperiode der Rechteckschwingung. Das Programm stellt somit eine unendliche Schleife dar, die nur durch einen äußeren Eingriff in das System unterbrochen werden kann.

2.3 Umsetzen des Flußdiagramms in Assembler-Anweisungen

Unter Benutzung der vorhandenen Befehle und Adressierungsmöglichkeiten wird nun mit Hilfe des Flußdiagramms ein Programm in Assembler-Sprache erstellt (*Bild 13*). Bei den beiden Verzweigungsbefehlen in Zeile 8 und 10 ist folgendes zu beachten. Ob eine Verzweigung ausgeführt wird oder nicht, hängt vom Zustand des Statusregisters ab. In unserem Fall also bei dem Befehl BNE, ob das Zero-Bit gesetzt oder nicht gesetzt ist. Aus der Tabelle ist aber ersichtlich, daß dieses Bit nicht nur durch einen direkten Vergleich, sondern auch durch andere Befehle (LDA, DEC usw.) gesetzt wird. Der Befehl STA beeinflußt aber das Statusregister nicht. Ob der Befehl BNE M3 in Zeile 8 ausgeführt wird, hängt entweder vom Befehl in Zeile 6 (LDA ZE1) oder vom Befehl in Zeile 12 (DEC ZE3) ab, je nachdem, ob dabei das Zero-Bit gesetzt wird oder nicht. Um bei der Abfrage in Zeile 10 (BEQ M1) das Statusregister zu setzen, wird vorher der Inhalt von ZE2 in den Akkumulator übernommen. Ein direkter Vergleich mit der Zahl Null muß also nicht durchgeführt werden.

2.4 Übersetzen des Assembler-Programms in die Maschinensprache

In vielen Fällen, d. h. immer dann, wenn größere Systeme zur Verfügung stehen, ist mit der Aufstellung des Assembler-Programms die Arbeit des Programmierens abgeschlossen. Die Übersetzung in den Maschinen-Code kann der Prozessor selbst übernehmen. Da wir hier aber von einem „Kleinst"-System ausgehen, das diese Vorteile nicht besitzt, soll diese Übersetzung mit der „Papier-und-Bleistift"-Methode erfolgen. Mit Hilfe der Tabelle wird Zeile für Zeile des Assembler-Programms in den Maschinen-Code

```
ZEILE          SYMB-CODE        KOMMENTAR

  1              LDA =1           1 ⟶ A
  2              STA PADD         (A) ⟶ PADD
  3          M1: INC PAD          (PAD)=(PAD)+1
  4              LDA ZEØ          (ZEØ)⟶ZE2
  5              STA ZE2
  6              LDA ZE1          (ZE1)⟶ZE3
  7              STA ZE3
  8          M2: BNE M3           Sprung n. M3, wenn(ZE3)≠ Ø
  9              LDA ZE2          (ZE2)⟶A
 10              BEQ M1           Sprung n. M1, wenn(ZE2)= Ø
 11              DEC ZE2          (ZE2)=(ZE2)-1
 12          M3: DEC ZE3          (ZE3)=(ZE3)-1
 13              JMP M2
```

Bild 13. Symbolischer Code des Reckteckgenerator-Programms

übersetzt. Dazu sind noch einige vorbereitende Überlegungen notwendig. Als erstes muß die Adresse für den Programmanfang festgelegt werden. Sie kann eigentlich im gesamten Speicherbereich liegen. Da aber Seite 0 und auch Seite 1 des RAMs besondere Funktionen erfüllen können, legen wir den Anfang des Programms z. B. auf die erste Adresse der Seite 2 (Zelle 200). Die Adressen der Zellen PADD und PAD sind durch das System festgelegt. Man muß also noch die Adressen der Hilfszellen ZE0...ZE3 angeben,

```
Adressen:

        ADRE     NAME
          ØØ     :=ZEØ
          Ø1     :=ZE1
          Ø2     :=ZE2
          Ø3     :=ZE3
        17ØØ     :=PAD
        17Ø1     :=PADD


Programm:

ZEILE  ADRE   M-CODE        SYMB-CODE          KOMMENTAR

    1  2ØØ   A9 Ø1           LDA #Ø1      Bit Ø von Tor A als Aus-
    2  2Ø2   8D Ø1 17        STA PADD     gang festlegen.
    3  2Ø5   EE ØØ 17  M1:   INC PAD      Incrementieren von PAD
    4  2Ø8   A5 ØØ           LDA ZEØ      Umspeichern des Inhaltes
    5  2ØA   85 Ø2           STA ZE2      der Zellen ZEØ und ZE1
    6  2ØC   A5 Ø1           LDA ZE1      in die Zellen ZE2 und ZE3.
    7  2ØE   85 Ø3           STA ZE3
    8  21Ø   DØ Ø6     M2:   BNE M3       Sprung n.M3,wenn(ZE3) ≠ Ø
    9  212   A5 Ø2           LDA ZE2
   10  214   FØ EF           BEQ M1       Sprung n.M1,wenn(ZE2) = Ø
   11  216   C6 Ø2           DEC ZE2      Inhalt der Zellen ZE2 und
   12  218.  C6 Ø3     M3:   DEC ZE3      ZE3 um 1 erniedrigen.
   13  21A   4C 1Ø Ø2        JMP M2
```

Bild 14. Befehlsfolge: Rechteckgenerator

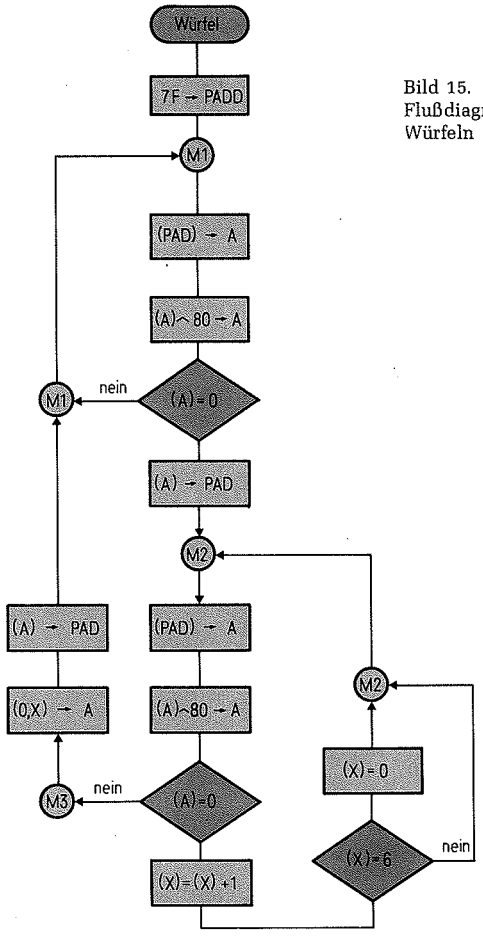


Bild 15. Flußdiagramm: Würfeln

dazu wählt man die ersten Zellen der Seite 0 (00...03). Die Übersetzung erfolgt dann nach folgendem Schema (*Bild 14*): Das Programm beginnt in Zelle 200. Dort steht A9, der Code für LDA (UNM), wobei die Zahl, die in den Akkumulator übernommen werden soll, sich in der Zelle 201 befindet. Der nächste Befehl ist ein Abspeichern des Akkumulator-Inhaltes in die Zelle PADD mit der absoluten Adresse 1701. Zelle 202 enthält also den Code 8D für STA(ABS), dann folgen das niederwertige Adreßbyte (01) in Zelle 203 und das höherwertige (17) in Zelle 204.

Besondere Sorgfalt muß bei der Berechnung der Sprungziele bei Verzweigungsbefehlen angewendet werden (siehe auch relative Adressierung). In Zelle 210 steht der Befehl BNE M3. Hierbei ist zu beachten, daß der Programmzähler bei der Entschlüsselung dieses Befehls schon die Zahl 212 enthält. Falls die Abfrage erfüllt ist, muß das Programm bei M3, d. h. bei Zelle 218, weitergeführt werden. Der Programmzähler muß um 6 erhöht werden. Der Code für BNE M3 lautet also D0 06. Auf die gleiche Art wird der in den Zellen 214 und 215 gespeicherte Rücksprung BEQ M1 berechnet. Der Programmzähler steht auf 216, das Sprungziel ist 205. Der Sprung führt um 11 Zellen zurück. Die negative Zahl –11 erhält man durch Bilden des 2er Complements der Zahl 11. Das 2er Complement wird dadurch gebildet, daß 0 und 1 gerade vertauscht werden, und zu diesem Ergebnis 1 addiert wird (11 = 0001 0001, komplementiert 1110 1110; 1110 1110 + 1 = 1110 1111 = EF). Somit ergibt sich für BEQ M1 der Code F0 EF. Mit den relativen Verzweigungsbefehlen kann man also um $+127_{10}$ (7F) nach höheren Adressen und um -128_{10} (80) zu niedrigeren Adressen springen.

3 Programmbeispiele

3.1 Würfeln

Durch dieses Programm wird das Spielen mit einem Würfel simuliert. Der Vorgang des Würfelns wird durch einen Tastendruck eingeleitet, beim Loslassen dieser Taste erscheint dann auf einer LED-Anzeigeeinheit die gewürfelte Augenzahl. Nach dem Start des Programms wird in einer Schleife geprüft, ob die Taste gedrückt ist oder nicht. Bei gedrückter Taste fängt das Programm an, die Zahlen 0...5 schnell zu durchlaufen, wobei nach jeder Zahl geprüft wird, ob die Taste noch gedrückt ist. Beim Loslassen wird die augenblickliche Zahl festgehalten und so umgewandelt, daß die entsprechenden Leuchtdioden des Würfelsymbols aufleuchten. Diese Codeumwandlung geschieht durch Aufsuchen des der Zahl zugeordneten Bitmusters in einer Tabelle. Das Flußdiagramm zeigt *Bild 15*. In der ersten Anweisung werden die Bits 0...6 des Tores A als Ausgang, Bit 7 als Eingang festgelegt. An diesen Eingang ist die Taste so angeschlossen, daß ihn die gedrückte Taste gegen Masse kurzschließt. In der folgenden Schleife wird Tor A eingelesen und durch Ausblenden festgestellt, ob Bit 7 gleich eins (Taste offen) oder null (Taste gedrückt) ist. Dieses Ausblenden geschieht durch Bilden der UND-Funktion (A) ^ 80. Damit werden Bit 0...6 des Akkumulators auf alle Fälle null, und nur Bit 7 behält seinen Wert. Ist die Taste nicht gedrückt, so springt das Programm zurück auf M1, andernfalls läuft es weiter zur nächsten Schleife. Dort wird auf die gleiche Weise geprüft, ob die Taste schon wieder losgelassen wurde. Wenn nicht, so wird der Inhalt des X-Registers um eins erhöht, und, wenn dabei nicht 6 herauskommt, nach M2 zurückgesprungen. Ist der Inhalt gleich 6, so wird das Register auf Null gesetzt. Diese Schleife wird so lange durchlaufen, bis die Taste losgelassen wird. Nun enthält das X-Register die gewürfelte Zahl. Dies ist aber zugleich auch die Adresse

derjenigen Zelle, aus der das Bitmuster geholt werden muß, das das entsprechende Würfelsymbol aufleuchten läßt.

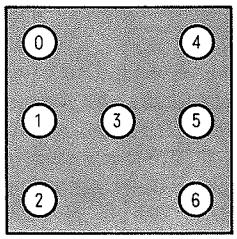
Zum Aufstellen dieser Code-Umsetz-Tabelle wird den sieben Leuchtdioden, die das Würfelsymbol bilden, jeweils ein Bit des Tores A zugeordnet (*Bild 16*). Die Tabelle und das Programm zeigt *Bild 17*.

3.2 Frequenzvariabler Rechteckgenerator

Das Programmbeispiel „Rechteckgenerator" soll nun so erweitert werden, daß durch Einwirkung von außen die Frequenz geändert werden kann. Hierfür sind zwei Drucktasten vorgesehen: die eine zum Erhöhen der Frequenz bis zu einer vorgegebenen oberen, die andere zur Verminderung auf eine untere Grenzfrequenz. Die Halbperiode der Rechteckschwingung ist durch die in zwei Zellen (ZE0, ZE1) gespeicherte Zahl gegeben. Eine Frequenzänderung wird also dadurch erreicht, daß diese Zahl erhöht (Frequenzverminderung) oder verkleinert (Frequenzerhöhung) wird. Bit 0 von Tor A ist wiederum der Ausgang des Rechteckgenerators, an Bit 1 und Bit 2 sind die beiden Tasten so angeschlossen, daß bei gedrückter Taste dieser Eingang gegen Masse kurzgeschlossen, ansonsten offen ist.

Das Flußdiagramm zeigt *Bild 18*. In der ersten Anweisung wird Bit 0 von Tor A als Ausgang festgelegt und in die Zellen ZE0 und ZE1 eine Zahl eingeschrieben, die der tiefsten Frequenz (im Beispiel etwa 100 Hz) entspricht. In der nächsten Anweisung werden die für den Verteiler notwendigen Adressen in die Zellen ZE2...ZE5 geschrieben. Es folgt der Programmteil Rechteck (*Bild 19*). Dieser entspricht im wesentlichen dem Programmbeispiel Rechteckgenerator, wobei aber nach Beendigung des Abzählvorgangs nicht auf den Anfang zurückgesprungen wird, sondern das Programm bei M2 weitergeführt wird. Im folgenden muß nun geprüft werden, ob eine Taste gedrückt worden ist, und welche. Hierzu wird Tor A in den Akkumulator eingelesen. Bit 1 und Bit 2 werden durch Bilden der UND-Funktion (A) ^ 06 ausgeblendet. War keine Taste betätigt, so ist der Akkumulator-Inhalt 6, und die beiden folgenden Abfragen werden übergangen, das Programm springt auf M1 zurück. Dies geschieht ebenfalls, wenn beide Tasten gleichzeitig gedrückt sind. Bei nur einer betätigten Taste enthält der Akkumulator entweder eine 2 oder eine 4. Diese beiden Zahlen werden nun zur Entscheidung herangezogen, ob die Frequenz erhöht oder erniedrigt werden soll. Dazu wird ein „programmierter indirekter Sprung" benutzt. Der Inhalt des Akkumulators wird in die Zelle S1 übertragen. Diese Zelle enthält aber den niederwertigen Teil einer Sprungadresse. Der höherwertige Teil dieser Adresse ist null, der Sprung führt also entweder auf die Zelle 02 oder die Zelle 04. Da dieser Sprung als indirekter Sprung programmiert ist, wird das eigentliche Sprungziel aus den Zellen 02 und 03, bzw. 04 und 05 geholt. Dort sind aber die Anfangsadressen der Programmteile INC, bzw. DEC gespeichert. Wird also Bit 1 von Tor A durch die Taste geerdet, so enthält der Akkumulator nach dem Ausblenden eine 4, und das Programm führt über den Programmteil DEC, bei geerdetem Bit 2 über den Programmteil INC.

Bild 16. ▶
Zuordnung der Bits 0...6 zum Würfelsymbol



▼ Bild 17. Befehlsfolge für das Würfelprogramm

Adressen:

ADRE	NAME
17ØØ	:=PAD
17Ø1	:=PADD

Tabelle:

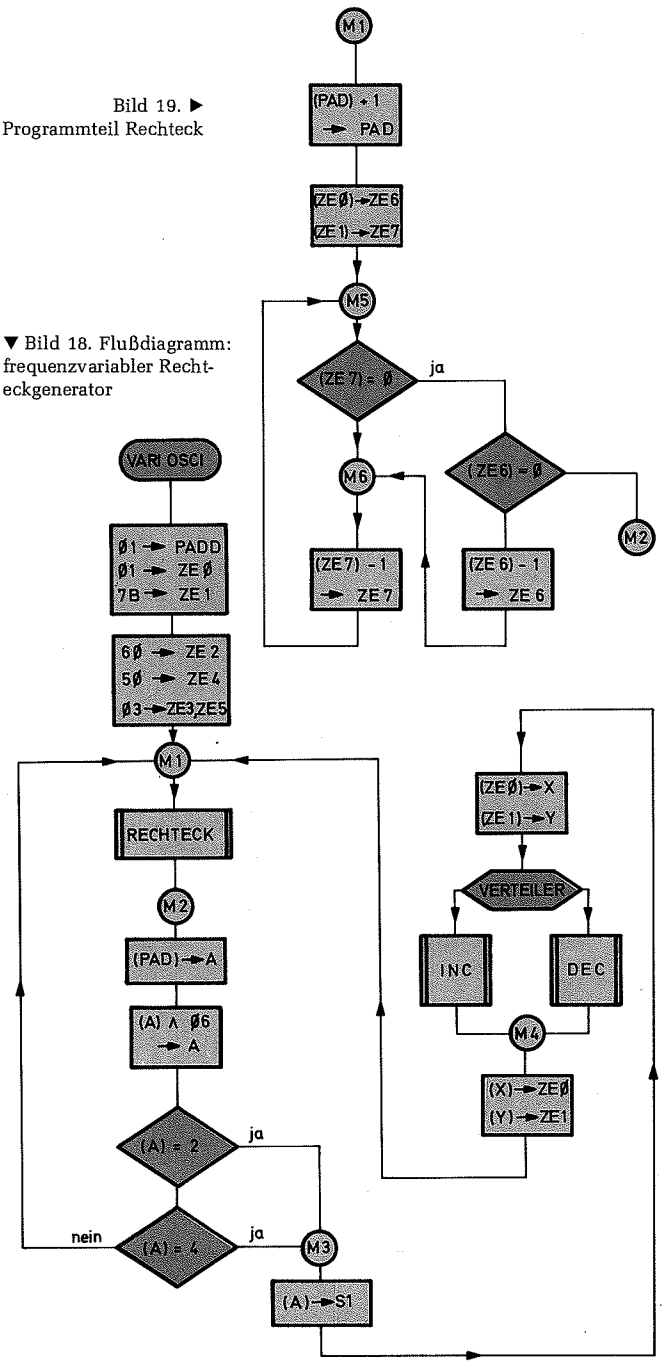
ADRE	M-CODE	KOMMENTAR
ØØØØ	Ø8	=1
ØØØ1	41	=2
ØØØ2	1C	=3
ØØØ3	55	=4
ØØØ4	5D	=5
ØØØ5	77	=6

Programm:

ZEILE	ADRE	M-CODE		SYMB-CODE	KOMMENTAR
1	2ØØ	A9 7F		LDA =7F	Bit Ø bis 6 von Tor A Aus-
2	2Ø2	8D Ø1 17		STA PADD	gang, Bit 7 Eingang
3	2Ø5	AD ØØ 17	M1:	LDA PAD	Einlesen von PAD
4	2Ø8	29 8Ø		AND =8Ø	Ausblenden von Bit 7
5	2ØA	DØ F9		BNE M1	Rücksprung wenn Taste nicht
6	2ØC	8D ØØ 17		STA PAD	gedrückt, sonst Löschen der
7	2ØF	AD ØØ 17	M2:	LDA PAD	Anzeige, und Prüfen ob Taste
8	212	29 8Ø		AND =8Ø	noch betätigt.
9	214	DØ Ø9		BNE M3	Wenn nicht, Sprung nach M3
1Ø	216	E8		INX	Inhalt des X-Reg. um Eins er-
11	217	EØ Ø6		CPX =Ø6	höhen, und wenn ≠ 6, Sprung
12	219	DØ F4		BNE M2	nach M2, sonst X-Reg. Null
13	21B	A2 ØØ		LDX =ØØ	setzen, und dann nach M2
14	21D	FØ FØ		BEQ M2	springen.
15	21F	B5 ØØ	M3:	LDA Ø,X	Bitmuster aus Tabelle holen
16	221	8D ØØ 17		STA PAD	und an Tor A ausgeben.
17	224	1Ø DF		BPL M1	Rücksprung nach M1

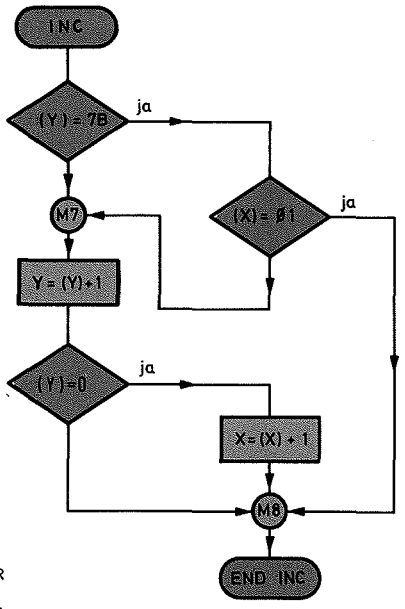
Bild 19. ▶
Programmteil Rechteck

▼ Bild 18. Flußdiagramm: frequenzvariabler Rechteckgenerator



Bevor diese Programmteile durchlaufen werden, wird der Inhalt der Zellen ZE0 und ZE1 in die Indexregister X und Y übernommen. Im Programmteil INC (*Bild 20*) wird die die Frequenz bestimmende Zahl um eins erhöht, wobei vorher verglichen wird, ob die unterste Frequenzgrenze schon erreicht ist. Durch diesen Programmteil wird die Frequenz vermindert. Ebenso wird in DEC (*Bild 21*) erst verglichen, ob die obere Frequenzgrenze (im Beispiel etwa 1 kHz) schon erreicht ist. Wenn nicht, wird die Zahl um eins erniedrigt und somit die Frequenz erhöht. Aus beiden Teilen springt das Programm nach M4, der Inhalt des X- und Y-Registers wird in die Zellen ZE0 und ZE1 zurückgespeichert, und die nächste Halbperiode beginnt wieder bei M1. Die Befehlsfolge des Programms zeigt *Bild 22*.

Bild 20. ▶
Programmteil INC



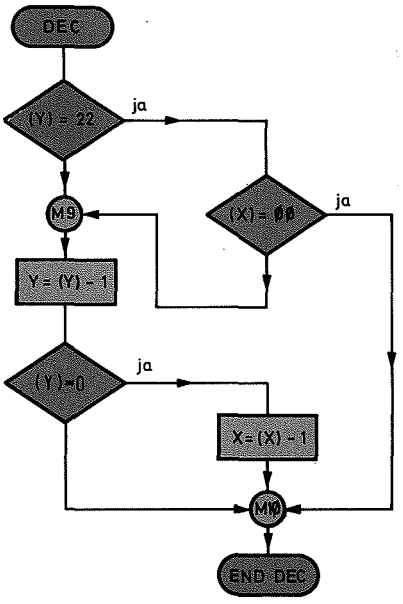
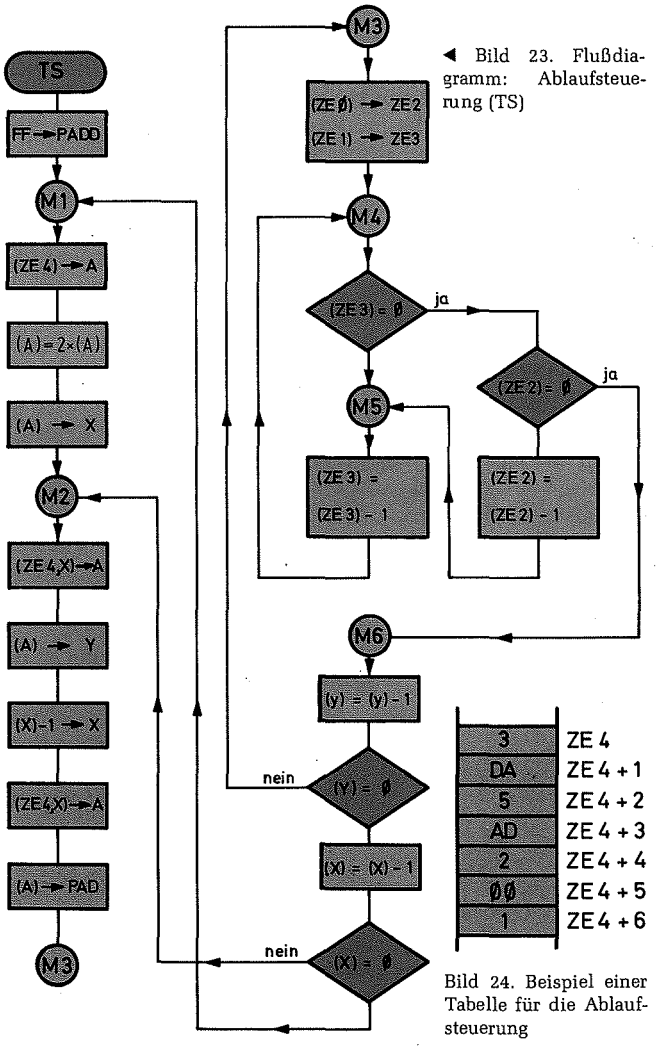


Bild 21. Programmteil DEC

▼ Bild 22. Befehlsfolge: frequenzvariabler Rechteckgenerator

```
Adressen:

      ADRE    NAME            KOMMENTAR

        ØØ    :=ZEØ     Verzögerungs-
        Ø1    :=ZE1     zeit
        Ø2    :=ZE2     Anfangsadresse Programm-
        Ø3    :=ZE3     teil  INC
        Ø4    :=ZE4     Anfangsadresse Programm-
        Ø5    :=ZE5     teil  DEC
        Ø6    :=ZE6     Hilfszelle
        Ø7    :=ZE7     Hilfszelle
      17ØØ    :=PAD
      17Ø1    :=PADD
        37Ø   :=INC
        36Ø   :=DEC
        348   :=S1

Programm:

ZEILE  ADRE  M-CODE        SYMB-CODE        KOMMENTAR

    1  3ØØ  A9 Ø1          LDA =Ø1     BitØ,Tor A als Ausgang
    2  3Ø2  8D Ø1 17       STA PADD    def.
    3  3Ø5  A9 Ø1          LDA =Ø1     Verzögerungszeit für
    4  3Ø7  85 ØØ          STA ZEØ     untere Frequenz fest-
    5  3Ø9  A9 7B          LDA =7B     legen
    6  3ØB  85 Ø1          STA ZE1
    7  3ØD  A9 Ø3          LDA =Ø3     Adressen für Verteiler
    8  3ØF  85 Ø3          STA ZE3     einschreiben
    9  311  85 Ø5          STA ZE5
   1Ø  313  A9 6Ø          LDA =6Ø
   11  315  85 Ø2          STA ZE2
   12  317  A9 7Ø          LDA =7Ø
   13  319  85 Ø4          STA ZE4
   14  31B  EE ØØ 17   M1: INC PAD     Anfang Rechteckgene-
   15  31E  A5 ØØ          LDA ZEØ     rator
   16  32Ø  85 Ø6          STA ZE6
   17  322  A5 Ø1          LDA ZE1
   18  324  85 Ø7          STA ZE7
   19  326  DØ Ø6      M5: BNE M6
   2Ø  328  A5 Ø6          LDA ZE6
   21  32A  FØ Ø7          BEQ M2
   22  32C  C6 Ø6          DEC ZE6
   23  32E  C6 Ø7      M6: DEC ZE7
   24  33Ø  18             CLC
   25  331  9Ø F3          BCC M5      Ende Rechteck
   26  333  AD ØØ 17   M2: LDA PAD     Prüfen,ob und welche
   27  336  29 Ø6          AND =Ø6     Taste gedrückt,durch
   28  338  C9 Ø2          CMP =Ø2     Ausblenden von Bit 1
   29  33A  FØ Ø4          BEQ M3      und 2 und Vergleich
   3Ø  33C  C9 Ø4          CMP =Ø4     mit 2 und 4.Nummer der
   31  33E  DØ DB          BNE M1      Taste als Adresse nach
   32  34Ø  8D 48 Ø3   M3: STA S1      S1 schreiben.
   33  343  A6 ØØ          LDX ZEØ     Verzögerungszeit nach X
   34  345  A4 Ø1          LDY ZE1     und Y Reg.umspeichern
   35  347  6C ØØ ØØ       JMP (VERT)  Indirekter Sprung ent-
   36  34A  86 ØØ      M4: STX ZEØ     weder nach INC oder DEC
   37  34C  84 Ø1          STY ZE1     Rückspeichern der geänder-
   38  34E  18             CLC         ten Zeit.Rücksprung auf
   39  34F  9Ø CA          BCC M1      Anfang Rechteckgen.

   40  36Ø  CØ 7B          CPY =7B     Programmteil INC
   41  362  DØ Ø4          BNE M7
   42  364  EØ Ø1          CPX =Ø1     Falls Verzögerungszeit für
   43  366  FØ Ø4          BEQ M8      untere Frequenzgrenze noch
   44  368  C8          M7: INY         nicht erreicht,(X)+(Y) um
   45  369  DO Ø1          BNE M8      1 erhöhen.
   46  36B  E8             INX
   47  36C  4C 4A Ø3   M8: JMP M4

   48  37Ø  CØ 22          CPY =22     Programmteil DEC
   49  372  DØ Ø4          BNE M9
   50  374  EØ ØØ          CPX =ØØ     Falls Verzögerungszeit
   51  376  FØ Ø4          BEQ M1Ø     obere Frequenzgrenze noch
   52  378  88          M9: DEY         nicht erreicht,(X)+(Y) um
   53  379  DØ Ø1          BNE M1Ø     1 erniedrigen.
   54  37B  CA             DEX
   55  37C  4C 4A Ø3  M1Ø: JMP M4
```



◀ Bild 23. Flußdiagramm: Ablaufsteuerung (TS)

3	ZE 4
DA	ZE 4 + 1
5	ZE 4 + 2
AD	ZE 4 + 3
2	ZE 4 + 4
ØØ	ZE 4 + 5
1	ZE 4 + 6

Bild 24. Beispiel einer Tabelle für die Ablaufsteuerung

```
Adressen:

        ADRE    NAME                        KOMMENTAR
          ØØ    :=ZEØ                       Taktzeit MSB
          Ø1    :=ZE1                       Taktzeit LSB
          Ø2    :=ZE2                       Hilfszelle
          Ø3    :=ZE3                       Hilfszelle
          Ø4    :=ZE4                       Zahl der Schaltzustände
        17ØØ    :=PAD
        17Ø1    :=PADD

ZEILE   ADRE    M-CODE      SYMB-CODE       KOMMENTAR

    1   ØØØØ    A9 FF           LDA =FF     Tor A als Ausgang fest-
    2   ØØ2     8D Ø1 17        STA PADD    legen.
    3   ØØ5     A5 Ø4       M1: LDA ZE4     Zahl der Schaltzustände
    4   ØØ7     ØA              ASL         mit 2 multiplizieren und
    5   ØØ8     AA              TAX         ins X-Register übernehmen.
    6   ØØ9     B5 Ø4       M2: LDA ZE4,X   Zeitdauer aus ZE4+X ins
    7   ØØB     A8              TAY         Y-Register schreiben.
    8   ØØC     CA              DEX         Bitmuster holen und an
    9   ØØD     B5 Ø4           LDA ZE4,X   Tor A ausgeben.
   1Ø   ØØF     8D ØØ 17        STA PAD
   11   212     A5 ØØ       M3: LDA ZEØ     Beginn Verzögerungs-
   12   214     85 Ø2           STA ZE2     zeit
   13   216     A5 Ø1           LDA ZE1
   14   218     85 Ø3           STA ZE3
   15   21A     DØ Ø6       M4: BNE M5
   16   21C     A5 Ø2           LDA ZE2
   17   21E     FØ Ø7           BEQ M6
   18   22Ø     C6 Ø2           DEC ZE2
   19   222     C6 Ø3       M5: DEC ZE3
   2Ø   224     18              CLC         unbedingter Rücksprung
   21   225     9Ø F3           BCC M4      nach M4.
   22   227     88          M6: DEY         Abzählen des Zeittaktes
   23   228     DØ E8           BNE M3
   24   22A     CA              DEX         Wenn (X) ≠ Ø, neuen Schalt-
   25   22B     DØ DC           BNE M2      zyklus beginnen, sonst Prog-
   26   22D     FØ D6           BEQ M1      ramm wiederholen.
```

Bild 25. Befehlsfolge: Ablaufsteuerung

3.3 *Ablaufsteuerung*

Mit diesem Programm soll eine einfache Steuerung, wie sie zum Beispiel bei Verkehrsampeln, Automaten und ähnlichen Geräten Verwendung findet, verwirklicht werden (vergleiche hierzu auch [1]). In vorgegebenen Zeitabständen sollen verschiedene Schalter (Relais, Thyristoren u. ä.) durch den Mikroprozessor betätigt werden. Eine Einwirkung von außen in diesen Funktionsablauf wird hierbei noch nicht berücksichtigt. Ist dieser Funktionsablauf einmal angestoßen, läuft er, ohne Änderung der vorgegebenen Reihenfolge, für immer ab.

Zur Realisierung dieses Programms wird jedem der 8 Bits des Tores A ein Schaltverstärker zugeordnet, der durch ein markiertes Bit (=1) eingeschaltet, durch ein nichtmarkiertes Bit (=0) ausgeschaltet wird. Durch Ausgabe eines entsprechenden Bitmusters kann eine beliebige Kombination dieser 8 Schalter betätigt werden. Außerdem soll die zeitliche Dauer eines solchen Schaltzustandes vorgegeben werden können. Dazu wird im Prozessor ein Zeittakt erzeugt. Die Dauer eines Schaltzustandes ist dann ein ganzzahliges Vielfaches dieses Zeittaktes. Beide Werte, das auszugebende Bitmuster und die zugehörige Zeitdauer, werden in einer Tabelle festgehalten, von wo sie vom Hauptprogramm abgerufen werden. Diese Art der Programmierung hat den Vorteil, daß bei einer Änderung der Ablaufsteuerung nur die Tabelle geändert werden muß, nicht aber das Programm.

Die Tabelle wird nach folgendem Schema aufgebaut. Sie beginnt mit der Zelle ZE4. Diese enthält die Anzahl der verschiedenen Schaltzustände der Steuerung. Ihre Adresse ist zugleich auch die Basisadresse. Jeder Schaltzustand belegt zwei aufeinanderfolgende Speicherzellen. Die erste enthält das Bitmuster, die zweite die Zeitdauer. *Bild 23* zeigt das Flußdiagramm des Programms. Der Programmablauf läßt sich am besten an einem Beispiel verfolgen. *Bild 24* stellt die Tabelle einer Ablaufsteuerung dar. Es werden 3 verschiedene Zustände ausgegeben (Inhalt von ZE4 ist 3). Die Bitmuster und Zeiten sind in den Zellen ZE4 + 1 bis ZE4 + 6 gespeichert. Nachdem im Programm die Bits 0...7 des Tores A als Ausgänge festgelegt wurden, wird der Inhalt von ZE4 in den Akkumulator übernommen und mit 2 multipliziert. Diese Multiplikation geschieht durch Linksschieben des Akkumulatorinhalts um eins (3 = 0000 0011, einmal nach links schieben ergibt 0000 0110 = 6). Diese Zahl wird in das X-Register übernommen. Mit der Adresse der Zelle ZE4 als Basisadresse wird nun durch indizierte Adressierung mit dem X-Register der Inhalt der Zelle ZE4 + 6 in das Y-Register geholt. Diese Zahl entspricht der Zeitdauer. Der Inhalt des X-Registers wird um eins erniedrigt und der Inhalt der Zelle ZE4 + 5 geholt. Dies ist das Bitmuster, das sogleich über Tor A ausgegeben wird. Danach durchläuft das Programm zwischen M3 und M6 eine Verzögerungsschleife, die die Dauer eines Zeittaktes darstellt. Beim Austritt aus dieser Schleife wird das Y-Register um eins erniedrigt. Falls der Inhalt des Y-Registers nicht null ist, wird diese Schleife ein zweites Mal durchlaufen, so lange bis Y gleich null ist. Jetzt wird der Inhalt des X-Registers um eins erniedrigt, und falls auch er nicht null ist, wird wiederum, beginnend bei M2, eine neue Zeit und ein neues Bitmuster aus der Tabelle geholt und ausgegeben. Ist die Tabelle abgearbeitet ((X) = 0), beginnt das Programm bei M1 aufs Neue. *Bild 25* zeigt die Befehlsfolge des Programms.

3.3.1 Verkehrsampel

Mit diesem Programm soll eine Verkehrsampelanlage an einer Straßenkreuzung simuliert werden. Es sind zwei Ampelpaare gegeben, die nach folgendem Schema geschaltet werden:

Phase	Ampel 1	Ampel 2	Zeit
1	rot	grün	10 s
2	rot+gelb	gelb	4 s
3	grün	rot	10 s
4	gelb	rot+gelb	4 s

Danach folgt wieder Phase 1.

Die Lampen der Ampeln werden wie folgt dem Tor A des Mikroprozessors zugeordnet.

Ampel 1	rot	Bit 0	Ampel 2	rot	Bit 4
	gelb	Bit 1		gelb	Bit 5
	grün	Bit 2		grün	Bit 6

Die Zahl der Schaltzyklen ist 4. Das erste auszugebende Bitmuster ist 0100 0001 = 41, das zweite 0010 0011 = 23, dann 0001 0100 = 14, und 0011 0010 = 32.

Damit erhält man folgende Tabelle:

ADR	CODE	KOMMENTAR
0000	FF	Maximale Zeitver-
0001	FF	zögerung ca. 0.5 s
0002		Hilfszelle
0003		Hilfszelle
0004	04	Schaltzyklen
0005	32	A1 gelb, A2 rot + gelb
0006	08	Zeit ca. 4 s
0007	14	A1 grün, A2 rot
0008	14	Zeit ca. 10 s
0009	23	A1 rot + gelb, A2 gelb
000A	08	Zeit ca. 4 s
000B	41	A1 rot, A2 grün
000C	14	Zeit ca. 10 s

Literatur

[1] *Feger, Gößler:* Einführung in die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 4, S. 79...84, H. 6, S. 64...70, H. 7, S. 53...58, H. 8, S. 67...74, H. 9, S. 71...78.

[2] *Tireford, H.:* Die Adressierungsarten bei Mikroprozessoren. ELEKTRONIK 1976, H. 12, S. 49...53.

[3] Mikrocomputer mit Tastatur und Anzeigeeinheit. ELEKTRONIK 1976, H. 11, S. 91...92.

Wer das Programmieren von Mikroprozessoren erlernen will, der muß die Adressierungsarten verstehen, und er muß mit ihnen umgehen können. Am Beispiel des Mikroprozessors M 6800 (Motorola), bei dem sechs verschiedene möglich sind, werden die Adressierungsarten erklärt. Die meisten anderen Typen werden aber nach denselben Prinzipien programmiert.

Dipl.-Ing. Hervè Tireford

Die Adressierungsarten bei Mikroprozessoren

Der Ablauf eines Mikroprozessor-Programms wird durch eine Folge von Befehlen gesteuert, die im Speicher abgelegt ist. Sie besteht aus einer Reihe von 8 bit langen Worten und wird Objekt-Programm genannt. Das Ziel (engl. *object*) beim Programmieren ist es, ein Objekt-Programm zu erzeugen, das den Mikroprozessor veranlaßt, die gewünschte Aufgabe auszuführen.

1 Informationsdarstellung

Die Grundeinheit der Information ist bei den 8-bit-Mikroprozessoren ein Byte, das durch ein binäres Muster (oder Wort) von 8 bit dargestellt wird. Durch ein Byte lassen sich verschiedenartige Informationen darstellen:

- eine Zahl zwischen Null und 255 (im Dualsystem)
- eine Zahl zwischen –128 und +127 (im Zweierkomplement)
- eine Zahl zwischen 00 und 99 (BCD-Code)
- ein binäres Muster zwischen 00 000 000 und 11 111 111
- eine Oktalzahl zwischen 000 und 377
- eine Hexadezimalzahl zwischen 00 und FF
- ein Buchstabe des Alphabets, die Zahlen 0...9 oder ein Satzzeichen (im ASCII-Code)
- die Hälfte einer Speicheradresse
- ein Befehl
- was immer auch der Programmierer darstellen möchte.

Was ein solches binäres Muster tatsächlich bedeuten soll, hängt vom Programmierer ab und vom Inhalt des Programms, das er schreibt.

1.1 Hexadezimale Schreibweise

Wegen der besseren Übersichtlichkeit faßt man bei der Angabe der Registerinhalte, der Adressen usw. je 4 bit zu einem Hexadezimalzeichen (oft auch kurz „Hex-Zeichen" genannt) zusammen (*Tabelle 1*). D. h., das 8-bit-Wort 0101 1010 kann auch als 5 A dargestellt werden, und das 16-bit-Muster 1101 0001 1101 1111 ist in hexadezimaler Schreibweise D1DF. Im folgenden wird ausschließlich diese Schreibweise verwendet.

Tabelle 1. 4-bit-Muster und die entsprechenden Hexadezimalzeichen

Hexadezimal-zeichen	4-bit-Muster		
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

2 Speicher, Peripherie und interne Register

Läßt man den hardwaremäßigen Aufbau außer Betracht, dann stellt sich dem Programmierer der Mikroprozessor (MPU) wie in *Bild 1* dar. Jeder Speicherplatz wird fortlaufend numeriert und kann 1 Byte aufnehmen. Die Nummer

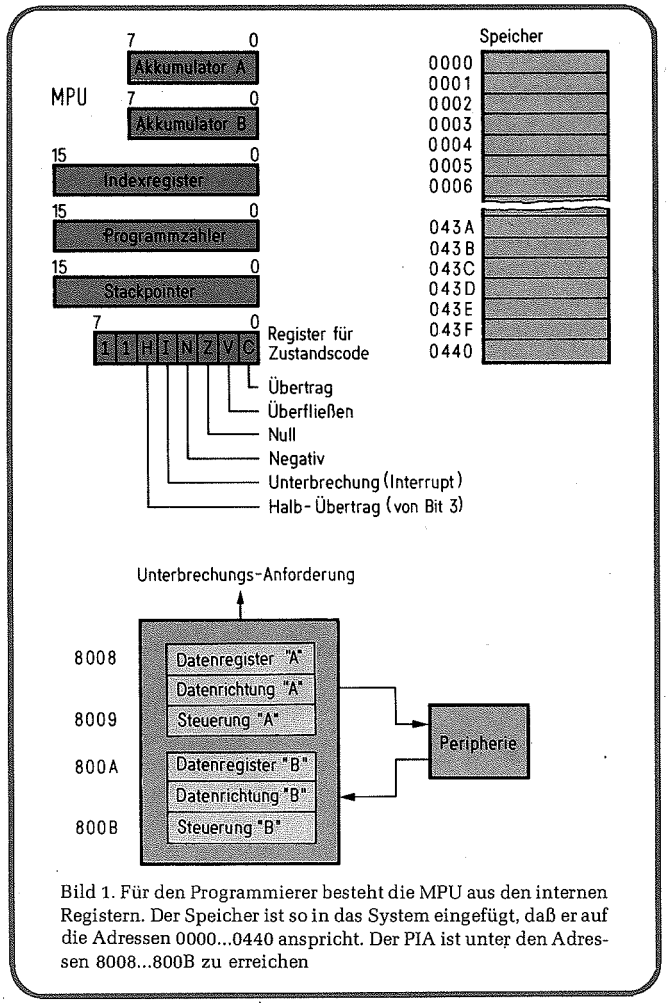


Bild 1. Für den Programmierer besteht die MPU aus den internen Registern. Der Speicher ist so in das System eingefügt, daß er auf die Adressen 0000...0440 anspricht. Der PIA ist unter den Adressen 8008...800B zu erreichen

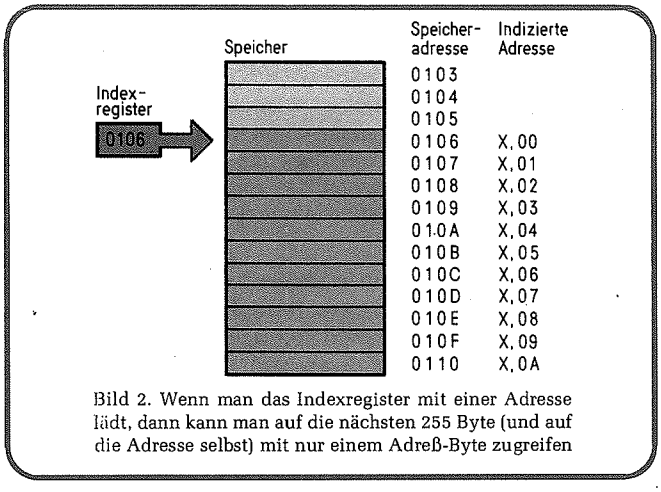


Bild 2. Wenn man das Indexregister mit einer Adresse lädt, dann kann man auf die nächsten 255 Byte (und auf die Adresse selbst) mit nur einem Adreß-Byte zugreifen

des Speicherplatzes bezeichnet man als Adresse, sie besteht aus zwei Byte und wird wieder mit Hexadezimalzeichen geschrieben.

Zwischen der MPU (Zentraleinheit) und peripheren Einheiten werden Daten über Bausteine wie den Peripherie-Interface-Adapter (PIA) übertragen. Dessen interne Register werden beim Einschreiben und Auslesen wie normale Speicherplätze behandelt. Eine wichtige Information für den Programmierer ist deshalb die Angabe der Adressen, die dem PIA zugeordnet sind. Unter der Annahme, daß das entsprechende Steuerwort im Steuerregister (des PIA) abgelegt wurde, werden Daten, die unter Adresse 8008 abgespeichert sind, automatisch zum Peripheriegerät (z. B. Fernschreiber) übertragen. Umgekehrt können Daten, die von der Peripherie kommen, von der Adresse 800A abgerufen werden.

Die MPU sieht der Programmierer als ein Gebilde mit sechs Registern.

2.1 Die Akkumulatoren

Die Akkumulatoren A und B sind 8-bit-Register, in die die Operanden und das Ergebnis arithmetischer Operationen eingeschrieben werden. Sie sind Brennpunkte der Datenbewegungen innerhab des Systems und viele Informationsübertragungen werden durch sie gesteuert. Zahlreiche Befehle führen Operationen durch, bei denen Daten aus beiden Akkumulatoren oder Daten aus einem Akkumulator und einer Speicherzelle „verarbeitet" werden.

2.2 Das Indexregister

Das Indexregister (IR) ist 16 bit lang und kann als Zeiger aufgefaßt werden, der sich auf einen beliebigen Speicherplatz (Adresse) innerhalb des Gesamtspeichers richten läßt. Er verbleibt dort, und man kann sich im weiteren Programmablauf wieder auf ihn beziehen.

Bild 2 zeigt eine weitere wichtige Verwendungsart des Indexregisters: Der Befehl „Lade das Indexregister" (LDX) wird dazu verwendet, die Hexadezimalzahl 0106 in das Register zu bringen. Danach zeigt das Indexregister auf Speicherplatz 0106. Ohne Indexregister kann der Zugriff zum Inhalt des Speicherplatzes 0106 nur über zwei Adreß-Bytes (01 und 06) erfolgen. Mit Indexregister kann man auf den Speicherplatz 0106 mit nur einem Adreß-Byte (hier 00) zugreifen (addiere 00 zum Inhalt des Indexregisters, um die gewünschte Adresse zu erhalten). In ähnlicher Weise kann der Speicherplatz 0107 mit dem Byte 01 angewählt werden (addiere 01 zu Inhalt des Indexregisters, um die gewünschte Adresse zu erhalten).

Da man mit einem Byte 256 verschiedene Werte darstellen kann, kann man durch diese *indizierte Adressierungsart* Speicherplätze anwählen, die im Bereich von X, 00 (d. h. die Speicherplatzadresse ist identisch mit dem Inhalt X des Indexregisters, weil X + 0 = X) bis X, FF (Speicherplatzadresse identisch mit dem Inhalt X des Indexregisters plus 255, weil X + FF = X + 255) liegen.

Der Inhalt des Indexregisters kann durch mehrere Befehle verändert werden: z. B. INX (addiere 1 zum Indexregister); DEX (subtrahiere 1 vom Indexregister); LDX (lade das Indexregister mit einem vorgegebenen Bitmuster). Nach kurzer Übung im Programmieren wird man den Wert des Indexregisters als Hilfe für die Datenmanipulation sehr zu schätzen wissen.

2.3 Der Programmzähler

Der Programmzähler (PC) ist ein 16-bit-Register, dessen Inhalt die Speicheradresse anzeigt, unter welcher der als nächster auszuführende Befehl abgespeichert ist. Vor der Ausführung des Befehls wird der Programmzähler automatisch mit einer neuen Adresse geladen, unter der sich dann der nächste Befehl befindet. Die Adressen sind Elemente einer linearen Folge, d. h. auf 0003 folgt 0004 usw.

Auch der Inhalt des Programmzählers kann durch eine Anzahl von Befehlen verändert werden, etwa „JUMP" (springe) oder „BRANCH" (verbinde). Diese Befehle lösen einen oder eine Folge von Befehlen aus, die nicht als nächstes Element der linearen Folge herausgegriffen und ausgeführt werden. „BRANCH"-Befehle können mit Bedingungen verknüpft sein, z. B. „Hole den Befehl vom Speicherplatz 0010, wenn der Inhalt des Akkumulators A gleich Null ist, ist das nicht der Fall, so führe den nächsten Befehl in der Folge aus".

2.4 Der Stapelzeiger

Der Stapelzeiger (*Stackpointer*, SP) ist ein 16-bit-Register, das eine Speicheradresse beinhaltet, und zwar aus einem Bereich, den man Stapelspeicher (*Stack*) nennt.

Der Stapelspeicher hat zwei Aufgaben:

- Der Inhalt beider Akkumulatoren kann mit einem 1-Byte-Befehl (anstelle der 2- oder 3-Byte-Befehle für „Store Accumulator") im Stapelspeicher abgespeichert werden
- Alle Register der MPU können im Stapelspeicher abgespeichert werden. Dieser Fall tritt ein, wenn der Prozessor sein gerade laufendes Programm zur Durchführung einer Aufgabe mit höherer Priorität unterbrechen muß. Ist das entsprechende Programm beendet, wird der Inhalt aller MPU-Register wieder aus dem Stapelspeicher zurückgeholt, und das ursprüngliche Programm kann weiterlaufen. Dieselbe Prozedur ergibt sich beim Abarbeiten von Unterprogrammen.

2.5 Zustandscode-Register

Das Zustandscode-Register (*Condition-Code-Register*, CCR) besteht aus sechs getrennten Flipflops, die auch als Fahnen oder *„Flags" bezeichnet werden. Sie signalisieren* bestimmte Zustände innerhalb der MPU. Ist z. B. das Ergebnis der von der MPU zuletzt ausgeführten Operation gleich Null, dann wird das „Null-Flipflop" (Z) gesetzt. Die Wirkung verschiedener Befehle hängt vom Zustand des CCR ab; damit kann man beispielsweise Programmverzweigungen realisieren.

Eine für den Programmierer gut geeignete Darstellung der MPU zeigt *Bild 3*, hier sind die drei 16-bit-Register als Zei-

ger aufgefaßt, die auf beliebige Speicheradressen gerichtet werden können.

3 Befehls- und Adressierungsarten

Viele Befehle beeinflussen irgendwelche Daten-Bytes. Ein Teil eines solchen Befehls muß daher das Daten-Byte zur MPU bringen oder festlegen, wo das Byte gefunden werden kann: z. B. bei

Lade Akkumulator A unmittelbar (*immediate*)
Lade Akkumulator A direkt (*direct*)
Lade Akkumulator A indiziert (*indexed*)
Lade Akkumulator A erweitert (*extended*).

Diese vier Befehle können in zwei Arten aufgeteilt werden:

- Die direkten, indizierten und erweiterten Befehle laden den Akkumulator A mit einem Byte, das aus einem Speicherplatz aufgerufen wird.
- Beim „unmittelbaren" Befehl ist das Byte, das im Akkumulator A abgespeichert werden soll, selbst ein Teil des Befehls.

Der Befehl „Lade Akkumulator A" (LDA A) wird in der MPU durch ein einzelnes Byte im Maschinencode dargestellt. Immer dann, wenn die MPU den Maschinencode für LDA A feststellt, holt sie sich ein Byte und bringt es in den Akkumulator A, entsprechend der im Befehl festgelegten Art. Da es vier verschiedene Möglichkeiten gibt, den Akkumulator A mit einem Byte zu laden, gibt es auch vier verschiedene Maschinencodes für LDA A (*Tabelle 2*).

Die vier Befehle werden folgendermaßen verwendet:

1. Immediate: Der Maschinencode 8602 veranlaßt, daß der Akkumulator A mit der Hexadezimalzahl 02 geladen wird; d. h. Immediate-Befehle benötigen zwei Bytes, wobei das zweite Byte die Daten selbst darstellt.
2. Direct: Der Maschinencode 9602 veranlaßt, daß der Inhalt des Speicherplatzes 0002 in den Akkumulator übernommen wird. Für Bild 3 hieße das, daß die Hexadezimalzahl 24 in den Akkumulator A gebracht wird. Diese „direkte" Methode des Speicherzugriffs erlaubt es, Daten in den ersten 256 Speicherplätzen sehr rasch zu adressieren. Befehle bei der „direkten" Adressierungsart benötigen zwei Bytes; der Adressenteil (02) besteht nur aus einem einzigen Byte, da der erste Teil der 2-Byte-Adresse gleich 00 ist.
3. Indexed: Steht im Indexregister z. B. die Adresse 013A (Bild 3), so veranlaßt der Befehl A602, daß der Inhalt (E1) des Speicherplatzes 013C (d. h. Indexregister plus 02) in den Akkumulator A geladen wird. Indizierte Befehle ermöglichen mit nur zwei Bytes den Zugriff zu einem Speicherbereich, dessen Adressen durch den Inhalt des Indexregisters festgelegt sind. Der Bereich erstreckt sich bis 255 Adressen „über" den Inhalt des Indexregisters. In diesem Fall handelt es sich um einen 1-Byte-Befehl plus ein Byte für den „Versatz" des Indexregisters.
4. Extended: Der Maschinencode B6800B veranlaßt die MPU, den Akkumulator A mit dem Inhalt des Speicherplatzes 800B zu laden. In Bild 3 ist dies z. B. das Steuerregister B. „Extended"-Befehle sind 3-Byte-Befehle. Der Adressenteil des Befehls umfaßt zwei Bytes, womit sich alle Speicherplätze im Adressenbereich von 0...65536 ansprechen lassen, d. h. die Adressen 0000...FFFF in hexadezimaler Schreibweise.

Es gibt aber noch zwei andere Adressierungsarten im Mikroprozessor M 6800: „Inherent" (zu deutsch etwa innewohnend, eigen, selbstanhaftend) und „Relative".

Tabelle 2. Die vier Arten des Befehls LDA A

Maschinencode	Befehl
86	Lade Akkumulator A unmittelbar (immediate)
96	Lade Akkumulator direkt (direct)
A6	Lade Akkumulator indiziert (indexed)
B6	Lade Akkumulator erweitert (extended)

5. Inherent: Einige Befehle brauchen kein zweites oder drittes Byte, um eine Adresse festzulegen oder Daten zu transportieren. Diese Befehle bestehen nur aus einem einzigen Byte, und die Adresse haftet dem Befehl selbst schon an.
   Beispiele:
   INX (addiere 1 zum Indexregister);
   DEC B (subtrahiere 1 vom Inhalt des Akkumulators B);
   CLRA (lösche den Inhalt von Akkumulator A).
6. Relative: Es gibt eine Gruppe von Befehlen, die Sprungbefehle, die alle die relative Adressierungsart verwenden. Es gibt unbedingte und bedingte Sprungbefehle.
   Beispiele:
   Unbedingt: BSR BALANCE (Springe zum Unterprogramm „BALANCE" und kehre zum nächstfolgenden Befehl zurück, wenn das Unterprogramm ausgeführt ist.
   Bedingt: BNE SUBTRACT (Gehe zum nächstfolgenden Befehl, wenn das Ergebnis des zuletzt ausgeführten Befehls gleich Null war. Falls es nicht gleich Null war, steht der als nächster auszuführende Befehl im Speicherplatz mit der Bezeichnung „SUBTRACT").

Ein Sprungbefehl beinhaltet ein Byte im Maschinencode, das die Art des Sprungs festlegt (BRA, BSR, BNE, BEQ usw.), gefolgt von einem Byte, das die relative Adresse des Bestimmungsortes vom Sprung angibt.

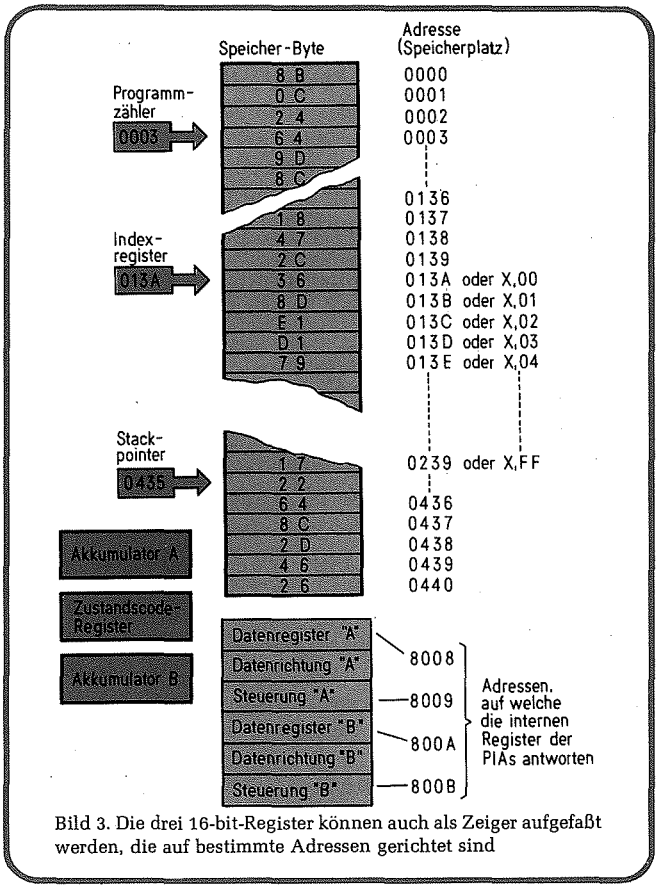


Bild 3. Die drei 16-bit-Register können auch als Zeiger aufgefaßt werden, die auf bestimmte Adressen gerichtet sind

Man stelle sich die folgende Befehlsfolge vor, zu einem Zeitpunkt, zu dem der Programmzähler 0113 enthält. Die Adresse des Bytes ist mit einem Stern gekennzeichnet.

Speicherplatz	*Speicherinhalt*
0110	
0111	20 (= BRA)
0112	03
0113 *	
0114	
0115	
0116	nächster Befehl

(Bestimmung nach BRA 03)

Als der Programmzähler den Wert 0111 enthielt, veranlaßte er die MPU, den Speicherinhalt unter dieser Adresse zu holen. Der Inhalt ist 20, der Code eines BRA-Befehls. PC erhöhte sich automatisch auf 0112, damit die MPU dort den Teil der relativen Adresse des 2-Byte-Befehls finden konnte, nämlich 03. Bevor der im Decodierer stehende Befehl, hier BRA 03, ausgeführt werden kann, wird der PC wiederum um eins erhöht, d. h. auf 0113 gebracht. Damit enthält er die Adresse, unter der der Mikroprozessor den nächsten Befehl zu finden glaubt. Die MPU hat jedoch noch nicht BRA 03 ausgeführt. Sie nimmt daher den laufenden Inhalt des PC (0113) und addiert dazu die relative Adresse (03), was 0116 ergibt. Diese Zahl wird in den PC geladen, wobei der vorherige Inhalt überschrieben, d. h. zerstört wird.

Der Befehl BRA 03 bewirkt daher, daß alle Befehle unter den Adressen 0113, 0114 und 0115 übersprungen werden. Das zweite Byte des Sprungbefehls ist die relative Adresse des Bestimmungsortes des Sprungs, die zum Inhalt des PC addiert wird und einen Wert zwischen +127 und –128 (dezimal) annehmen kann (da es als Zweier-Komplement-Zahl interpretiert wird). Das bedeutet, daß der Bestimmungsort eines Sprunges zwischen +127 und –128 Plätze von der Adresse des Sprungbefehls plus 2 entfernt sein kann. Man muß jeweils 2 dazuaddieren, da, wie auch das Beispiel zeigt, der PC zwei Schritte weiterläuft, nachdem er den Sprungbefehl selbst liest.

4 Programmiertechniken

Grundsätzlich kann man auf zwei verschiedene Arten Programme schreiben: Man erstellt selbst den Maschinencode, oder man programmiert in Assemblersprache.

Tabelle 3. Kleines Programm im mnemonischen und im Maschinencode

Speicheradresse	Maschinencode	Symb. Adresse (Label)	Mnemonischer Code	Bemerkung
2007	86		LDA A	Lade Akkumulator A mit dem Code für „*" im ASCII
2008	2A			
2009	C6		LDA B	Lade Akkumulator B mit der Zahl 03
200A	03			
200B	B7	OUTPUT	STA A	Bringe ein Sternchen (*) zum Ausgangsregister
200C	80			
200D	08			
200E	5A		DEC B	Subtrahiere 1 vom Akkumulator B
200F	26		BNE OUTPUT	Falls nicht gleich Null springe zurück zu „OUTPUT"

4.1 Erstellen des Maschinencodes

Jeder Befehl wird durch eine kurze Buchstabenfolge, den sogenannten mnemonischen Code, dargestellt. Einige wurden bereits erwähnt, z. B. LDA A, BSR, INX, DECB usw. Ist das Programm nicht allzu lang, dann kann man es Schritt für Schritt „per Hand" in den Maschinencode umwandeln, wie es in *Tabelle 3* dargestellt ist.

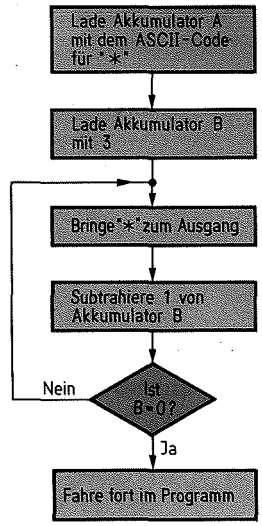


Bild 4. Flußdiagramm des Programms von Tabelle 3 und 4

Die erste Aufgabe besteht darin, das Problem zu definieren. Es soll ein Programm geschrieben werden, das bewirkt, daß der ASCII-Code für ein Sternchen (*) drei aufeinanderfolgende Male über das Ausgangsregister zur Peripherie gebracht wird (dabei wird vorausgesetzt, daß das periphere Gerät einen schnellen Puffer hat, der es ermöglicht, drei Sternchen zu speichern, während sie gedruckt werden). Das Flußdiagramm für das Programm ist in *Bild 4* dargestellt. Der Inhalt des Akkumulators B wird als Zähler verwendet, um festzustellen, wieviele Sternchen dem Ausgang zugeführt werden. Man sieht in der Tabelle, wie der Maschinencode in einzelne Bytes aufgeteilt ist und wie jeder Speicherplatz mit einem Byte belegt ist. Der erste Befehl, LDA A, ist ein unmittelbarer Befehl. Es folgt das Daten-Byte 2A (= ASCII-Code für „*"). Befehl Nummer zwei, ebenfalls unmittelbar, lädt den Akkumulator B mit 3. Der Befehl STA A (erweitert, benötigt 3 Byte) unter der Adresse 8008 kopiert den ASCII-Code 2A vom Akkumulator A in das Datenregister A. Ein „Inherent"-Befehl unter der Adresse 200E subtrahiert 1 vom Inhalt des Akkumulators B. Der Sprungbefehl bewirkt, daß der Programmzähler zum Befehl unter der symbolischen Adresse OUTPUT zurückspringt, falls das Ergebnis des Befehls DEC B nicht gleich Null war. Verwendet man die vorher schon aufgezeigte Methode, so ist die relative Adresse für den Sprungbefehl –6 (oder FA in hexadezimaler Schreibweise). Das Schreiben der Programme auf diese Art und deren Umwandlung in den Maschinencode bedingt oft zeitraubende Fehler. Wird ein Befehl ausgelassen, so ist es notwendig, ihn wieder einzusetzen und dann die Adressen aller darauffolgenden Befehle zu ändern. Zusätzlich wird es notwendig, alle relativen Adressen neu zu berechnen. Bei Programmen mit mehreren hundert Bytes ist das ein aussichtsloses Unterfangen.

4.2 Programmieren in Assemblersprache

Eine bessere Lösung ist es, das Programm im mnemonischen Code zu schreiben, ihn vom Computer selbst in den Maschinencode übersetzen zu lassen sowie alle relativen Adressen ebenfalls vom Computer berechnen zu lassen. Der Computer kann der Mikroprozessor selbst sein.

Die Folge der vom Programmierer geschriebenen mnemonischen Befehle wird Quellen-Programm (source program) genannt. Das Übersetzen des Quellen-Programms in ein Objekt-Programm (im Maschinencode) nennt man Assemblieren. Ein Computer, der assembliert, muß entsprechend programmiert werden. Das Programm, das die Übersetzungsarbeit ausführt, wird Assembler genannt. Fast alle Hersteller von Mikroprozessoren bieten ein Assembler-Programm für ihren speziellen Mikroprozessor an.

Tabelle 4. Programm von Tabelle 3 in Assemblersprache

```
          PIA EQU 8008
          LDA B # 03      Lade Zahl 03 in Akku B
          LDA A # '*'     Lade „*"-Code in Akku A
OUTPUT    STA A PIA       Bringe Zeichen heraus
          DEC B           Zähler
          BNE OUTPUT      Noch welche?
```

Benützt der Anwender den Mikroprozessor M 6800, so hat er die folgenden Möglichkeiten: Mit jeweils einem einfachen Erprobungskit oder dem Entwicklungssystem MES 6800, ausgerüstet mit minimal 8 KByte Speicherkapazität, kann ein Assembler „geladen" werden.

Ein anderes komfortables Entwicklungssystem, genannt EXORciser, hat ebenfalls Platz für den Assembler. Daneben aber enthält es andere Programme, die dem Entwickler helfen, seine eigenen Programme zu entwickeln und zu korrigieren. Weiterhin erlaubt das System die Entwicklung der Hardware für alle spezifischen Anwendungsfälle.

Schließlich gibt es noch sogenannte Cross-Assembler auf verschiedenen Timesharing-Systemen oder zum Gebrauch auf bereits bei Anwendern vorhandenen Computer-Anlagen. Im Gegensatz zum gerade erwähnten *Resident-Assembler* ist ein „Cross-Assembler" ein Übersetzungsprogramm, das auf einem Computer anderer Bauart läuft als auf dem, für den das Objekt-Programm geschrieben wurde.

Tabelle 4 zeigt das gleiche Programm wie Tabelle 3, nur ist es diesmal für automatisches Assemblieren geschrieben: Der Ausdruck PIA EQU 8008 sagt dem Assembler, daß der „PIA"sich auf den Speicherplatz 8008 bezieht und daß daher der Assembler jedesmal, wenn er auf die Buchstaben PIA trifft, die Adresse 8008 dafür einsetzen muß.

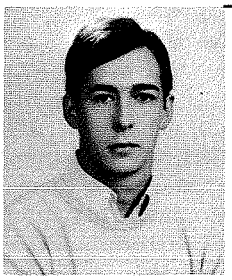
Das Zeichen „#" sagt dem Assembler, daß es sich um unmittelbare Befehle handelt. Der Apostroph vor dem Sternchen teilt dem Assembler mit, daß das Sternchen ein ASCII-Zeichen ist. Damit ersetzt der Assembler automatisch das Zeichen „*" durch den entsprechenden ASCII-Code. Das Wort ‚OUTPUT' ist eine Bezeichnung, die vom Assembler einem bestimmten Speicherplatz zugeordnet ist. Er bezieht sich jedesmal auf diese Adresse, wenn diese Bezeichnung aufgerufen ist. Eine solche symbolische Adresse (*Label*) ermöglicht es dem Programmierer, eine Liste von Befehlen festzulegen, die an einem Label beginnen, ohne daß er weiß, unter welcher Adresse die Befehle wirklich abgespeichert sind.

Beim Assembler besteht eine Programmaussage aus ein bis vier Punkten (oder Feldern):

SYMB. ADRESSE, BEFEHL, OPERAND, KOMMENTAR.

Das letzte Feld dient dazu, dem Programmierer zu helfen, das Programm zu verstehen. Es kann also irgendwelche Kommentare zur Programmgestaltung enthalten und wird vom Assembler nicht berücksichtigt.

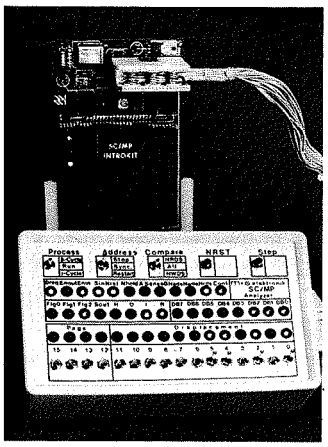
Alle Adressierungsarten mit Ausnahme der unmittelbaren werden vom Assembler selbst ausgewählt. Letztere wird durch das Zeichen „#" vor dem Operanden gekennzeichnet. Zusätzlich zum Befehlssatz des Mikroprozessors verfügt der Programmierer über eine Anzahl besonderer Befehle, sogenannte Assembler Direktiven. Diese Befehle geben dem Programmierer eine Kontrolle über den Prozeß des Assemblierens. Zusätzlich führen sie häufig benötigte Aufgaben automatisch durch, wobei sie dem Programmierer viel Wiederholungsarbeit ersparen.



Dipl.-Ing. Hervé Tireford ist Franzose und studierte Kybernetik an der ISIN in Nancy. 1972 begann er seine Tätigkeit am europäischen Hauptsitz von Motorola in Genf, wo er heute als Verantwortlicher im Bereich „Mikroprozessoren-Software" arbeitet.
Hobby: Fußball
ELEKTRONIK-Leser seit 1972

Testgerät für SC/MP-Mikroprozessor

Die Firma m + s elektronik stellt ein Prüfgerät vor, das es ermöglicht, den Zustand sämtlicher Anschlüsse des Mikroprozessors



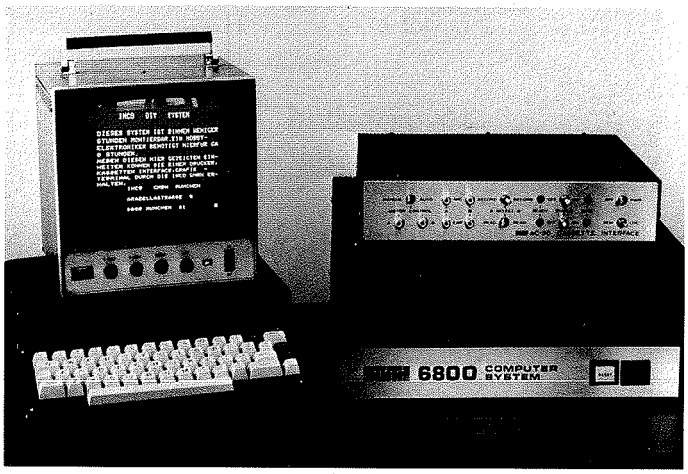
SC/MP (von National Semiconductor) laufend zu überwachen. Die Inhalte von Adressen- und Datenbus werden zwischengespeichert, so daß die im Zeitmultiplex vom Prozessor abgegebenen Informationen an den LEDs ständig sichtbar sind. Verschiedene mögliche Betriebsarten, wie z. B. Einzelschritt, Befehlsschritt, Stop auf Vergleichsadresse und Auto-Restart, bedeuten für den Benutzer gute Hilfen bei der Fehlersuche. Beim Erreichen der eingestellten Instruktions- oder Datenadresse gibt das Testgerät zusätzlich einen Triggerimpuls ab. Auch hier sind drei Möglichkeiten einstellbar: Triggerimpuls, wenn Adresse erreicht ist; Triggerimpuls, wenn aus Adresse gelesen wird und Triggerimpuls, wenn in Adresse geschrieben wird. Damit werden oszillografische Messungen sowohl am Prozessor selbst wie auch an Peripheriegeräten erleichtert. Der Anschluß des Prüfgerätes erfolgt über einen 40poligen IC-Clip; die CMOS-Eingänge des Testers halten das untersuchte System von zusätzlichen Belastungen frei.

□ Hersteller: m + s elektronik GmbH, Bergstr. 2, 8751 Leidersbach, Tel. (0 60 28) 3 67.

µC-Entwicklungssystem arbeitet in BASIC oder Assembler

Mit einfachsten Mitteln und innerhalb weniger Stunden läßt sich das Mikrocomputer-Entwicklungssystem DIY der Firma COI München installieren. Es besteht aus dem Mikrocomputer selbst, der auf der Basis des Mikroprozessors 6800 aufgebaut ist, einem Datensichtgerät, der Eingabe-Tastatur, einem Kassetten-Interface und einem Betriebssystem. Das Monitor-Programm ist in BASIC oder Assembler erhältlich und erlaubt auch dem weniger Erfahrenen das Erlernen des Umgangs mit dem Computer. Als weiteres Zubehör werden ein Drucker und ein Interface zum Anschluß handelsüblicher Fernsehapparate angeboten. Das Bildschirm-Terminal ist bereits für weniger als DM 1500.– erhältlich.

□ Vertrieb: C.O.I. München, Arabellastr. 5, 8000 München 81.



Mikroprozessor-Entwicklungssysteme sind im allgemeinen nicht billig. Viele potentielle Anwender wollen sich aber mit der neuen Technik vertraut machen, ohne gleich Zehntausende zu investieren. Trotzdem soll nicht auf jeglichen Komfort verzichtet werden. Diesen Kreis spricht das PDS-System der Firma Motorola an, das neben die Möglichkeiten, „zu Fuß" oder „motorisiert" zu programmieren, gewissermaßen noch das Fahrrad stellt.

Rudolf Hofer

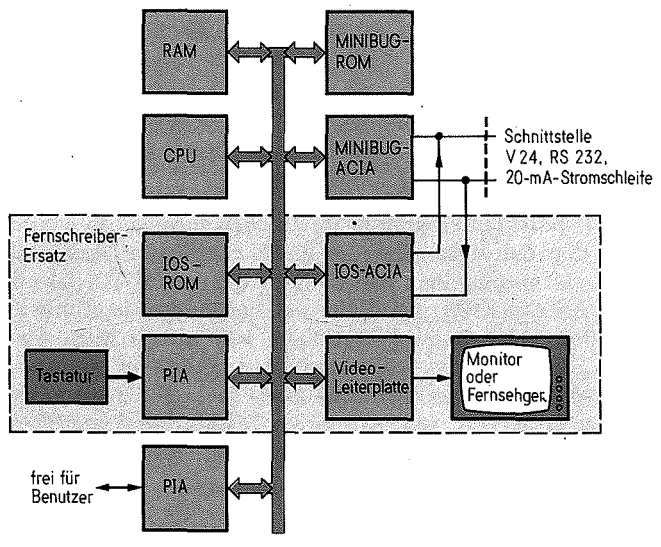
Programmbeispiele auf preiswertem Entwicklungssystem realisiert

1. Hardware-Aufbau

Das System besteht aus einer Mikrocomputer-Leiterplatte, einer Video-Leiterplatte, einer Tastatur und einem Monitor. Anstatt des Monitors kann auch ein handelsüblicher Fernsehempfänger als Datensichtstation verwendet werden (der Hf-Modulator ist bereits auf der Video-Leiterplatte vorhanden). Der Mikrocomputer enthält in der Grundausbaustufe 384 Byte RAM, von denen 128 Byte im wesentlichen für die veränderlichen Parameter des Betriebsprogrammes „MINIBUG II" benötigt werden, außerdem ist ein Sockel für einen 1-KByte-ROM vorhanden. Über die Schnittstellen „RS 232", „TTL" oder „20-mA-Stromschleife" kann ein Terminal (z. B. TTY) angeschlossen werden. Die Datenübertragungsgeschwindigkeit ist dabei zwischen 110 und 9600 Bd umschaltbar. Der ACIA-Baustein, über den dabei der Datenverkehr abgewickelt wird, kann aber auch intern mit einem weiteren ACIA verbunden werden, der von einem zweiten Hilfsprogramm, dem Input/Output Supervisor oder IOS-Programm (in einem ROM gespeichert), gesteuert wird. *Bild 1* zeigt die Blockschaltung dieser Anordnung. Die Tastatur, die Video-Leiterplatte und der Monitor ersetzen dabei den Fernschreiber. Das IOS-Programm sorgt dafür, daß über den IOS-ACIA normgerechte Fernschreiber-Signale im richtigen Format auf dem Bildschirm dargestellt werden. Dadurch kann das System auch für einen anderen Mikrocomputer als Terminal dienen (z. B. für den Exorciser). Die Verbindung zwischen den beiden ACIAs wird zu diesem Zweck mit Miniaturschaltern aufgetrennt, die sich auf einer der Leiterplatten befinden.

Als Ein/Ausgabe-Baustein steht dem Benutzer ein Peripherie-Interface-Adapter (PIA) zur Verfügung, der 16 programmierbare Ein/Ausgänge und Anforderungsleitungen für maskierte und nichtmaskierte Interrupts hat. Ein zweiter PIA handhabt die von der Tastatur ankommenden Daten und ist für den Anschluß eines Druckers vorgesehen. Das entsprechende Programm ist ebenfalls im IOS-ROM enthalten.

Eines der im letzten Abschnitt beschriebenen Beispiele zeigt, wie man einen preiswerten OEM-Drucker anstatt des vorgesehenen wesentlich teureren Druckers anschließt. Auch das von Motorola lieferbare Modem wurde nicht verwendet, da zur Abspeicherung der Programme auf Kassette eine Schaltung (*Bild 2*) genügt, die mit „Pfennigware" auf-



◄ Bild 1. Blockschaltung des PDS-Systems

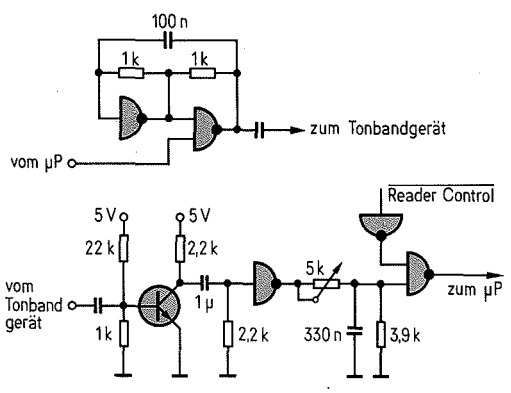


Bild 2. „Modem" zur Datenspeicherung auf Kassette

gebaut ist: Am Ausgang („Senden") des MINIBUG-ACIA wird der Modulator angeschlossen. Er unterscheidet sich von den normalerweise für diesen Zweck verwendeten Schaltungen dadurch, daß er nicht zwei Frequenzen, sondern nur eine erzeugt. Jedes H-Signal am Eingang öffnet ein Tor, das zugleich Bestandteil des Oszillators ist. Die Frequenz (für Kassettenrecorder am besten etwa 3,5 kHz) wird über den Mikrofoneingang des Recorders aufgenommen. Ein L-Signal am Eingang sperrt das Tor, und auf das Band wird die Frequenz Null aufgenommen. Die Demodulation wird durch dieses Verfahren sehr einfach: Im Prinzip genügt dafür ein Tiefpaß. Das Signal wird in der angegebenen Schaltung vom Kopfhörerausgang des Recorders abgenommen. Der Lautstärkeeinsteller muß deshalb immer in der gleichen Stellung stehen, die am besten mit Hilfe eines Oszilloskops ermittelt wird. Ebenso wird die Einstellung des 5-kΩ-Potentiometers anhand der Ausgangsimpulsform vorgenommen. Das Signal *Reader Control* zeigt an, ob der Mikrocomputer für die Ladefunktion bereit ist (L-Taste gedrückt?).

Es kommt vom MINIBUG-ACIA. wird aber vom Mikrocomputer erst nach dem es von einem Gatter invertiert wurde abgenommen. Das Ausgangssignal des Demodulators wird dem Mikrocomputer über den Empfangsanschluß des MINIBUG-ACIA zugeführt.

Die Schaltung wurde in Verbindung mit einem Billig-Kassettenrecorder (Preis etwa 80 DM) verwendet und arbeitet bei 110 Bd einwandfrei. Der Recorder verfügt über eine automatische Aussteuerung bei Aufnahme.

2 Betriebs-Software

2.1 IOS-Programm

Das IOS-Programm arbeitet auf der Basis des nichtmaskierten Interrupts. Wenn der Benutzer eine Taste drückt, gibt der PIA ein Interruptsignal an die CPU, die daraufhin – als Interrupt-Serviceroutine – das Datenregister des PIA liest und den Inhalt an den IOS-ACIA weitergibt. Dieser wandelt das parallele in ein serielles Signal um. Der Mikroprozessor kehrt dann vom Interruptprogramm zurück. Inzwischen hat der MINIBUG-ACIA das serielle Signal empfangen und wieder in ein paralleles umgewandelt. Falls der Mikroprozessor nicht gerade ein Benutzerprogramm bearbeitet, liest er das ACIA-Register und interpretiert den Inhalt, wie es das Monitor-Programm (MINIBUG) vorschreibt.

In der umgekehrten Richtung empfängt der IOS-ACIA die Daten, die von einem Benutzer- oder dem Monitor-Programm über den MINIBUG-ACIA ausgegeben werden und fordert bei der CPU einen nichtmaskierten Interrupt an. Die entsprechende Interrupt-Serviceroutine liest den ACIA-Inhalt und gibt ihn an den Bildschirm aus (falls vorhanden auch an den als Option erhältlichen Drucker). Noch einmal: Der Umweg über die beiden ACIAs wird gemacht, damit Programme, die den MINIBUG-ACIA benutzen genausogut mit einem Fernschreiber als Terminal laufen.

2.2 MINIBUG-Programm

Nach dem Drücken der gesondert verdrahteten Reset-Taste „befindet sich" der Mikroprozessor im MINIBUG-Programm, das im Prinzip genauso funktioniert wie das etwas ältere MIKBUG-Programm [1]. Es sorgt vor allem dafür, daß eine Programmeingabe vom Terminal überhaupt möglich ist. Dazu verfügt es über die Funktion „Speicherinhalt anzeigen und ändern", d. h., nach Eingabe eines M und einer Adresse (bestehend aus vier Sedezimalziffern) erscheint auf dem Bildschirm der Inhalt dieser Speicherzelle (zwei Sedezimalziffern). Soll er geändert werden, wird er einfach überschrieben. Auf diese Weise gibt man schrittweise das Programm im Sedezimalcode ein.

Soll ein Benutzerprogramm ausgeführt werden, gibt man ein G und die Startadresse ein. Ins Monitor-Programm kommt man wieder mit Reset. Für den Programmtest benutzt man den Befehl Software-Interrupt (SWI); wird er an die Stelle des Operationscodes eines Befehls gesetzt, dann wird das Programm an dieser Stelle angehalten und der Inhalt aller CPU-Register „ausgedruckt". Die Eingabe eines R bewirkt ebenfalls das „Ausdrucken" aller Registerinhalte.

Wichtig sind noch die Befehle „Punch" und „Load", da sie zum Abspeichern eines Programms auf Kassette und zum Laden des RAM von der Kassette benötigt werden. Das Drücken der Taste P bewirkt die Ausgabe eines bestimmten RAM-Bereiches über den MINIBUG-ACIA, ein mitlaufendes Tonbandgerät kann mit Hilfe des beschriebenen Modems die Daten (mit Prüfsumme) aufzeichnen. Die Grenzen des auszugebenden Bereiches werden vorher unter den Adressen A002...A005 abgelegt. Wird die Taste L gedrückt, dann geht zunächst das Signal *Reader Control* auf H-Pegel, und der ACIA ist empfangsbereit. Werden jetzt vom Tonbandgerät Daten im Format, wie es von der L-Funktion erzeugt wird, angeliefert, dann legt sie der Mikrocomputer im ursprünglichen Bereich ab. Entdeckt er einen Fehler in der Prüfsumme, dann gibt er ein Fragezeichen aus. Der Ladevorgang kann nun mit einem L fortgesetzt oder nach dem Zurückspulen des Bandes erneut begonnen werden.

Neuerdings ist zum PDS-System eine Assembler/Editor-Karte erhältlich, die mit der vorhandenen Software zusammenarbeitet.

2.3 Zugriff zum Bildschirm

Der Bildschirm kann mit 16 Zeilen zu je 32 Zeichen beschrieben werden. Der 512-Byte-RAM ist unter den Adressen C000...C1FF direkt erreichbar. Wird in eine dieser Speicherzellen z. B. mit Hilfe der M-Funktion eine zweistellige Sedezimalzahl geschrieben, dann wird diese als ASCII-Zeichen interpretiert und an der entsprechenden Stelle des Bildschirms dargestellt. Auf diese Weise sind auch Kleinbuchstaben und sogar griechische Zeichen darstellbar.

Eine zweite Methode, einen Text auf den Bildschirm zu bringen, benutzt Unterprogramme des MINIBUG-Programms. Mit dem folgenden kleinen Programm werden z. B. die von der Tastatur eingegebenen Zeichen fortlaufend auf dem Bildschirm dargestellt:

```
Schleife    JSR  INCH      Lese IOS-ACIA
            BRA  Schleife  Springe nach Schleife
```

Das Unterprogramm „INCH" bringt jeweils ein Zeichen an die Stelle, an der gerade der Cursor steht, es wird unter der Adresse E11F aufgerufen.

Soll vom Programm ein Zeichen ausgegeben werden, lädt man den auszugebenden ASCII-Code in den Akkumulator A und springt dann zum Unterprogramm OUTC, das unter der Adresse E108 zu erreichen ist. Besondere Möglichkeiten bieten zusätzliche Funktionen wie „Bildschirm löschen", „Zeile löschen", „Cursor in Ausgangsposition" usw. Dabei wird der entsprechende Code (bei „Bildschirm löschen" z. B. CRTL E $\triangleq 05_{16}$) von der Tastatur oder vom Programm wie ein anderes Zeichen einem der beiden Unterprogramme „übergeben".

Eine weitere Möglichkeit, Zeichen auf den Bildschirm zu bringen, hat man mit dem Unterprogramm PDATA 1 (An-

fangsadresse E130). Vor dem Unterprogrammsprung wird dabei das Indexregister mit der Anfangsadresse eines Speicherbereichs geladen. Ab dieser Zelle werden die folgenden Speicherinhalte als ASCII-Zeichen interpretiert und auf dem Bildschirm dargestellt, bis der Fernschreib-Code EOT (04) den Text abschließt. Auch hier können wieder die Zeichen eingefügt werden, die als Befehl für „Bildschirm löschen", „Zeile löschen" usw. entschlüsselt werden (*Bild 3*).

3 Programmbeispiele

Die folgenden Programme wurden auf dem Entwicklungssystem erstellt. Das Assemblieren wurde dabei per Hand durchgeführt, d. h., die Programme wurden direkt im Sedezimal-Code eingegeben. In den Flußdiagrammen und Kommentaren werden zum Teil symbolische Adressen verwendet (z. B. END1), die teilweise in ein- oder mehrfache Klammern gesetzt sind, dabei bezeichnet END1 die Speicherzelle mit der Adresse END1, der Inhalt dieser Speicherzelle wird mit (END1) bezeichnet, und ((END1)) bezeichnet ein Datum, das unter der in END1 gespeicherten Adresse zu erreichen ist.

3.1 *Verschieben eines Speicherbereichs*

Wenn man einzelne Programmteile getrennt entwickelt, kann es unter Umständen erforderlich werden, größere Bereiche im RAM an eine andere Stelle zu verschieben. Dafür eignet sich das Programm von *Tabelle 1*. Das zugehörige Flußdiagramm ist in *Bild 4* dargestellt. Die Anfangs- und Endadresse des zu verschiebenden Bereiches werden unter den Adressen ANF1/ANF1 + 1 bzw. END1/END1 + 1 vor dem Programmstart abgelegt. Die Endadresse des Bereichs, der „gefüllt" werden soll, wird in END2/END2 + 1 eingeschrieben. Da es sich jeweils um volle 2-Byte-Adressen handelt, kann das Programm für den gesamten adressierbaren Speicherbereich verwendet werden. Zu beachten ist lediglich, daß sich die Bereiche nicht überschneiden, wenn man von höheren zu niedrigeren Adressen verschiebt. Das Beispiel zeigt, wie man mit Hilfe der indizierten Adressierung ganze Bereiche bearbeiten kann. In diesem Fall wird das Indexregister zuerst mit der obersten Adresse des zu verschiebenden Bereichs geladen, sie weist wie ein Zeiger auf die Zelle, deren Inhalt anschließend in den Akkumulator geladen werden soll. In Zeile 6 wird das Indexregister dekrementiert, der Zeiger wird also um ein Byte nach unten verschoben. Der neue Inhalt wird schließlich in END1 und END1 + 1 abgelegt, da das Indexregister einen neuen Zeiger, nämlich

Tabelle 1. Programm für das Verschieben eines Speicherbereichs

Marke	Mnemonischer Code	Adress.-art	Sedezimal-code	Kommentar
STOPP	SWI		3F	
START	LDX END1	D	DE D5	(END1) → Indexregister
	LDAA 00	X	A6 00	((END1)) → AkkuA
	CPX ANF1	D	9C D3	(END1) < (ANF1)?
	BLT STOPP		2D F7	Sprung auf STOPP, wenn Bed. erfüllt
	DEX		09	Dekrementiere Indexreg.
	STX END1	D	DF D5	Indexreg. → END1
	LDX END2	D	DE D7	(END2) → Indexreg.
	STAA 00	X	A7 00	(Akku A) → END 2
	DEX		09	Dekrementiere Indexreg.
	STX END 2	D	DF D7	Indexreg. → END 2
	BRA START		20 EC	Unbedingter Sprung nach START

Symbolische Adressen: ANF1 ≙ 00D3/00D4 (höherwertiges/niederwertiges Byte), END1 ≙ 00D5/00D6, END2 ≙ 00D7/00D8; Adressierungsarten: I ≙ unmittelbar (immediate), D ≙ direkt, X ≙ indiziert (indexed), E ≙ erweitert (extended) – nur angegeben bei mehreren Möglichkeiten

END2 und END2 + 1, aufnehmen muß. Er wird ebenfalls um eins vermindert, – nachdem der Akkumulator-Inhalt an der Stelle abgespeichert wurde, auf die dieser Zeiger gerichtet war. Danach wird das Indexregister wieder in END2/END2 + 1 abgespeichert (Beachte: Das Indexregister hat 16 bit und bearbeitet jeweils zwei Byte: das im Adreßteil eines Befehls angegebene [= höherwertig] und das nächsthöhere [= niederwertig]). Die Zellen END1/END1 + 1 und END2/END2 + 1 beinhalten also immer den jeweils aktuellen Zeigerstand für die beiden Bereiche, die bearbeitet werden. Die Zelle ANF1 beinhaltet einen konstanten Wert und wird vom Programm nicht verändert. Man könnte sie einsparen, wenn man anstelle des direkten Vergleichsbefehls in Zeile 4 den unmittelbaren Vergleichsbefehl verwendete. Der Befehlscode hieße dann 8C statt 9C, und der Adreßteil D3, der jetzt die Stelle, unter der der Operand zu finden ist, angibt, würde durch den Operanden selbst (Anfangsadresse des zu verschiebenden Bereichs) ersetzt werden. Bei einer solchen Änderung verändern sich natürlich auch die relativen Rücksprungadressen der Befehle BLT und BRA (weil der Adreßteil nun 2 Byte umfaßt). Bei relativen Sprüngen geht man beim Mikroprozessor 6800 von dem Byte aus, das dem gesamten Sprungbefehl folgt (für den BLT-Befehl in

```
LDX ANF     Lade Indexreg. mit Anfangsadresse des Textbereichs
JSR PDATA1  Springe zum Unterprogramm PDATA1

ANF: 05     Code für CRTL E ≙ lösche Bildschirm
     41     ASCII-Code für A
     42                    B
     43                    C
     44                    D
     45                    E
     04     Code für EOT = Ende des Textes
```

Bild 3. Das Unterprogramm PDATA1 bringt die Daten ab der im Indexregister gespeicherten Adresse auf den Bildschirm, bis der Text durch EOT abgeschlossen wird

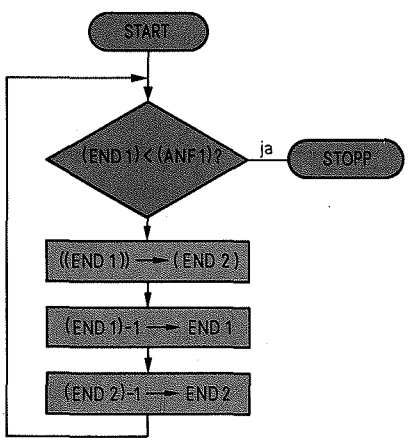
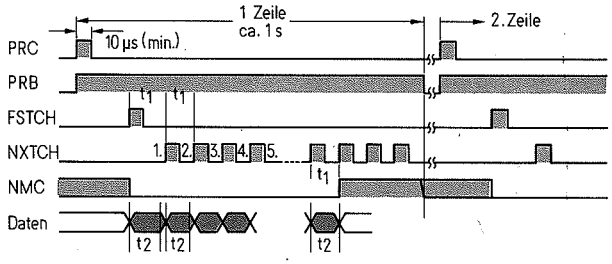


Bild 4. ▶
Flußdiagramm für das Verschieben eines Speicherbereichs

```
  LDS END2
  LDX END1
S:LDAA 00,X
  DEX
  PSHA
  CPX ANF1,I
  BNE S
  SWI
```

◀ Bild 5.
Elegantere Methode, einen Speicherbereich zu verschieben

Bild 7. ▶
Impulsdiagramm der Druckersignale



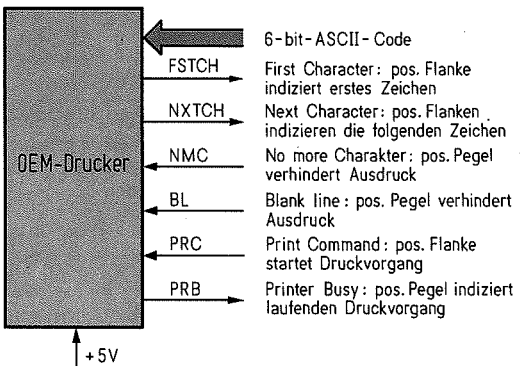


Bild 6. Ein- und Ausgangssignale des OEM-Druckers

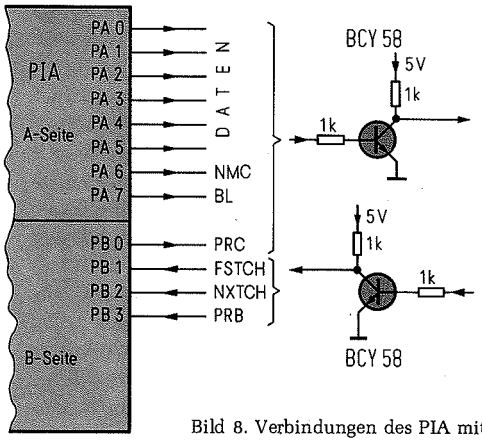
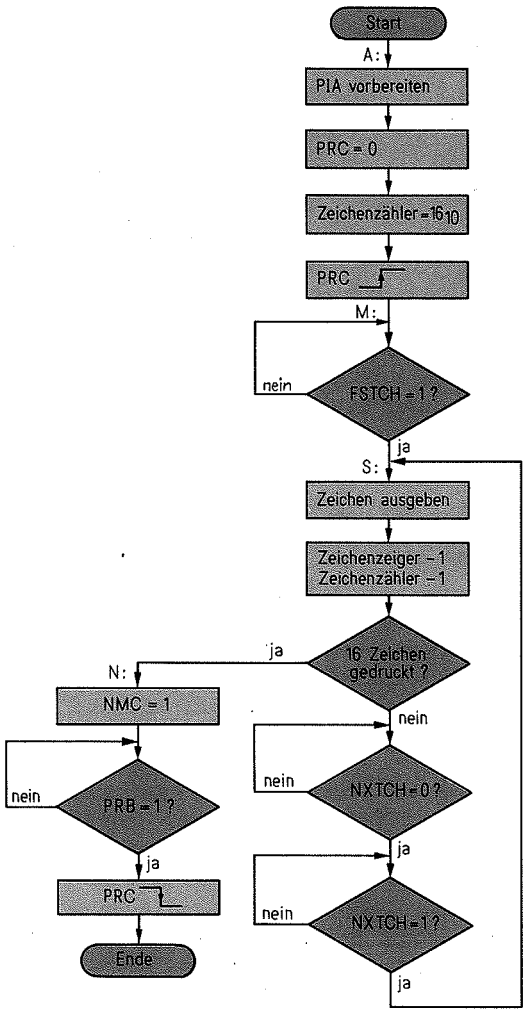


Bild 8. Verbindungen des PIA mit dem Drucker




```
A:  CLR   8005 }        DRR ansprechbar
    CLR   8007 }
    LDAA  FF,I }        PIA A alles Ausgänge
    STAA  8004 }
    LDAA  01,I }        PIA B0 = Ausg., PIA B1...7 = Eingang
    STAA  8006 }
    LDAA  04,I }        DR ansprechbar
    STAA  8005 }
    STAA  8007 }
    LDAA  01,I }        PRC = "0"
    STAA  8006 }
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP                 Verzögerung
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    NOP
    LDAB  10,I          Zeichenzähler = 16₁₀
    LDAA  FE,I }        PRC
    STAA  8006 }
M:  LDAA  8006,E }      DR B  Akku
    COMA                Akku invertieren
    ANDA  02,I }        Maske für PIA B1 (FSTCH)
    BEQ   M             FSTCH = "1"?
S:  LDAA  00,X          Hole Zeichen
    ANDA  3F,I          Maske für nied. 6 bit
    COMA
    STAA  8004          Zeichen ausgeben
    DEX                 Zeichenzeiger -1
    DECB                Zeichenzähler -1
    BLT   N             16 Zeichen gedruckt?
NXTCH 0: LDAA 8006,E }
    COMA                NXTCH = "0"?
    ANDA  04,I
    BNE   NXTCH 0 }
NXTCH 1: LDAA 8006,E }
    COMA                NXTCH = "1"?
    ANDA  04,I
    BEQ   NXTCH 1 }
    BRA   S
N:  LDAA  BF,I }        NMC = "1"
    STAA  8004 }
K:  LDAA  8006,E }
    COMA                PRB = "1"?
    ANDA  08,I
    BNE   K }
    LDAA  01,I }        PRC
    STAA  8006 }
    RTS                 Unterprogramm-Rücksprung
```

Bild 9. Flußdiagramm und Befehlsfolge für das Drucken einer Zeile (die Adressierungsarten wurden gemäß der Tabelle angegeben)

Zeile 5 wäre das 09 in Zeile 6), weil der Programmzähler schon beim Bearbeiten des Befehls auf den Befehlsteil des nächsten hochgezählt wird. Von hier aus zählt man einfach vorwärts oder rückwärts bis zum Befehlsteil des anzuspringenden Befehls. Beim Rücksprung muß natürlich die Komplementschreibweise benutzt werden, also FF = –1, FE = –2 usw. Ein Vorwärtssprung vom BLT-Befehl in Zeile 5, zum Dekrementier-Befehl in Zeile 10 würde die Sprungweite 07 erfordern.

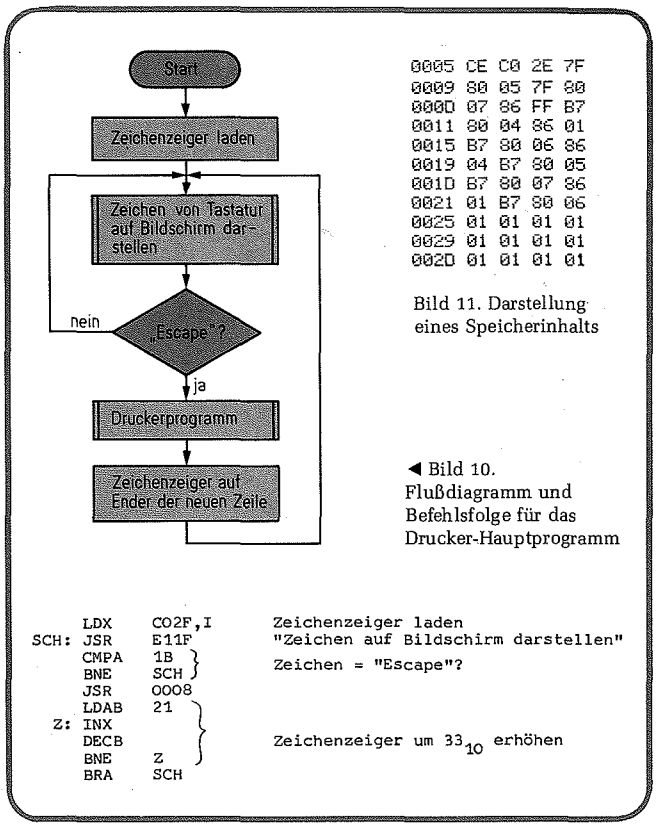
Das Problem, einen Speicherbereich zu verschieben, kann aber noch wesentlich einfacher gelöst werden (*Bild 5*). An diesem Beispiel wird die vorteilhafte Verwendung des Stapelspeichers demonstriert. Am Programmbeginn wird der Stapelzeiger mit einer Adresse geladen, die den Anfang des Stapelspeichers festlegt. Der Inhalt des Indexregisters deutet auf das gerade zu verschiebende Byte, er wird in einer Schleife dekrementiert und mit der unteren Bereichsgrenze verglichen. Durch den Befehl PSHA wird der Inhalt des Akkumulators in den Stapelspeicher übertragen und der Stapelzeiger dekrementiert.

3.2 Ansteuerung eines OEM-Druckers

Zum Registrieren von Meßwerten oder für ähnliche Aufgaben eignen sich billige OEM-Drucker (Preis unter 500 DM) wie der Typ 5010, der von der Firma Kontron vertrieben wird. Er „bedruckt" aluminiumbeschichtetes Papier mit 16 Zeichen pro Zeile und kann 64 Zeichen durch eine 5 x 7-Punkte-Matrix gemäß dem 6-bit-ASCII-Code darstellen. Zur Stromversorgung genügt eine Spannung von 5 V, die man über eine „dicke" Diode von der Mikrocomputer-Versorgung ableiten kann. *Bild 6* zeigt, welche Signale der Drucker benötigt und abgibt, in *Bild 7* ist das zugehörige Impulsdiagramm dargestellt. Diese Signale sollen rein softwaremäßig erzeugt bzw. erkannt werden. Zwischen Mikroprozessor und Drucker liegt nur ein PIA – in diesem Fall der Benutzer-PIA des PDS-Systems – mit einer Transistor-Anpassungsstufe pro Ein/Ausgang (*Bild 8*).

Das Programm wird so ausgelegt, daß zuerst ab der zweiten Bildschirmzeile ein Text geschrieben werden kann, der ausgedruckt wird, wenn die Taste „Escape" gedrückt wird. Bei jedem Tastendruck wird eine halbe Bildschirmzeile ausgegeben, da der Drucker nur 16 Zeichen in einer Zeile darstellen kann. *Bild 9* zeigt das Flußdiagramm und die Befehlsfolge für die Ausgabe einer Zeile. In *Bild 10* ist der Ablauf des Hauptprogramms dargestellt: Es verwendet das MINIBUG-Unterprogramm, das die von der Tastatur her eingegebenen Zeichen auf dem Bildschirm darstellt. Der Inhalt des Indexregisters wird im Druckerprogramm als Zeiger aufgefaßt, der auf die Speicherzelle mit dem auszugebenden Zeichen weist. Im Hauptprogramm wird dieser Zeichenzeiger zunächst auf das sechzehnte Zeichen der zweiten Bildschirmzeile (Adresse C02F) gerichtet, da der Drucker von rechts nach links arbeitet. Nach dem Druckvorgang wird er um 33 erhöht, damit man die nächsten 16 Zeichen erfaßt. Auf diese Weise wird jeweils eine halbe Bildschirmzeile fortlaufend ausgedruckt.

Im Druckerprogramm müssen zuerst die PIA-Anschlüsse als Ein- oder Ausgänge festgelegt werden. Das geschieht folgendermaßen: Die beiden Steuerregister der A- und B-Seite (Adressen 8005 und 8007) werden auf Null gesetzt, damit sind unter den Adressen 8004 und 8006 die Datenrichtungsregister (DRR) erreichbar. Jede „0" macht den entsprechenden Anschluß jetzt zum Eingang, jede „1" macht ihm zum Ausgang. Danach wird Bit 2 der Steuerregister auf „1" gesetzt, wodurch unter den Adressen 8004 und 8006 die Datenregister erreichbar sind. Da für das Druckkommando PRC die positive Flanke maßgebend ist, wird dieser Ausgang zunächst für eine gewisse Zeit zu Null gemacht, bevor er auf „1" gesetzt wird. Den entsprechenden Wert schreibt man in das Datenregister (in diesem Fall in das DR der B-Seite). Zu beachten ist, daß die Transistor-Anpassungsstufen das Ausgangssignal jeweils invertieren. Dasselbe gilt natürlich auch für die Eingänge. Der besseren Übersichtlichkeit halber wurden an etlichen Stellen des Programms deshalb die Daten vor der Ausgabe bzw. nach der Eingabe komplementiert (COM-Befehl).



◄ Bild 10.
Flußdiagramm und Befehlsfolge für das Drucker-Hauptprogramm

```
0005 CE C0 2E 7F
0009 80 05 7F 80
000D 07 86 FF B7
0011 80 04 86 01
0015 B7 80 06 86
0019 04 B7 80 05
001D B7 80 07 86
0021 01 B7 80 06
0025 01 01 01 01
0029 01 01 01 01
002D 01 01 01 01
```

Bild 11. Darstellung eines Speicherinhalts

Akkumulator B wird als Zeichenzähler benutzt und mit der Zahl 16 (dezimal) geladen. Da er in einer Schleife dekrementiert wird (ebenso wie der Zeichenzeiger), führt die Abfrage (BLT-Befehl) bei negativem Inhalt zu einem Sprung zur Marke N. Danach werden keine weiteren Zeichen mehr gedruckt, und PRC wird wieder „0".

Die Synchronisation der Zeichenausgabe mit dem Signal NXTCH erreicht man dadurch, daß man zuerst den Zustand „0" abfragt und die weitere Programmausführung so lange lahmlegt, bis er eingetreten ist. Erst danach wird der Zustand „1" abgefragt. Man erfaßt damit die positive Flanke und kann sofort nach deren Auftreten den Code für das nächste Zeichen bereitstellen.

Durch geringfügige Änderungen ist es mit diesem Programm z. B. möglich, einen Speicherinhalt übersichtlich darzustellen. *Bild 11* zeigt einen solchen Ausdruck, bei dem die ersten vier Zeichen eine Adresse und die nachfolgenden Zweiergruppen jeweils ein Byte der ab dieser Adresse enthaltenen Daten bedeuten. In diesem Fall hat sich das Programm selber aufgelistet.

Literatur

[1] *Blaseio, G.*: Mikrocomputer zum Selbstbau. ELEKTRONIK 1976, H. 9, S. 53...58.
[2] *Kreidl, J.*: Arbeitsweise von Debug-Programmen. ELEKTRONIK 1977, H. 6, S. 99...101.

Arbeitsweise von Debug-Programmen

Von den Mikroprozessor-Herstellern werden heute Mikrocomputer auf einer Leiterplatte zu Preisen angeboten, die weit unter 1000 DM liegen. Über eine genormte Schnittstelle wird ein Fernschreiber (oder ein anderes Terminal) angeschlossen, über den man das Programm in den Speicher „eintippt" und das Ergebnis der Arbeit ausdrucken läßt. Dafür, daß der Mikrocomputer die Anweisungen versteht, sorgt ein Hilfsprogramm, das üblicherweise in einen ROM von 1/2 oder 1 KByte abgespeichert ist und u.a. mit Monitor oder Debug-Programm bezeichnet wird. Eines der ersten Hilfsprogramme dieser Art entstand bei der Firma Motorola für den Mikroprozessor 6800 und wurde dort MIKBUG genannt. Die Firma PEP hat es um einige wichtige Funktionen erweitert und mit dem Namen PEPBUG versehen (inzwischen gibt es aber auch von Motorola erweiterte Versionen). Es bildet in dieser Form das „Betriebssystem" für eine Anlage, die sowohl zum Erstellen von Programmen als auch zum Programmieren von PROMs geeignet ist.

Grundsätzliche Arbeitsweise

Debug-Programme arbeiten im Prinzip alle nach demselben Schema, obwohl sie in bezug auf den Bedienungskomfort doch erhebliche Unterschiede aufweisen. Im *Bild* ist das Flußdiagramm für das PEPBUG-Programm dargestellt, das neben den Grundfunktionen „Speicherinhalt ändern", „Registerinhalte anzeigen", „Programmstart", „Programm auslesen" und „Programm einlesen" noch die außerordentlich nützlichen Möglichkeiten gestattet, das Programm in Einzelschritten durchzutasten und definierte Haltepunkte (*Breakpoint*) zu setzen.

Nach dem Restart, der auch beim Einschalten der Spannungsversorgung vorliegt, oder einem Software-Interrupt werden bestimmte Programmteile gestartet. Nachdem das Betriebssystem an den linken Protokollrand einen Stern gedruckt hat, kreist es in einer Schleife, bis der Benutzer ein Kommando gibt (das aus einem Buchstaben besteht). Wird einer der Buchstaben L, M, P oder B gedrückt, verzweigt das Programm zu den entsprechenden Unterprogrammen, druckt nach der Ausführung wieder einen Stern aus und erwartet die nächste Anweisung, indem es wieder die Warteschleife durchläuft. Nach den Buchstaben S oder G wird – nach Ausführung des zu testenden Programms – am Programmbeginn nach einem Software-Interrupt weitergemacht, d.h., es werden beispielsweise nach jedem Einzelschritt die Registerinhalte ausgedruckt.

Einen Software-Interrupt erreicht man mit einem Befehl (SWI), der beim Mikroprozessor 6800 speziell für Testzwecke realisiert wurde. Er wurde für die Funktionen „Einzelschritt" und „Haltepunkt" ausgenutzt, so daß keine Hardware-Zusätze erforderlich sind. Der SWI-Befehl wird im zu testenden Programm anstelle eines beliebigen Befehls eingefügt (bzw. anstelle des betreffenden Operations-Codes). Ist das Programm an dieser Stelle angelangt, werden sämtliche Registerinhalte im Stapelspeicher abgelegt, und der Prozessor macht an einer Adresse weiter, die in zwei festgelegten Speicherzellen hinterlegt ist. Das Debug-Programm sorgt nach dem Restart dafür, daß in diese Zellen die im Flußdiagramm mit SWI bezeichnete Startadresse geschrieben wird. Fügt der Benutzer also einen SWI-Befehl ein, wird das Programm an dieser Stelle angehalten, und die momentanen Registerinhalte werden ausgedruckt. Anschließend wartet das Debug-Programm auf eine neue Anweisung.

Bei der Version der Firma PEP übernimmt das Einfügen des SWI-Befehls das Hilfsprogramm: Nach jedem Drücken der Taste „S" z. B. wird der Operationscode des nächsten Befehls durch SWI ersetzt und nach der Ausführung des vorhergehenden Befehls und Ausdrucken der Registerinhalte wieder eingefügt.

Die einzelnen Funktionen

Zur Zwischenspeicherung von variablen Daten benötigt das Debug-Programm einen gewissen RAM-Bereich, den der Benutzer nicht verändern soll. Eine Ausnahme bilden bestimmte Zellen, die für folgende Parameter vorgesehen sind:

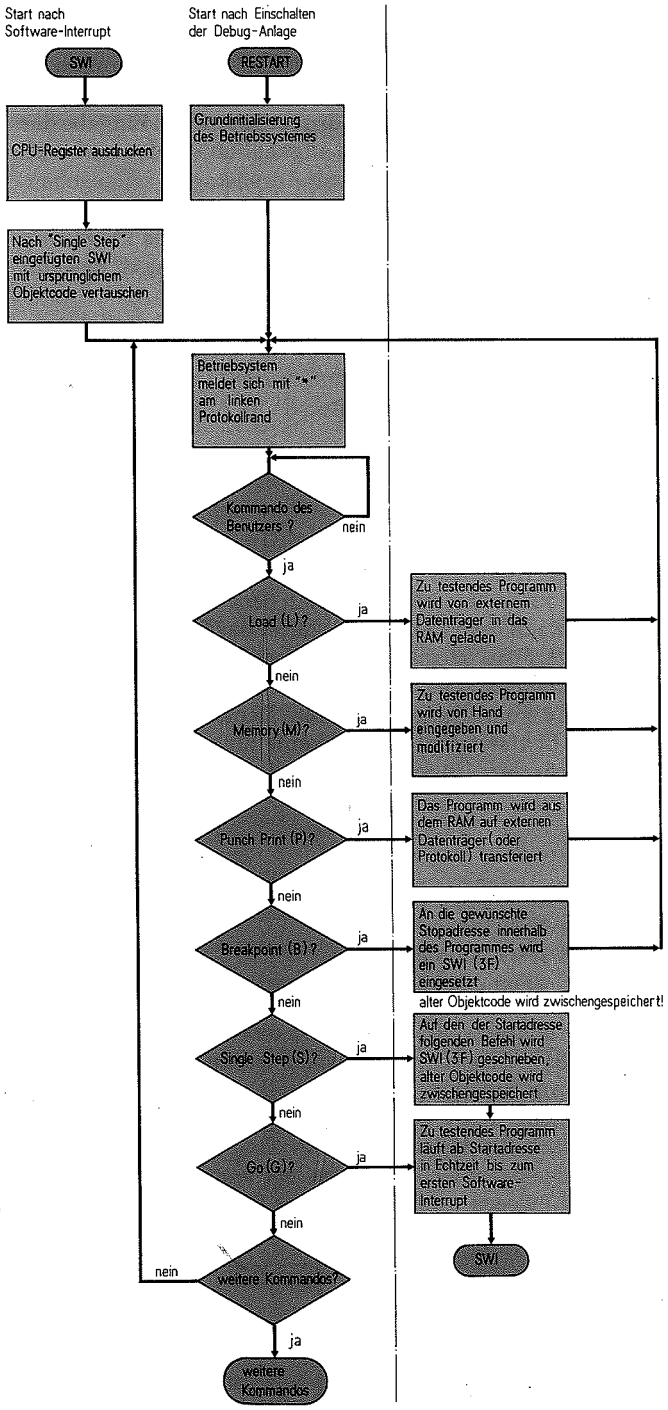
F000/F001: Startadresse (höher-/niederwertig) für Interruptanfrage (IRQ)
F002/F003: Anfangsadresse des Ausstanzbereiches
F004/F005: Endadresse des Ausstanzbereichs
F006/F007: Startadresse für nicht maskierten Interrupt (NMI)
F043: Zustandscode-Register
F044: Akkumulator B
F045: Akkumulator A
F046/F047: Indexregister
F048/F049: Programmzähler

Beim Auftreten eines maskierten (Interruptanfrage) oder nicht maskierten Interrupts verzweigt das Programm an die unter F000/F001 bzw. F006/F007 hinterlegte Adresse. Die Speicherzellen F043...F049 erlauben die Vorbesetzung der Register schon vor dem Programmstart.

In der folgenden Beschreibung der einzelnen Funktionen sind die vom Benutzer einzugebenden Zeichen unterstrichen, nicht unterstrichene Zeichen werden vom Mikrocomputer (µC) – gesteuert durch das Debug-Programm – selbständig ausgegeben.

Anzeigen und Verändern des Speicherinhaltes

Eingabe von „M" nach dem Stern – µC druckt Leerzeichen. Eingabe der 4stelligen sedezimalen Adresse – µC druckt Speicheradresse und -inhalt sedezimal aus. Eingabe eines beliebigen Zeichens außer „Leerzeichen" – µC druckt Adresse und Inhalt der nächsten Speicherzelle. Eingabe „Leerzeichen" und beliebiges Zeichen außer 0...F – µC



Das Flußdiagramm zeigt die Arbeitsweise eines Debug-Programms

druckt Stern, und es kann eine neue Funktion aufgerufen werden. Wird nach der vom µC ausgedruckten Adresse und dem betreffenden Speicherinhalt ein Leerzeichen eingetippt, kann der Inhalt dieser Zelle überschrieben werden. Nachdem die zwei Sedezimalzeichen eingegeben sind, druckt der µC Adresse und Inhalt der nächsten Speicherzelle aus. Die Funktionen „Anzeigen" und „Verändern" können beliebig ineinander verschachtelt werden.

Beispiel:
*M 0000
*0000 00.
*0001 0B.
*0002 01 0A
*0003 27 86
*0004 A2 .
*

Ausdrucken und Verändern der Registerinhalte

Eingabe „R" nach dem Stern – µC druckt die Registerinhalte des Mikroprozessors in folgender Reihenfolge aus: Zustandscode-Register, Akkumulator B, Akkumulator A, Indexregister, Programmzähler, Stapelzeiger (*Stackpointer*). Mit Ausnahme des Stapelzeigers können diese Registerinhalte in den dafür vorgesehenen Speicherzellen verändert werden.

Beispiel:
*R 76 EF 00 3F89 0000 F042 .

Programmstart

Eingabe der gewünschten Startadresse mit der M-Funktion in die Speicherzellen F048/F049, dann Eingabe von „G" nach dem Stern – µC verzweigt zur angegebenen Adresse. Eine Rückkehr vom Anwenderprogramm ist durch Drükken der Restart-Taste oder durch den Befehl SWI im Anwenderprogramm möglich. Wird der SWI-Befehl verwendet, dann druckt der µC automatisch die Registerinhalte aus.

Beispiel:
*M F048
*F048 12 00
*F049 47 00
*F04A 00 .
*G

Ausstanzen von Speicherbereichen

Diese Funktion wird verwendet, um Programme auf einen Datenträger wie Lochstreifen oder Magnetbandkassette zu übertragen. Zuerst werden Anfangs- und Endadresse des abzuspeichernden Bereiches unter den Adressen F002...F005 abgelegt. Nach dem Stern wird dann ein „P" eingegeben. Über den PIA-Baustein des Mikroprozessor-Systems werden die Daten in einem formatierten Code ausgegeben, der auch eine Prüfsumme enthält. Für den Anschluß eines Kassetten-Recorders ist zusätzlich ein Modem erforderlich.

Einlesen von externem Datenträger

Programme, die mit der P-Funktion auf einen externen Datenträger gespeichert wurden, können mit einem „L" nach dem vom µC ausgedruckten Stern wieder eingelesen werden. Die Eingangsleitung führt wieder über den PIA-Baustein. Das Programm wird im selben Bereich abgelegt, aus dem es ursprünglich ausgelesen wurde. Beim Auftreten eines Prüfsummenfehlers wird ein Fragezeichen ausgedruckt; soll der Ladevorgang trotzdem fortgesetzt werden, wird ein „L" eingegeben. Andernfalls ist der Datenträger zurückzusetzen und erneut ein „L" einzugeben. Die Eingabe von „S9" oder Restart beenden den Ladevorgang.

Einzelschritt

Programme können mit dieser Funktion schrittweise durchgetastet werden: Eingabe „S" nach dem Stern – µC startet das Programm ab der in den Zellen F048/F049 abgelegten Startadresse und führt genau einen Befehl aus. Danach werden die Registerinhalte ausgedruckt (Reihenfolge siehe R-Funktion). Das Programm kann mit „S" im Einzelschritt, mit „G" oder „T" fortgesetzt werden.

Beispiel:
*F048 00.
*F049 19 00
*F04A 00 .
*S
*F8 EF 00 3F89 0003 F042

Adreßstopp

Diese Funktion bietet die Möglichkeit, das Programm an bis zu fünf vorher festgelegten Adressen anzuhalten: Eingabe „B" nach dem Stern – µC druckt Leerraum. Eingabe der Adresse, an der das Programm angehalten werden soll – µC druckt Stern. Sind bereits fünf Adreßstopps eingegeben, druckt der µC nach „B" sofort einen Stern. Wenn der Programmlauf einen Adreßstopp erreicht, wird das Programm angehalten, und die Register werden ausgedruckt. Es kann mit „S", „T" oder „G" fortgefahren werden.

Eingabe „E" nach dem Stern eliminiert alle Adreßstopps. Zu beachten ist, daß Adreßstopps nur auf den Operationscode eines Befehls gelegt werden dürfen. Vor dem Ein- und Auslesen von Programmen müssen alle Adreßstopps eliminiert werden.

Überwachungslauf

Diese Funktion überwacht ein Programm von einem Einschaltpunkt (wird mit der B-Funktion gesetzt) bis zu einem Ausschaltpunkt (wird mit „C" nach dem Stern gesetzt): Eingabe „T" nach dem Stern – µC startet das Programm ab der unter F048/F049 abgelegten Adresse. Ab dem Einschaltpunkt werden nach jedem Befehl die Registerinhalte ausgedruckt, bis zum Ausschaltpunkt. Danach läuft das Programm in Echtzeit weiter. Sind keine Einschaltpunkte definiert, wird der Überwachungslauf ab der Startadresse durchgeführt.

Beispiel:

```
*T
*F8 EF 00 3F89 0003 F042
*F4 EF 00 3F89 0004 F042
*F4 EF 00 3F89 0005 F042
*F8 EF FF 3F89 0006 F042
```

Ein Überwachungslauf kann durch einen im Programm abgelegten SWI-Befehl unterbrochen werden, d. h., das Debug-Programm geht in die Grundstellung und druckt einen Stern aus. Ein- und Ausschaltpunkte werden wieder durch ein „E" nach dem Stern gelöscht. Sie dürfen nur auf den Operationscode eines Befehls gelegt werden und müssen vor dem Auslesen und Laden eines Programms eliminiert werden.

Optionen

Mit einer weiteren Version eines Debug-Programmes (µ PEPBUG) können u. a. relative Adressen berechnet sowie RAM-Bereiche getestet und in ein PROM übertragen werden.

Ing. Josef Kreidl

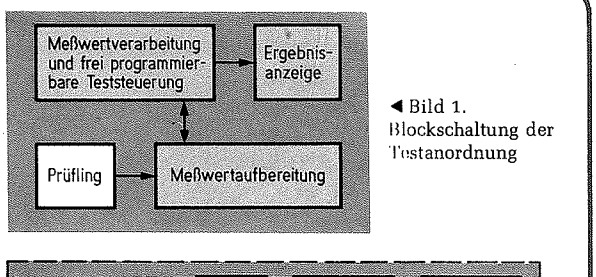
Einfache Programmiersprache für den SC/MP-Mikroprozessor

Zum Einsatz auf dem bekannten 8-bit-Mikroprozessor SC/MP hat die Firma National Semiconductor eine neue, leicht erlernbare Programmiersprache entwickelt; diese Programmierhilfe trägt die Bezeichnung NIBL (*National Industrial Basic Language*) und ist ähnlich aufgebaut wie die Standard-Programmiersprache BASIC. Damit soll vor allen Dingen denjenigen Entwicklern der Einstieg in die Software erleichtert werden, die bisher über keine Programmiererfahrung verfügten. Die Programmierhilfe steht Interessenten kostenlos zur Verfügung und kann in Lochstreifenform bezogen werden. In Kürze wird die komplette Software auch in ROMs erhältlich sein, für die zusätzlich eine Broschüre zum Selbststudium geplant ist.

☐ Vertrieb: National Semiconductor GmbH, Industriestr. 10, 8080 Fürstenfeldbruck, Tel. (0 81 41) 13 71.

Testautomat-Steuerung mit Mikroprozessor

Bei der Fertigung von elektronischen und mechanischen Geräten müssen Funktionen und Parameter von nichtspezialisierten Arbeitskräften schnell und zuverlässig geprüft werden. Mit einem Mikroprozessor kann ein preiswertes Gerät aufgebaut werden, das diese Forderung erfüllt und außerdem flexibel und leicht erweiterbar ist. *Bild 1* zeigt die einzelnen Baugruppen: Der Peripherie-Interface-Adapter (PIA) bildet die Schnittstelle zwischen dem Prozessorsystem und der Peripherie. Das Prozessorsystem besteht aus dem Programmspeicher (ROM), einem Schreib/Lesespeicher (RAM) und der Zentraleinheit (MPU = Mikroprozessor 6800 von Motorola) mit Steuer- und Versorgungseinheiten (*Bild 2*). Der PIA wird aufgrund der Architektur des Systems von der MPU wie ein Speicherplatz im RAM oder ROM behandelt. Die Peripherie enthält einen 8-bit-Analog/Digital-Umsetzer, einen Analogmultiplexer, eine Anzeigeeinheit sowie einen Vorwahlschalter, über den die Anzahl der Meßpunkte eingestellt wird. Bevor der Testablauf und das zugehörige Programm näher untersucht werden, soll der PIA MC 6820 eingehend erläutert werden, da er ein wesentliches Element für die Realisierung derartiger Schaltungen mit geringem Aufwand darstellt.



◄ Bild 1. Blockschaltung der Testanordnung

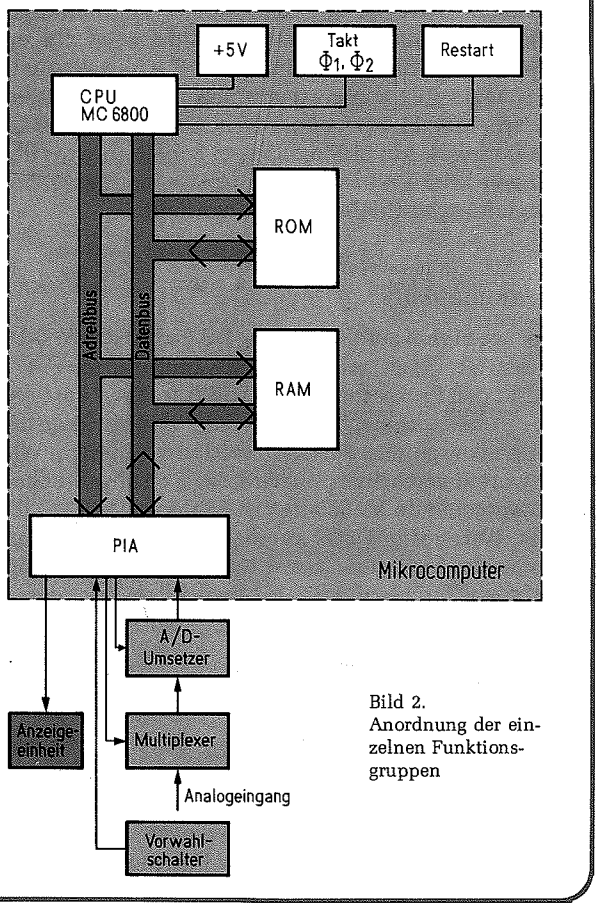


Bild 2. Anordnung der einzelnen Funktionsgruppen

Peripherie-Interface-Adapter

Der Baustein MC 6820 teilt sich in zwei weitgehend gleiche Hälften. Er enthält insgesamt sechs 8-bit-Register, von denen je drei auf die A- und auf die B-Seite entfallen (*Bild 3*):

1. *Datenregister*
   Das Datenregister ist das Transfer-Register für Daten, die vom Mikrocomputer-Datenbus zur Peripherie oder umgekehrt gelangen sollen. Werden über das Datenregister Daten ausgegeben, wirkt es als Latch. Eingeschriebene Daten stehen am Ausgang so lange an, bis ein neues Wort in das Register geschrieben wird. Werden Daten eingelesen, so wirkt das Register als frei schaltbares Gatter. Einzulesende Daten müssen während der Lesezeit statisch anstehen.
2. *Datenrichtungs-Register*
   Das Datenrichtungs-Register ist für die Richtungssteuerung des Datenflusses über das Datenregister verantwortlich. Jeder Leitung PA $(B)_0$...PA $(B)_7$ ist ein Bit zugeordnet. Ist dieses Bit „0", so ist die zugehörige Leitung als Eingang geschaltet, ist es „1", so ist die Leitung als Ausgang definiert. Es kann jede beliebige Bit-Kombination verwendet werden.

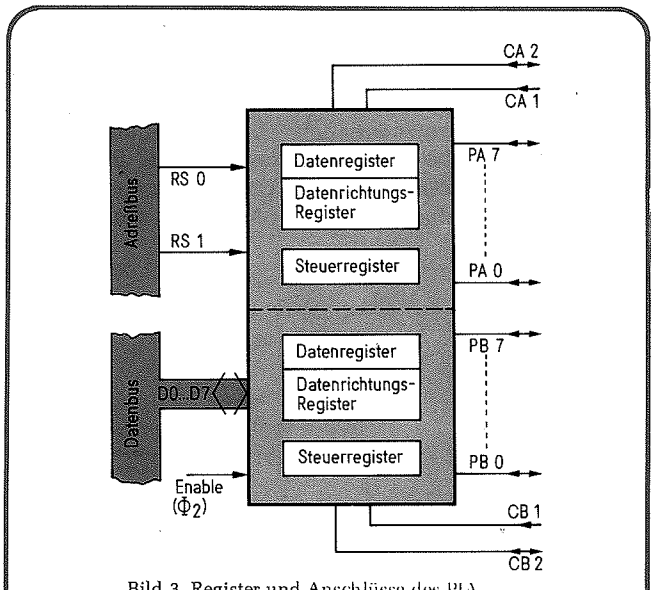


Bild 3. Register und Anschlüsse des PIA

Tabelle 1. Wirkung der einzelnen Bits (b_0...b_7) im Steuerregister A des PLA

	Bit = „0"	Bit = „1"
b_0	Sperrt über CA1 kommende Interrupts	Gibt Interrupt über CA1 frei (entsprech. Zustand von b_1)
b_1	Interrupt erscheint beim 1-0-Übergang an CA1	Interrupt erscheint beim 0-1-Übergang an CA1
b_2	Datenrichtungs-Register kann adressiert werden	Datenregister kann adressiert werden
b_3... b_5	siehe Tabelle 2	
b_6	Interrupt-Flag: wird gesetzt, wenn CA2 als Eingang geschaltet und ein aktiver Übergang erkannt wurde, wird automatisch nach einem Lesezyklus des Datenregisters rückgesetzt	
b_7	Interrupt-Flag: verhält sich wie b_6, jedoch für CA1	

Tabelle 2. CA2-Steuerung mit den Bits b_3...b_5 im Steuerregister A

b_5 = „1": CA2 als Ausgang geschaltet			b_5 = „0": CA2 als Eingang geschaltet
b_4	b_3		In diesem Fall haben b_4 und b_3 auf CA2 die gleiche Wirkung wie b_1 und b_0 auf CA1
0	0	CA2 geht nach dem ersten 1-0-Übergang der ENABLE-Leitung die einem MPU-Lesezyklus folgt, auf Null. Rückstellung durch nächsten aktiven CA1-Übergang	
0	1	Wie oben, jedoch Rückstellung durch nächsten 1-0-Übergang der ENABLE-Leitung	
1	0	CA2 geht auf Null, wenn b_3 = 0 in Steuerregister geschrieben wird	
1	1	CA2 geht auf Eins, wenn b_3 = 1 in Steuerregister geschrieben wird	

3. Steuerregister

Das Steuerregister ist für die Interrupt-Steuerung des PIA zuständig. Zusätzlich schaltet es die Adresse von Daten- und Datenrichtungs-Register um. Die *Tabellen 1* und *2* zeigen die Wirkungen der einzelnen Bits im Steuerregister des A-Teils. Da die Ausgangsstrukturen der B-Seite und der A-Seite voneinander abweichen, wurde eine unterschiedliche Interrupt-Verarbeitung gewählt. Wie aus *Bild 4* ersichtlich, ist der PIA auf der B-Seite in der Lage, höhere Lasten zu treiben als auf der A-Seite, wobei natürlich TTL-Last getrieben werden kann. Aufgrund dieses Unterschiedes wurde das Verhalten von CB2 (CA2) festgelegt. Sind die Steuerleitungen CB2 (CA2) als Ausgang geschaltet, so werden sie oft im „Hand-Shaking"-Betrieb (z. B. Rückmeldung einer akzeptierten Anforderung) verwendet. Wie schon erwähnt, ändert die Leitung CA2 ihren Zustand nach einem MPU-Lesezyklus des Datenregisters. Die CB2-Leitung hingegen ändert ihren Zustand nach einem Schreibzyklus in das Datenregister der B-Seite.

Adressierung des PIA

Der PIA wird über zwei Leitungen (RS 0, RS 1) an den Adreßbus des Mikroprozessor-Systems angeschlossen. Damit können vier interne Register direkt angewählt werden. Da jedoch sechs vorhanden sind, mußte ein Weg gefunden werden, wie man trotzdem alle sechs Register ansprechen kann. Diese Aufgabe übernimmt das Bit b_2 in den beiden Steuerregistern. Datenregister und Datenrichtungs-Register haben jeweils dieselbe Adresse. Wird in einem der Steuerregister b_2 gesetzt, dann kann das zugehörige Datenregister (DR) angesprochen werden; ist $b_2 = 0$, kann das Datenrichtungs-Register (DDR) „geladen" werden. Je nach Verdrahtung der höherwertigen sechs Adreßleitungen des Mikrocomputers ergibt sich dann die vollständige Adressierung des PIA, die z. B. folgendermaßen aussehen kann:

Adresse 8008: DR/DRR A-Seite
Adresse 8009: Steuerreg. A-Seite
Adresse 800A: DR/DRR B-Seite
Adresse 800B: Steuerreg. B-Seite

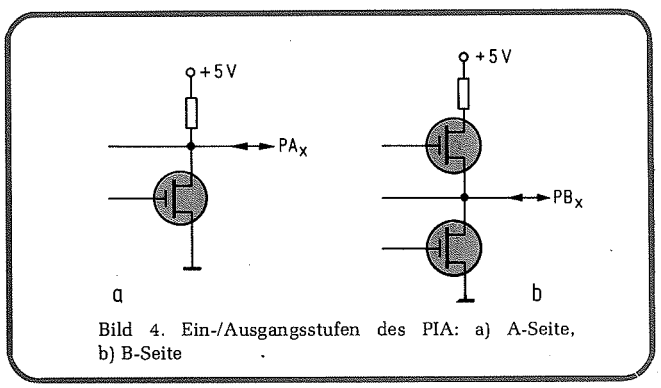


Bild 4. Ein-/Ausgangsstufen des PIA: a) A-Seite, b) B-Seite

Anschluß der Peripherie

In *Tabelle 3* ist die Verwendung der PIA-Leitungen zur Peripherie hin zusammengestellt, *Bild 5* zeigt die zugehörige Schaltung. Über die Leitungen PB 0 und PB 1 wird ein Analog-Multiplexer angesteuert, der jeweils ein Signal der Meßwerte MW 1...MW 4 auswählt und an den A/D-Umsetzer weiterleitet. Mit binärcodierten Schaltern wird über die Leitungen PB 5 und PB 6 die Anzahl der anzuwählenden Analogleitungen (Meßstellen) eingestellt.

Programmablauf

Über ein Signal auf der Leitung PB 2 wird der A/D-Umsetzer gestartet. Nach erfolgter Umwandlung fordert dieser über CA1 den Prozessor auf, die an PA 0...PA 7 anstehenden Daten zu übernehmen. Die Daten werden eingelesen und mit im RAM abgespeicherten Grenzwerten verglichen. Wird ein Grenzwert überschritten, so wird der Test abgebrochen und die entsprechende Meßstelle mit Fehlerart (MAX, MIN) angezeigt. Waren alle Werte innerhalb der Toleranz, stoppt der Tester an der letzten eingestellten Meßstelle und zeigt den fehlerfreien Durchlauf (PB 3, PB 4) an. *Bild 6* zeigt den Ablauf in einem Flußdiagramm.

Das entsprechende Programm für den Mikrocomputer stellt das Flußdiagramm (*Bild 7*) dar. In diesem Programm werden die internen Register des Mikroprozessors für folgende Aufgaben verwendet: Stackpointer (Stapelzeiger): definiert den Stapelspeicher (Registerinhalte bei Interrupt und Unterprogrammaufrufen retten).

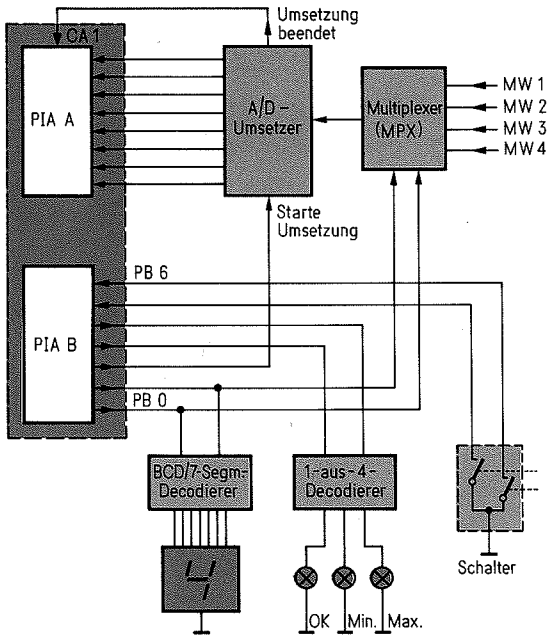
Indexregister: Adreßzeiger für Grenzwerttabelle.
Akkumulator A: enthält gelesenes Datenwort.
Akkumulator B: enthält Wert des Meßstellen-Schalters.

Tabelle 3. Verwendung der PIA-Leitungen

Leitung	Richtung	Verwendung
PA 0...PA 7	Eingang	Daten vom A/D-Umsetzer
CA1	Eingang	Fertigmeldung des A/D-Umsetzers
PB 0...PB 1	Ausgang	Steuerung des Analog-Multiplexers und der Meßplatz-Anzeige
PB 2	Ausgang	Startsignal für den A/D-Umsetzer
PB 3...PB 4	Ausgang	Ergebnisausgabe
PB 5...PB 6	Eingang	Meßstellen-Vorwahl

Aufgabe des Hauptprogrammes „Test" ist es, den PIA zu initialisieren und die Peripherie für jede Messung vorzubereiten. Im Interrupt-Programm (*Bild 8*) werden die Daten gelesen und verarbeitet, und bei Grenzwertüberschreitungen wird die entsprechende Anzeigelampe angesteuert. *Bild 9* zeigt den Aufbau der Grenzwerttabelle. Als Adreßbereiche wurden gewählt: Hauptprogramm ab 2000, Interrupt-Programm ab 203E, Grenzwerttabelle ab 2100, PIA ab 8008.

Den Assembler-Ausdruck für das komplette Programm zeigt *Bild 10*.



◀ Bild 5.
Anschluß der Peripherie an den PIA

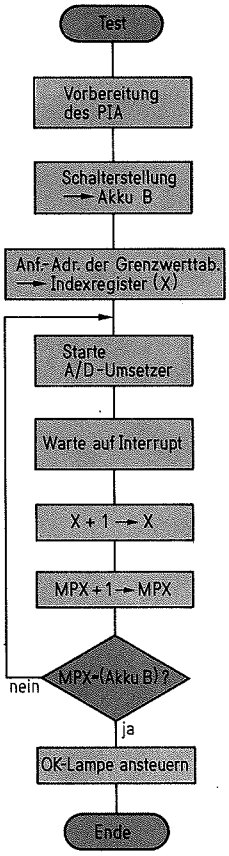


Bild 7. ▶
Flußdiagramm des Hauptprogrammes: (Akku B) ≙ Inhalt von Akkumulator B

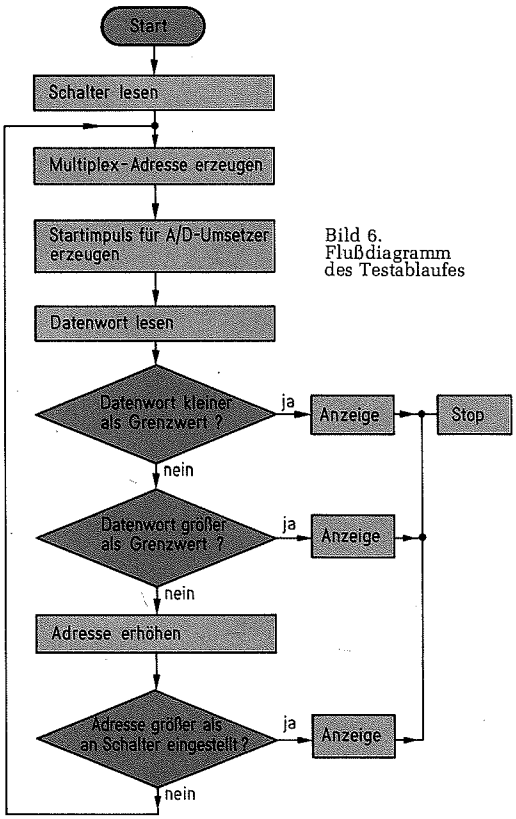
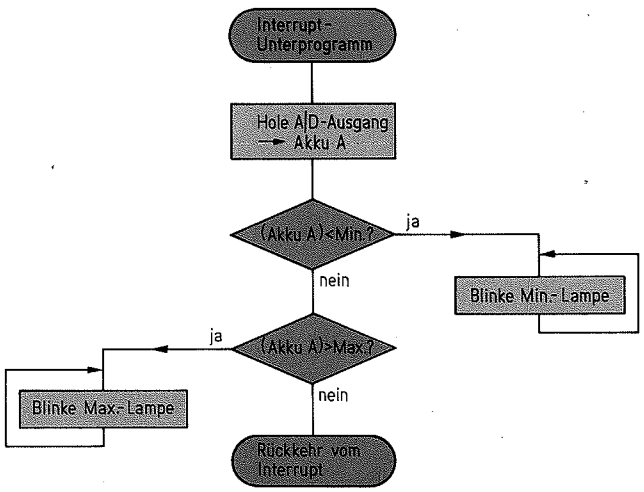


Bild 6.
Flußdiagramm des Testablaufes

Bild 8. ▼
Flußdiagramm des Interrupt-Programmes: (Akku A) ≙ Inhalt von Akkumulator A



```
PAGE  001   TEST


00001                       NAM    TEST
00002                       OPT    O          OBJEKTFILE ERSTELLEN
00003                  *
00004                  ********ADRESSZUWEISUNG********
00005       8008       PIAA   EQU    $8008
00006       800A       PIAB   EQU    PIAA+2
00007       2100       TABLE  EQU    $2100      GRENZWERTTAB.
00008       0000       MINI   EQU    0          OFFSET F MIN-WERT
00009       0010       MAXI   EQU    $10        "        MAX-WERT
00010                  *
00011                  ********PROGRAMMSTART********
00012 2000             *      ORG    $2000      STARTADRESSE
00013 2000 CE 203E     *      LDX    #INTERR    IRQ VEKTOR LADEN
00014 2003 FF FFF8     *      STX    $FFF8
00015 2006 0F          TEST   SEI                SETZE IRQ MASKE
00016 2007 8E 00FF     *      LDS    #$FF       STACK DEFINIEREN
00017                  *
00018                  *INITIALISIERUNG DER PIA
00019                  *
00020 200A CE 1F04     *      LDX    #$1F04
00021 200D FF 800A     *      STX    PIAB
00022                  *
00023                  *1F-> DATENRICHTUNGSREGISTER BIT0-4=EING,
00024                  *                           BIT5-7=AUSG
00025                  *04-> KONTROLLREG. UMSCHLTG AUF DATENREG.
00026                  *
00027 2010 86 07       *      LDA A  #7
00028 2012 B7 8009     *      STA A  PIAA+1
00029                  *
00030                  *7-> KONTROLLREG. A  UMSCHALTG AUF DATENREG.
00031                  *UND FREIGABE DES IRQ AN CA1 POS. FLANKE
00032                  *
00033 2015 F6 800A     *      LDA B  PIAB       SCHALTER NACH ACCU B
00034 2018 54          *      LSR B             RECHTSBUENDIG MACHEN
00035 2019 54          *      LSR B
00036 201A 54          *      LSR B
00037 201B 54          *      LSR B
00038 201C 54          *      LSR B
00039 201D CE 2100     *      LDX    #TABLE     TAFELADR.->X
00040                  *
00041                  *1SUS PULS AUF PB2 ERZEUGEN
00042                  *
00043 2020 B6 800A LOOP*      LDA A  PIAB
00044 2023 88 04       *      EOR A  #4
00045 2025 B7 800A     *      STA A  PIAB       BIT SETZEN
00046 2028 01          *      NOP               VERZOEGERN
00047 2029 01          *      NOP
00048 202A 01          *      NOP
00049 202B 01          *      NOP               BUS
00050 202C 88 04       *      EOR A  #4
00051 202E B7 800A     *      STA A  PIAB       BIT RUECKSETZEN
00052 2031 0E          *      CLI               IRQ MASKE FREI
00053 2032 3E          *      WAI               WARTE AUF IRQ
00054                  *
00055                  *INTERRUPT WIRD DURCH EOC GEMELDET
00056                  *
00057 2033 7C 800A     *      INC    PIAB       MPX+1->MPX
00058 2036 08          *      INX               X+1 ->X (TAFEL.ADR.)
00059 2037 84 03       *      AND A  #3         MPX->A
00060 2039 11          *      CBA               SCHALTERSTLLG ERREICHT?
```

```
PAGE  002   TEST


00061 203A 26 E4       *      BNE    LOOP       WENN NEIN NAECHSTE MESS.
00062 203C 20 FE       WAIT   BRA    WAIT       ENDE
00063                  *
00064                  *INTERRUPTPROGRAMM
00065                  *
00066 203E B6 8008 INTERR LDA A  PIAA       MESSWERT->A
00067 2041 A1 00       *      CMP A  MINI,X   A(MINI ?
00068 2043 25 05       *      BCS    ERROR1   FEHLER WENN JA
00069 2045 A1 10       *      CMP A  MAXI,X   A)MAXI ?
00070 2047 22 0F       *      BHI    ERROR2   FEHLER WENN JA
00071 2049 3B          *      RTI             ZURUECK INS HAUPTPROGRAMM
00072                  *
00073                  *FEHLER 1 BLINKE F1 LAMPE
00074                  *
00075 204A B6 800A ERROR1 LDA A  PIAB       LAMPENZUST.->A
00076 204D 88 12   LOOP1  EOR A  #18        BLINKE
00077 204F 8D 15       *      BSR    TIME-3   ZEITVERZ.
00078 2051 B7 800A     *      STA A  PIAB
00079 2054 88 08       *      EOR A  #8
00080 2056 20 F5       *      BRA    LOOP1
00081                  *
00082                  *FEHLER 2 BLINKE F2 LAMPE
00083                  *
00084 2058 B6 800A ERROR2 LDA A  PIAB
00085 205B 88 18   LOOP2  EOR A  #$18
00086 205D 8D 07       *      BSR    TIME-3
00087 205F B7 800A     *      STA A  PIAB
00088 2062 88 10       *      EOR A  #$10
00089 2064 20 F5       *      BRA    LOOP2
00090                  *
00091                  *160MS ZEITSCHLEIFE
00092                  *
00093 2066 CE 4E20     *      LDX    #20000     ZEITKONST.->X
00094 2069 09          TIME   DEX
00095 206A 26 FD       *      BNE    TIME       BIS X=0
00096 206C 39          *      RTS
00097                  *      END

TOTAL ERRORS 00000
```

Bild 10. Assembler-Ausdruck des gesamten Programms

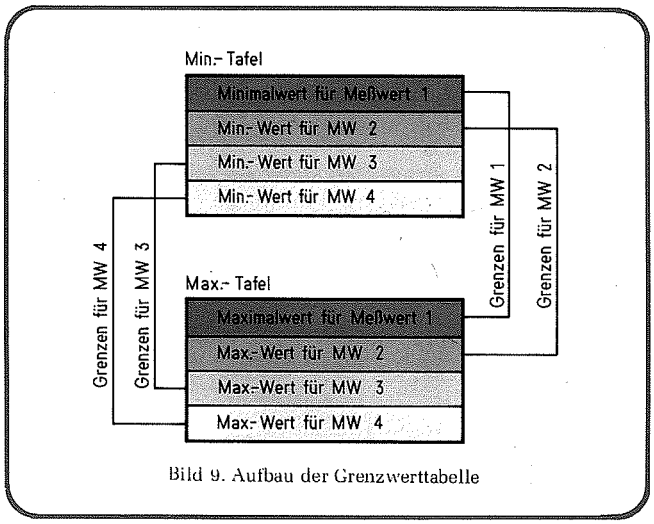


Bild 9. Aufbau der Grenzwerttabelle

Erweiterungsmöglichkeit

Das System verwendet einen PIA, damit können maximal vier Meßstellen abgefragt werden. Bei Verwendung eines zweiten PIA können bereits 100 Meßstellen abgefragt werden, und es stehen noch Leitungen für weitere Funktionen zur Verfügung. Die Änderung des Programms beschränkt sich auf unterschiedliches Adressieren der PIAs und ein Verlängern der Vergleichstabelle.

Die Analog/Digital-Umsetzung wurde außerhalb des Prozessors durchgeführt. Soll externe Hardware eingespart werden, kann die Wandlung auch zu großen Teilen vom Prozessor übernommen werden.

Dipl.-Ing. Hermann Lagrèze

Literatur

[1] *Tireford, H.*: Die Adressierungsarten bei Mikroprozessoren. ELEKTRONIK 1976, H. 12, S. 49...53.

[2] M 6800 Programming Reference Manual (Motorola).

[3] Applikationsbericht AN 757 (Motorola).

Dipl.-Ing. Hermann Lagrèze wurde in Göppingen geboren. Er studierte an der RWTH Aachen Nachrichtentechnik. Seit 1973 ist er bei der Motorola GmbH, GB Halbleiter, tätig. Er arbeitete anfangs als Applikationsingenieur für den Bereich Konsumelektronik. 1975 wechselte er in den Bereich Marketing Mikroprozessoren.

Hobbys: Filmen, Fotografieren, Kochen

Telefon: (0 81 06) 2 26 71

ELEKTRONIK-Leser seit 1970

Wenn es um Mikro-prozessoren geht, ist SASCO Ihr Partner

- Lernsysteme
- Entwicklungssysteme
- Einzelne Bauteile
- Prozessoren folgender Firmen:
  MOTOROLA
  6800, 2900, 10800 (ECL)
  NSC – SC/MP, PACE, IMP 16, 8080
  RCA – 1802 (CMOS)
  VALVO/SIGNETICS – 2650, 8×300
- Systemberatung
- Software, Firmware, Hardware
- Hauseigenes Mikroprozessor-Center

Vertrieb von elektronischen Bauelementen
Hermann-Oberth-Straße 16
8011 Putzbrunn bei München
Tel. 089/46 50 81 (Verwaltung)
Telex 05-29 504

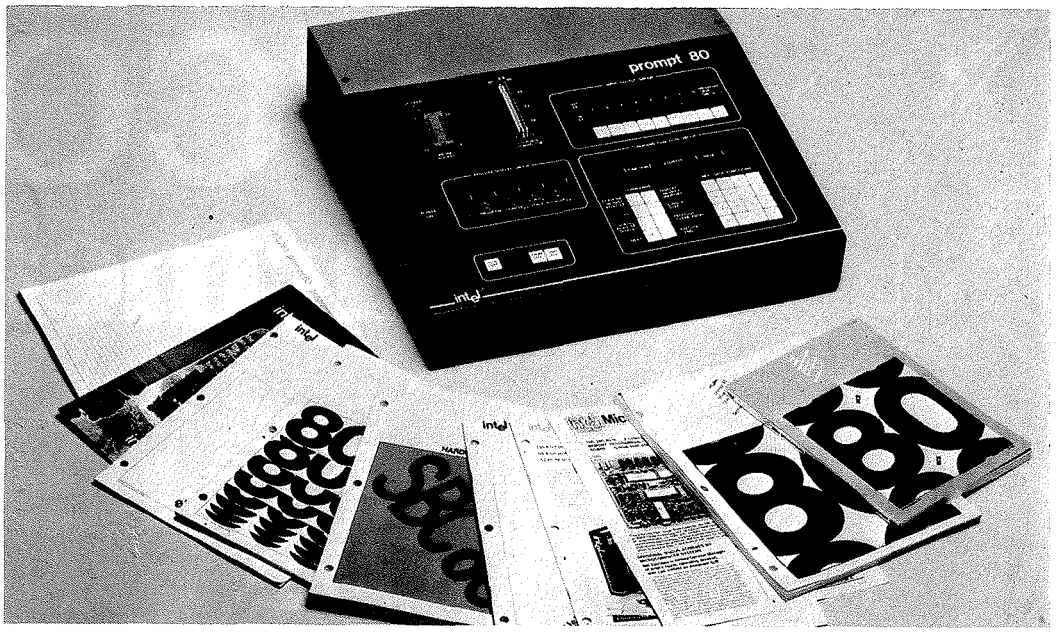
SASCO
DER DISTRIBUTOR

Hannover	Düsseldorf	Stuttgart	Nürnberg	München
Tel. 0511/86 25 86	Tel. 02150/14 33	Tel. 0711/24 45 21	Tel. 0911/204 152	Tel. 089/46 40 61-69
Telex 09-21123	Telex 08-53 695	Telex 07-23 936	Telex 06-23 097	Telex 05-29 504

Microcomputer-Entwicklungssystem programmiert auch EPROMs

Als Weiterentwicklung des Mikrocomputers SBC 80/10 stellt die Firma Intel ein verbessertes Entwicklungssystem für den Mikroprozessor 8080 A vor. Das Gerät trägt die Bezeichnung Intellec PROMPT 80 und ist von der Bedienung und Leistungsfähigkeit her mit einem programmierbaren Tischrechner vergleichbar. Die Bedienung erfolgt durch satzähnliche Befehls- und Dateneingaben über die Tastatur. Während der Programmeingabe oder des schrittweisen Programmablaufs kann der Anwender die internen Logiksequenzen detailliert verfolgen. Auf dem übersichtlichen Anzeigefeld werden dazu die jeweilige Speicheradresse, der Speicherinhalt sowie die Informationen der Ein/Ausgabe-Kanäle dargestellt. Durch diese Organisation kommt man beim Programmieren und Testen ohne Fernschreibmaschine und Datensichtgerät aus. Die interne Speicherbestückung setzt sich aus 1 K statischem RAM und 4 K ROM zusammen. In den ROMs ist ein Betriebsprogramm enthalten, das bei der Programmentwicklung die Kommunikation zwischen Anwender und Maschine ermöglicht. Außerdem enthält das PROMPT-80-System einen Programmiereinschub für EPROMs der Typen 8708 und 2704. In der vollen Ausbaustufe kann ein Speicherumfang von 64 K adressiert werden. Ferner ist die Erweiterung der Ein/Ausgabe-Kanäle auf insgesamt 256 Ports möglich. Die modulare Ausbaufähigkeit des Systems unter Verwendung der SBC-80-Moduln runden die Flexibilität und Leistungsfähigkeit dieses Programmierplatzes ab.

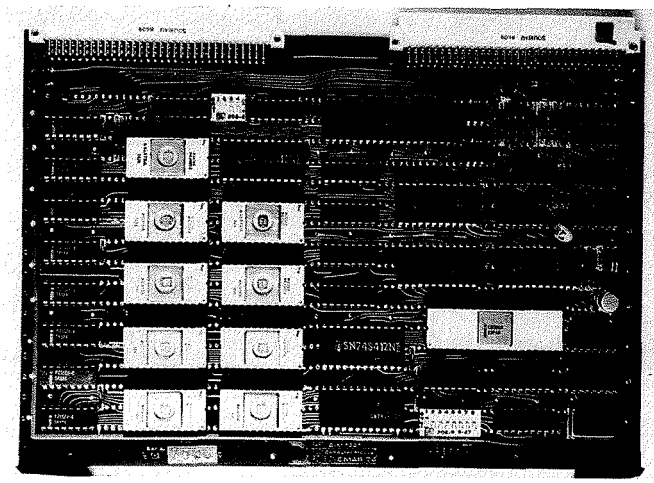
□ Vertrieb: Intel Semiconductor GmbH, Seidlstraße 27, 8000 München 2, Tel. (0 89) 55 81 41.



Mikrocomputer aus Norwegen

Die norwegische Firma A/S Mycron bietet ein Mikrocomputer-System an, das aus zahlreichen Funktionseinheiten besteht und mit umfangreicher Software geliefert werden kann. Die verschiedenen Ausbaustufen basieren alle auf der CPU-Einheit DIM 1001 (*Bild*). Sie enthält 1 KWorte RAM und Platz für 2 KWorte PROM sowie sieben Interrupt-Eingänge und einen UART-Ein-/Ausgabekanal. An Erweiterungskarten stehen z. B. eine schnelle Recheneinheit mit Gleitkomma-Arithmetik, Speichermoduln und verschiedene Ein-/Ausgabekarten zum Anschluß von Peripheriegeräten (Floppy Disk, Drucker, Modem, Bedienkonsole usw.) zur Verfügung. Weitere Ein-/Ausgabekarten, z. B. AD- und DA-Umsetzer, werden mit Optokopplern zur Potentialtrennung geliefert. An Software werden gegenseitig Monitor, Assembler, ein Dateisystem sowie ein BASIC- und ein PL/Micro-Übersetzer angeboten. PL/Micro ist eine höhere Programmiersprache ähnlich Intels PL/M.

□ Hersteller: A/S Mycron, P.O.Box 6199 Etterstad, Oslo 6, Norwegen, Tel. (02) 67 51 93.



Mikroprozessor für die Entwicklungsphase

Der Typ PIC 1650 von der Firma General Instrument ist ein Einchip-Mikrocomputer, der den ROM auf dem Chip enthält. Damit der Entwickler seine Programme testen kann, bevor er eine ROM-Maske erstellen läßt, hat die Firma jetzt den Mikroprozessor PIC 1664 entwickelt. Er ist in seinen gesamten Funktionen vollkommen identisch mit dem Typ PIC 1650, ist aber im 64poligen statt im 40poligen Gehäuse untergebracht, da die Adreßleitungen für einen externen PROM herausgeführt sind. Damit ist dieser Mikroprozessor natürlich auch für Kleinserien geeignet. Da eine der Ein/Ausgangsleitungen als zusätzliches Adreßbit verwendet wird, besteht die Möglichkeit, anstatt der 512 Befehle in der Einchip-Version 1024 Befehle speichern zu können.

□ Vertrieb: General Instrument GmbH, Nordendstr. 1a, 8000 München 40, Tel. (0 89) 28 40 31.

CMOS-Mikroprozessor-System erweitert

Als Ergänzung ihres 12-bit-Mikroprozessor-Systems mit der Zentraleinheit IM 6100 hat die Firma Intersil den Baustein IM 6102-MEDIC auf den Markt gebracht. Er enthält die Speichererweiterungslogik für 32 KWorte, die Steuerlogik für den simultanen direkten Speicherzugriff und einen programmierbaren Echtzeitgeber. Zusammen mit dem Mikroprozessor IM 6100 und den schon vorhandenen Peripherie-Bausteinen und Speichern kann der Entwickler mit dem neuen Bauelement Systeme realisieren, die nur mit CMOS-Elementen aufgebaut sind.

□ Vertrieb: Spezial-Electronic KG, Ortlerstr. 8, 8000 München 70, Tel. (0 89) 7 60 00 31.

Kassettensystem für Mikroprozessoren

Die Firma 3M bietet mit dem Miniatur-Digital-Kassettensystem DCD 1 eine kompakte Einheit an, die eine Datensicherheit von 10^{10} aufweist. Die parallele 8-bit-Schnittstelle ist speziell für Mikroprozessor-Systeme konzipiert. Man erreicht damit die ungewöhnlich hohe Datenübertragungsrate von 2530 Byte/s. Zusammen mit der 3M-Digital-Kassette DC 100 A bietet das Laufwerk Problemlösungen auf engstem Raum für Kassenterminals, Prozeßsteueranlagen usw.

□ Vertrieb: 3M Deutschland GmbH, Postfach 643, Carl-Schurz-Str. 1, 4040 Neuss 1, Tel. (0 21 01) 1 41.

Dipl.-Ing. Dirk Jansen

Mikroprozessor steuert Kartenanzeigegerät

Navigationsanlagen in Flugzeugen, Hubschraubern und Oberflächenfahrzeugen sollen dem Fahrzeugführer die ermittelte Position in leicht faßbarer Weise darstellen. Anschaulicher als eine reine Ziffernanzeige (z. B. in geografischen Koordinaten) ist die Anzeige auf einer maßstabsgetreuen Landkarte [1, 2]. Sie erlaubt eine ständige Kontrolle der Navigationsanlage durch Vergleich der Anzeige mit markanten Geländepunkten der Umgebung. Dazu wird ein genaues, an bestehende Navigationsanlagen anschließbares Kartenanzeigegerät benötigt. Die gefundene Lösung enthält einen steuernden Mikroprozessor, Schrittmotoren als Stellglieder und optoelektronische Impulsgeber zur Rückmeldung; sie ist auch anderweitig universell verwendbar, z. B. bei Maschinensteuerungen. Auch die zur Steuerung der Schrittmotoren erforderliche Software wird beschrieben.

1 Aufgabenstellung

Das Gerät erhält von der Fahrzeugnavigationsanlage über einen digitalen Datenkanal die UTM-Koordinaten[1]) des momentanen Standorts. Die jeweils 5stelligen Koordinatenwerte müssen von der im Kartengerät integrierten Elektronik ausgewertet und in die Anzeigekoordinaten umgerechnet werden. Dabei sind unterschiedliche Maßstäbe der Karten und eine mögliche Umschaltung der Nordrichtung (parallel zur langen oder kurzen Kante des Geräts) zu berücksichtigen. Das Gerät soll zudem über eine Selbsttesteinrichtung verfügen und harten Umweltbedingungen standhalten können.

[1]) Universale Transversale Mercator-Koordinaten, ein vorwiegend im militärischen Bereich verwendetes Gittersystem, das auf einer Zylinderprojektion der Erdoberfläche beruht.

2 Grundkonzept der Elektronik

Der kompakte Aufbau des Geräts (*Bild 1*) und die geringe Bauhöhe von 40 mm lassen der Elektronik wenig Raum. Sie ist auf vier Leiterplatten der Größe 100 x 200 mm² in der Bodenbaugruppe (*Bild 2*) versenkt angeordnet. Die elektrischen Verbindungen erfolgen über Flachbandkabel.

Der zur Verfügung stehende Raum zwingt zu einer Konzentration der elektronischen Funktionen. Berücksichtigt man, daß bei der Koordinatenberechnung mehrere 5stellige Werte gespeichert, umgeformt, subtrahiert und addiert werden müssen, ist ein Aufbau in konventioneller Technik mit Standardschaltungen nicht mehr möglich. Eine Abschätzung ergab einen Aufwand von mindestens 140...160 integrierten Schaltungen. Die Packungsdichte bringt Zuverlässigkeits- und Wärmeprobleme; die Stromversorgung ist aufwendig.

Der Entschluß für den Einsatz eines Mikroprozessors fiel leicht, da heute solche Bausteine von hoher Zuverlässigkeit und großem Temperaturbereich zur Verfügung stehen [3]. Die für MOS-LSI-Bausteine in der Literatur genannten Fehlerausfallraten liegen zwischen $0{,}1/10^6$ h für gute Industriequalität [4] und $0{,}06/10^6$ h für nach MIL-STD-883 Level B getestete Typen [5, 6]. Der Mikroprozessor Intel 8080 A wurde u. a. gewählt, weil dieser in entsprechender Qualität lieferbar, auf dem Markt weit verbreitet ist und über gute parallele Ein-/Ausgabe-Schaltungen (z. B. Baustein 8255) verfügt. Zudem erleichtern der Aufbau des Befehlscodes und die absolute Adressierung das Programmieren, wenn nur einfache Hilfsmittel zur Verfügung stehen (KIT 8080 und TTY).

Das Grundkonzept zeigt *Bild 3*. Im Mittelpunkt der Schaltung steht der Mikroprozessor, der über die EIN/AUSGABE

Bild 1. Außenansicht des Kartengeräts

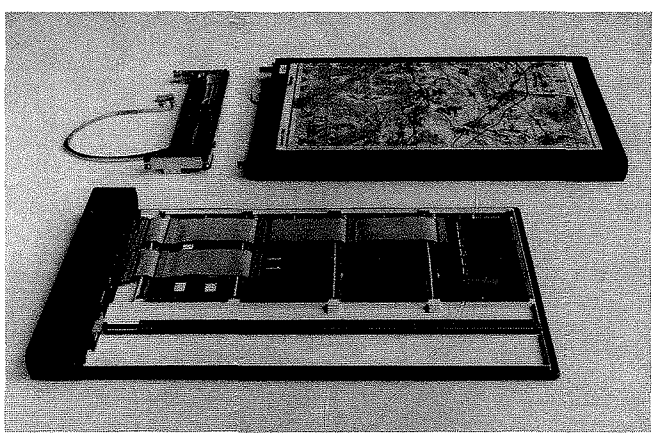


Bild 2. Das geöffnete Gerät. Die Elektronik ist in die Bodenbaugruppe eingelassen

Mit der peripheren Seite des ACIA ist ein Mikrocomputer in der Lage, ein peripheres Anschlußgerät mit asynchron-seriellem Datenformat optimal zu steuern. Dieses Format wird in der Regel für Systeme mit kleiner und mittlerer Übertragungsgeschwindigkeit mit 1800 bit/s und weniger benützt (ist aber nicht darauf beschränkt). Ein Fernschreiber, der eine Übertragungsgeschwindigkeit von 110 bit/s oder 10 Zeichen/s aufweist, ist ein typisches Beispiel eines langsamen Terminals. Zwischen ihm und einem ACIA wird ein Interface benötigt, um die TTL-kompatiblen Pegel des ACIA auf den Strombedarf von 20 mA des Fernschreibers umzusetzen. Ein Fernschreiber-Interface für nicht komplementierte Daten bedarf eines Optokopplers (z. B. 4N33) mit darauf folgenden Invertierern (*Bild 2*). Andere Fernschreiberausführungen (z. B. RS-232) können leicht mit Hilfe der RS-232-Interface-Bausteine MC1488 und MC1489 mit dem ACIA zusammengeschaltet werden (*Bild 3*).

2 Betrieb mit einem Modem

Der ACIA besitzt die Möglichkeit, die Datenübertragung zu oder von einem Fernterminal über ein Modem und Telefonleitungen zu steuern. Als Modem kann der langsame Typ MC6860 (*Bild 4*) eingesetzt werden. Er arbeitet mit der Methode der Modulation durch Frequenz-Ein- und Ausschaltung (FSK) zum Übertragen von Daten mit einer Geschwin-

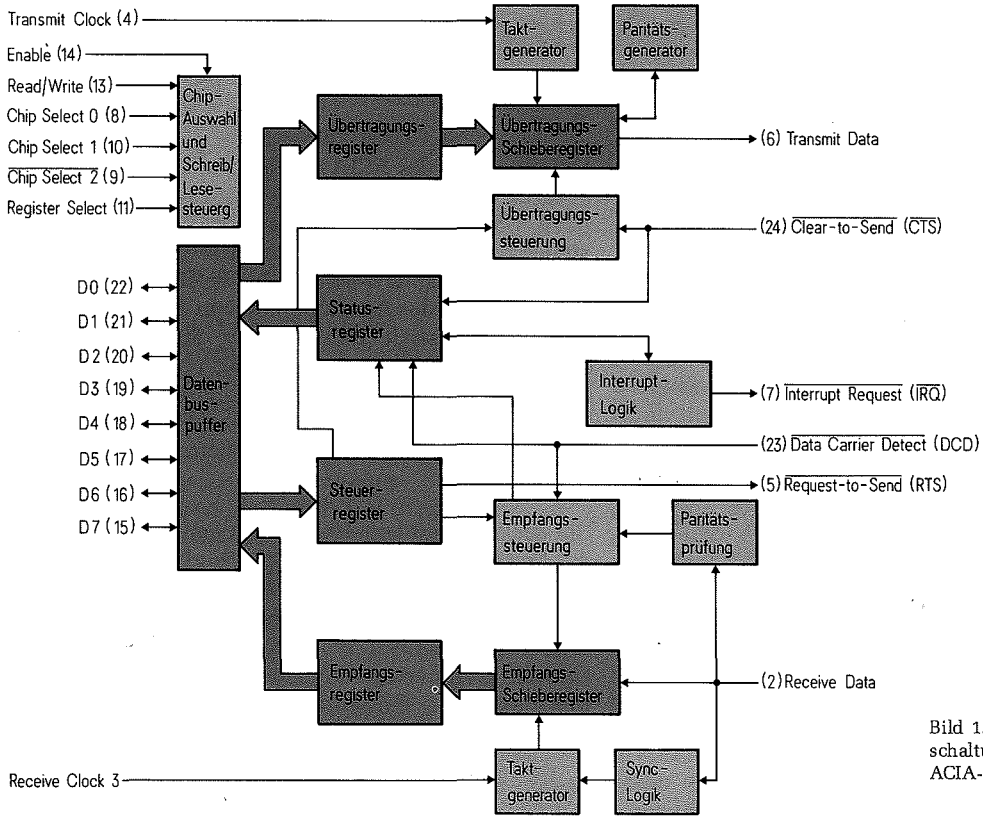


Bild 1. Block-schaltung des ACIA-Bausteins

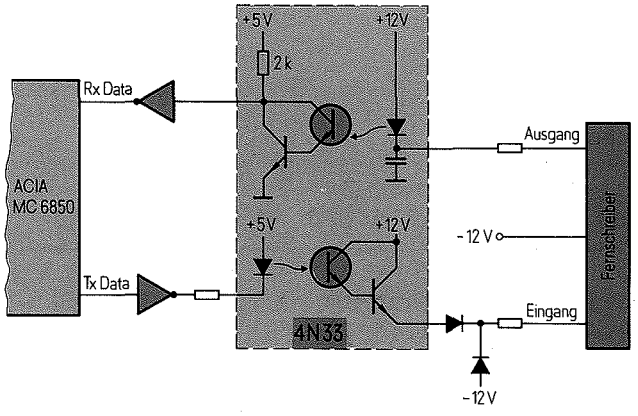


Bild 2. Fernschreiber-Interface mit Optokoppler

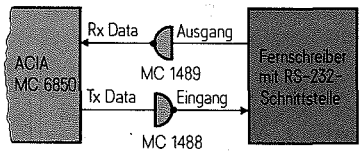


Bild 3. RS-232-Interface

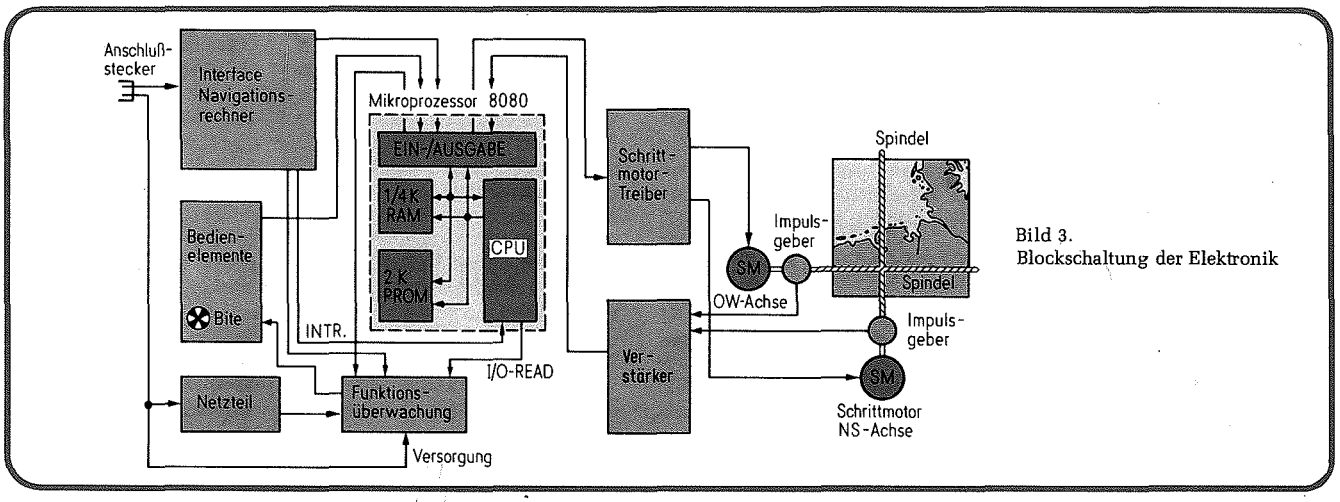


Bild 3.
Blockschaltung der Elektronik

mit den übrigen Funktionsblöcken korrespondiert. Alle Bedienelemente, Zifferneingabeschalter und Tasten sind unmittelbar auf die EINGABE geführt. Die vom Navigationsrechner übertragene Information wird in einem Interface aufbereitet und vom Mikroprozessor interruptgesteuert übernommen. Die AUSGABE steuert unmittelbar die Schrittmotortreiber an, die den Strom in den jeweils erregten Wicklungen freigeben. Das dazu für jeden Schritt berechnete Bit-Muster wird vom Programm erzeugt (s. Abschnitt 4).

Auf den Achsen der Schrittmotoren sind Impulsgeber angeordnet, die für jede der möglichen vier Schrittmotorstellungen einen 2-bit-Code ausgeben, der wiederum verstärkt und vom Mikroprozessor eingelesen wird. Dies dient, wie später noch näher erläutert wird, der Kontrolle der Schrittmotoren auf Schrittverlust.

Im Netzteil werden alle zum Betrieb erforderlichen Spannungen erzeugt. Die absolute Höhe dieser Spannungen und weitere, vom Prozessor und vom Interface abgeleitete Signale werden von der Funktionsüberwachungseinheit betrachtet. Bei Störung steuert diese Einheit den BITE-Indikator (*Built In Test Equipment*) an.

3 Schaltungen zur Selbstdiagnose

In einem Gerät, das aus vielen einzelnen Bauteilen unterschiedlicher Zuverlässigkeit besteht, ist die Wahrscheinlichkeit eines Fehlers bestimmbar aus der Zuverlässigkeit der einzelnen Bauelemente [7, 8]. Oftmals sind die Daten insbesondere der mechanischen Komponenten nicht bekannt oder nur mit großem Aufwand zu beschaffen. Da ein Ausfall des Gerätes nicht unmittelbar lebensbedrohend wirkt, kann mit dem Austausch von Gerätekomponenten gewartet werden, bis ein tatsächlicher Fehler auftritt. *Voraussetzung dafür ist allerdings, daß der Fehler sofort erkannt und angezeigt wird.* Der Bedienende, normalerweise mit anderen Aufgaben belastet, muß mit einem Blick die Funktionsfähigkeit des Gerätes feststellen können. Das Gerät enthält dazu Schaltungen, die alle Baugruppen kontinuierlich überwachen und jede Fehlfunktion zur Anzeige bringen.

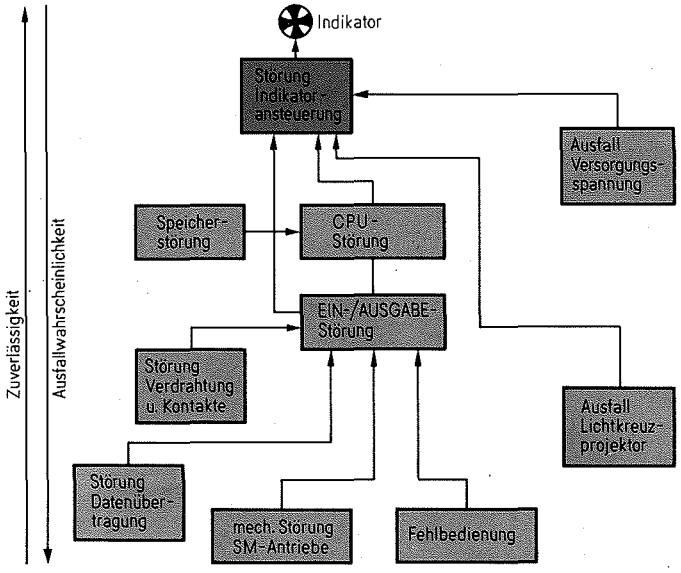


Bild 4. Mögliche Störfälle, geordnet nach der Wahrscheinlichkeit ihres Auftretens

Zur Konzeption der Selbsttesteinrichtung ist zunächst eine Ordnung der möglichen Störanfälle nach der Wahrscheinlichkeit ihres Auftretens durchzuführen (*Bild 4*). Eine mechanische Störung der Schrittmotorantriebe, hervorgerufen durch Verschmutzung, Verschleiß, Spindelblockierung usw., kann leichter eintreten als eine Unterbrechung in der Verdrahtung oder ein Ausfall des Mikroprozessors. *Die Überwachung von Baugruppen mit höherer Ausfallwahrscheinlichkeit erfolgt durch solche mit geringerer Ausfallwahrscheinlichkeit.* Die Einheit, die schließlich alle Signale zusammenfaßt und den Indikator ansteuert, ist aus wenigen diskreten Bauelementen hoher Zuverlässigkeit aufgebaut und besitzt eine eigene Stromversorgung.

Da die Ausfallwahrscheinlichkeit des Mikroprozessors als niedrig anzusehen ist, kann er gut zur Kontrolle anderer Baugruppen dienen. So kontrolliert er die Schrittmotoren, die Informationsübertragung vom Interface, die Verbindungsleitungen zu den Bedienelementen (über Paritätsvergleich) und führt einen Speichertest durch, der eine Aussage über die Funktion des RAM erlaubt. Wird eine Fehlerbedingung detektiert, geht das Programm in eine unendliche Schleife und ein Signal wird an die Überwachungsschaltung abgegeben. Die Funktion des Mikroprozessors selbst wird durch folgendes Mittel kontrolliert: Die im Programm häufig auftretenden Einleseoperationen über die EIN/AUSGABE lösen auf der READ-Leitung Impulse aus, die zur Überwachungseinheit geleitet werden. Dort triggern sie ein Monoflop, das so lange gekippt bleibt, wie die Impulse regelmäßig eintreffen. Bei Ausfall der Taktversorgung fallen diese Impulse aus. Bei allen anderen denkbaren Prozessorfehlern, wie Speicherstörung, Kurzschluß zwischen BUS-Leitungen, Ausfall von BUS-Treibern usw., wird mit größter Wahrscheinlichkeit der mit dem PROM bestückte Speicherraum verlassen und damit werden unsinnige Befehle eingelesen. Eine Rückkehr zu sinnvollen Befehlsabläufen ohne

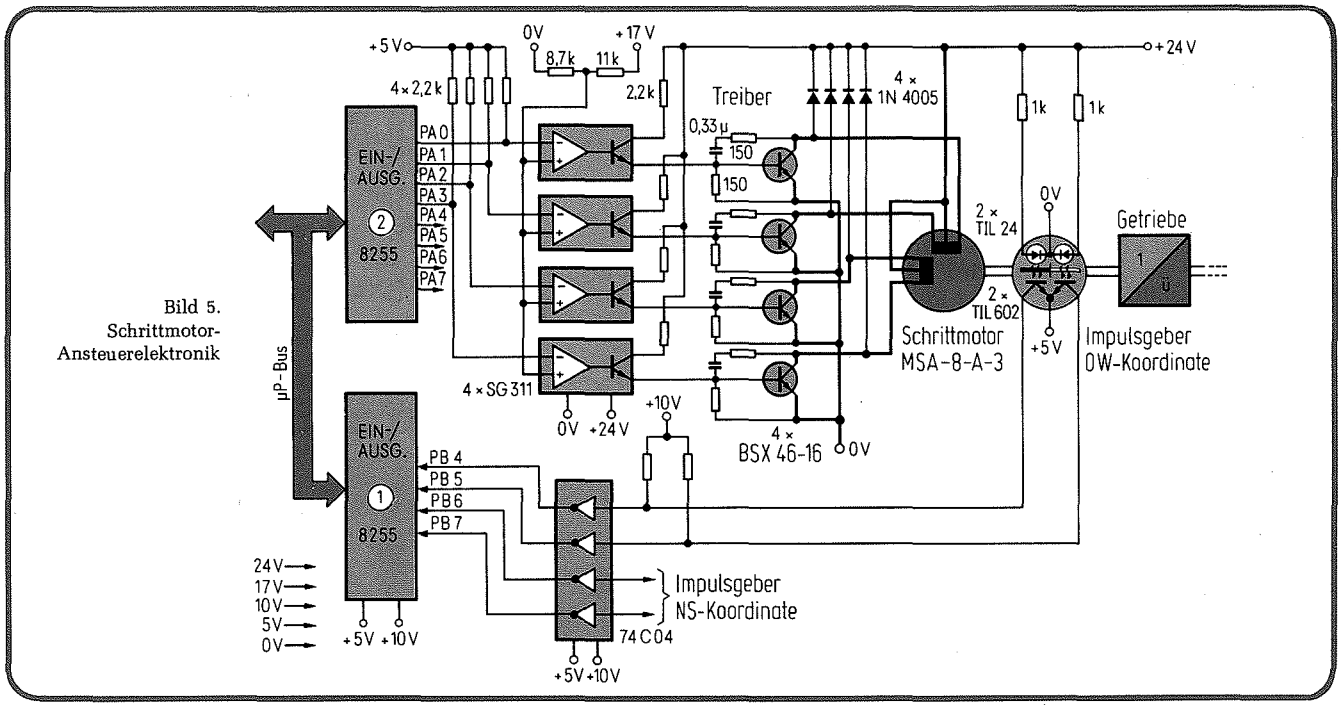


Bild 5. Schrittmotor-Ansteuerelektronik

äußeren Eingriff (Reset, Interrupt) ist sehr unwahrscheinlich. Auch in diesem Fall treten nach der Störung keine Pulse mehr auf der READ-Leitung auf und die BITE-Kontrolle spricht an. Eine 100%-Prozessorkontrolle wäre sehr aufwendig und erfordert eine Dreifachauslegung des Mikroprozessors [6].

Fensterkomparatoren überwachen die Versorgungsspannungen und sprechen bei einer Abweichung von mehr als ±0,5 V an. Die Lichtkreuz-Projektorlampen, die aufgrund der hohen Wärmeentwicklung nur eine begrenzte Lebensdauer haben, werden über den Versorgungsstrom kontrolliert.

Alle diese Maßnahmen tragen dazu bei, die Entdeckung eines Fehlers so wahrscheinlich wie möglich zu machen. Neben dieser kontinuierlich arbeitenden Funktionsüberwachung gibt es noch den Betriebszustand „TEST", der durch Umlegen eines Schalters aktiviert wird. In diesem Zustand werden statt der Koordinaten des augenblicklichen Standortes die Koordinaten jenes Punktes eingespeist, der zuvor zur Eichung des Gerätes verwendet wurde und auf dem Kartenblatt markiert ist. Die Schrittmotoren werden nun so gesteuert, daß bei richtiger Funktion aller Elemente dieser Punkt wieder angezeigt wird. *Da nahezu alle Bauelemente an dieser Nachlauffunktion beteiligt sind, hat dieser Test einen großen Aussagewert.*

Der BITE-Indikator, eine Kontrolleuchte (LED), ist so geschaltet, daß bei eingeschaltetem Gerät eine Daueranzeige erfolgt. Bei Störung wird die kontinuierliche Anzeige durch ein Blinken des Indikators abgelöst. Zur Unterscheidung zwischen einer Störung, die durch einen Gerätedefekt ausgelöst wird, und einer Störung, die durch Fehlbedienung oder Überschreiten des Anzeigebereichs verursacht wird, sind zwei Blinkfrequenzen vorgesehen.

Die BITE-Anzeige wird erst durch das Ausschalten des Geräts gelöscht.

4 Steuerung der Schrittmotorantriebe

Unter dem Aspekt der Zuverlässigkeit stellen die Schrittmotorantriebe die schwächste Baugruppe dar. Bei mechanischen Störungen können Schritte verloren gehen, die Fehler summieren sich auf und die Anzeige wird verfälscht. Zur Kontrolle der Schrittausführung ist eine Rückmeldung erforderlich, die durch einen Winkelcodierer erfolgen kann. Eine Lösung mit einem 13-bit-Winkelcodierer wird in [9] beschrieben. Wenn jedoch keine absolute Zuordnung zwischen Gehäuse und Anzeigeposition gefordert wird, reicht es aus, jeden einzelnen Schritt nach dem Muster „Kommando – Quittung" zu kontrollieren. Als Rückmeldung kann dann einfach ein Drehflügel dienen, der eine Lichtschranke abschattet.

Bild 5 zeigt die Schaltung eines der beiden Schrittmotorantriebe mit dem Impulsgeber. Die Motorspulen werden über Schalttransistoren angesteuert. CMOS-Inverter verstärken die Signale der Fototransistoren. Diese Bausteine sind ohne weitere Logik mit den Ein/Ausgabe-Schaltungen des Mikroprozessors verbunden; sein Programm erzeugt die entsprechenden Ansteuermuster und schaltet die Motoren ein und aus.

Der hier verwendete Schrittmotor vom Typ Clifton MSA-8-A-3 erlaubt 8 Schritte/Umdrehung. *Bild 6* zeigt das Einschwingverhalten des Motors bei geringer Dämpfung und träger Last. Der Überschwinger kann größer als ein Schritt sein. Der Schrittmotor ist fest mit dem Impulsgeber gekoppelt, dessen Auslesung abhängig vom Winkel ebenfalls in

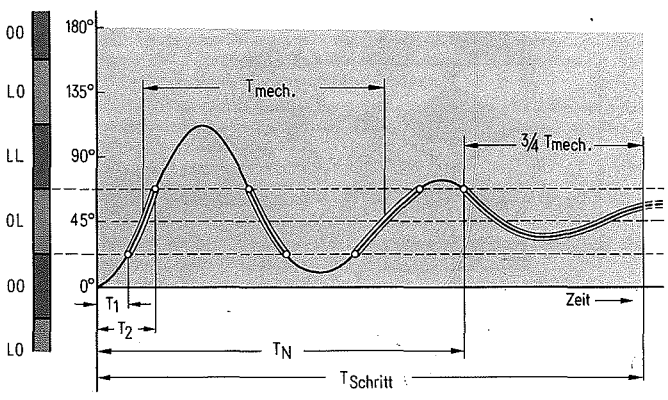


Bild 6. Einschwingverhalten des Schrittmotors bei geringer Dämpfung

Bild 6 eingetragen ist. Aus der Stellung $0^0 \triangleq 00$ gelangt der Schrittmotor nach einem Schritt in die Stellung $45^0 \triangleq 0L$. Zu jedem Zeitpunkt ist also die Stellung der Motorachse bekannt. Das entsprechende Ansteuermuster für die Schrittmotorspulen kann daraus abgeleitet werden.

Bild 7 zeigt ein vereinfachtes Ablaufdiagramm des Unterprogramms STEP, das die Schrittmotoren kontrolliert; es bedient beide Schrittmotore simultan und leitet aus dem Anfangszustand der Register A bis L Drehrichtung und Drehgeschwindigkeit ab. Zu Beginn wird aus der gespeicherten letzten Motorstellung die neue Motorstellung für beide Koordinaten berechnet. Das daraus bestimmte Ansteuermuster für die Schrittmotorwicklungen (Unterprogramm TRLT) wird ausgegeben. Nun wird eine definierte Zeit (Unterprogramm TIME) lange gewartet, in der die Schrittmotore sich auf den neuen Zustand einstellen können. Die Variation dieser Grundzeit erlaubt, die Drehgeschwindigkeit der Motore zu beeinflussen. Anschließend werden die Zähler H und L gesetzt. Die nun beginnende *1. Schleife* prüft, ob der berechnete Sollstand von den Motoren erreicht wird. Ist dies nach 20 Durchläufen noch nicht geschehen, wird das Programm abgebrochen und eine Fehlermeldung ausgelöst (Spindel blockiert). Dieser Teil des Programms berücksichtigt die Anlaufphase der Schrittmotore.

Der nun folgende Programmabschnitt beruht auf der Vorstellung, daß Schrittmotore bei ungünstigen Lastbedingungen erheblich überschwingen und in Resonanz geraten können. So ist es möglich, daß der Sollwert zwar kurzzeitig erreicht, aber dann sofort wieder verlassen wird. Dieses Ver-

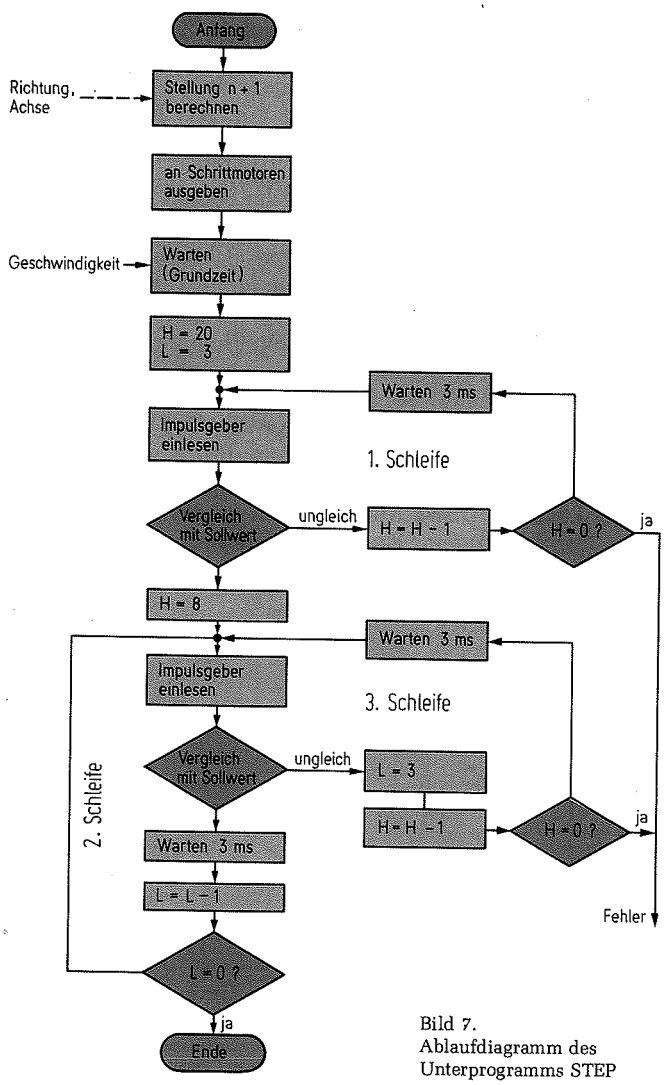


Bild 7.
Ablaufdiagramm des Unterprogramms STEP

Bild 8. Programmlisting des Unterprogramms STEP

```
Unterprogramm STEP zur simultanen Ansteuerung von 2 Schrittmotoren

Anfangszustand
der Register  : A enthält : 00  für keinen Schritt NS-Koordinate
                           01  für  einen Schritt positive Richtung
                           FF  für  einen Schritt negative Richtung

                B enthält : 00  für keinen Schritt ØW-Koordinate
                           01  für  einen Schritt positive Richtung
                           FF  für  einen Schritt negative Richtung

                C enthält : Zahl zwischen 00 und FF, gibt die
                            Grundverzögerungszeit an

                D enthält : die letzte, an den ØW-Schrittmotor ausge-
                            gebene Stellung PØSX, z.B. 00, 01, 02, 03

                E enthält : die letzte, an den NS-Schrittmotor ausge-
                            gebene Stellung PØSY

STEP :          PUSH H      Register H&L in STACK retten
                ADD  D      addiere D zum Akku
                ANI  03H    letzten 2 Bit ausblenden
                MØV  D,A    neue Stellung des Schrittmotors ØW in D

                MØV  A,B
                ADD  E
                ANI  03H
                MØV  E,A    neue Stellung des Schrittmotors NS in E

ØUTP :          MØV  A,D    PØSX in Akku laden
                CALL TRLT   Aufruf des Unterprogramms TRLT

TRLT übersetzt die Stellung PØSX in das entsprechende Strommuster
für die Schrittmotoren. Das Ergebnis steht in A und ist 4 Bit lang.

                MØV  H,A    nach H zwischenspeichern
                MØV  A,E
                CALL TRLT   PØSY wird in entsprechendes Stromfluß-
                            muster übersetzt
                RLC
                RLC         Inhalt von A wird 4 mal links geschoben(pack)
                RLC
                RLC
                ØRA  H      Die Stromflußmuster von PØSX und PØSY
                            werden in einem 8-Bit-Wort zusammengefaßt.
                ØUT  OECH   Die Stromflußmuster beider Motoren werden
                            über die EIN/AUSGABE, PØRT A ausgegeben.

Beide Schrittmotoren werden gleichzeitig angesteuert
                MØV  A,C    C enthält Grundzeit


ZEIT :          CALL TIME   Unterprogramm TIME verzögert um die
                            Grundzeit

                MVI  H,20H  Zähler H auf 20 setzen
                MVI  L,03H  Zähler L auf 3 setzen

INB :           MVI  B,00H  Register B zurücksetzen, bedeutet
                            Auswahl der ØW-Koordinate
                CALL INPT   Unterprogramm INPT aufrufen

Der Impulsgeber auf der ØW-Koordinate wird eingelesen

                CMP  D      Vergleich mit der Sollstellung PØSX
                JNZ  WT1    Sollstellung erreicht

                INR  B      Register B auf 01 setzen, bedeutet
                            Auswahl der NS-Koordinate
                CALL INPT   Unterprogramm INPT aufrufen
                CMP  E      Vergleich mit der Sollstellung PØSY

                JZ   ØN1    Beide Antriebe haben Sollstellung erreicht

WT1 :           DCR  H      Stellung nicht erreicht, H herunterzählen
                JZ   ERR    Fehlerbedingung. Blockierung

                MVI  A,01H  Verzögerung von ca. 3 m s setzen
                CALL TIME   über Unterprogramm TIME
                JMP  INB    Sprung zum Schleifenanfang

ØN1 :           MVI  H,08H  Zähler H neu setzen
                MVI  B,00H  B = 00 setzen
                CALL INPT   Einlesen ØW-Koordinate
                CMP  D      Vergleich mit Sollstellung PØSX
                JNZ  WT2    ungleich
                INR  B      B = 01 setzen
                CALL INPT   Einlesen NS-Koordinate
                CMP  E
                JZ   ØN2    Beide Impulsgeber stehen auf Sollstellung

WT2 :           MVI  L,03H  Zähler L setzen
                DCR  H      Zähler H herunterzählen
                JZ   ERR    Schleife 8 x durchlaufen: Fehler
                MVI  A,01H
                CALL TIME   3 ms Verzögerung
                JMP  ØN1+1  Rücksprung zum Schleifenanfang

ØN2 :           MVI  A,01H  3 ms Verzögerung
                CALL TIME

                DCR  L      Zähler L um 1 vermindern
                JNZ  ØN1+1  Schleife muß 3 mal durchlaufen werden.

FEKT :          MVI  A,00H  A-Register rücksetzen
                PØP  H      H&L aus dem STACK rückladen
                RET         Rücksprung ins Hauptprogramm

Marke ERR befindet sich außerhalb des Unterprogramms an zentraler Stelle.
```

(Fortsetzung des Programms nächste Seite)

```
                    Unterprogramm TRLT

TRLT :      PUSH B     Register B&C retten in STACK
            PUSH H     Register H&L retten in STACK
            LXI  H,TAB Register H&L mit Adresse TAB laden
            MVI  B,00  B = 00 setzen
            MOV  C,A   A enthält PØSX bzw. PØSY (00, 01, 02, 03)
            DAD        Addiert PØSX zu Adresse TAB. Ergibt
                       neue Adresse TAB + PØSX in H&L
            MØV  A,M   Durch H&L adressierte Speicherstelle
                       ins A-Register holen (= Ergebnis)
            PØP  H     Rückspeichern H&L
            PØP  B     Rückspeichern B&C

            RET        Rücksprung in Hauptprogramm

TAB :       DB   05H )
            DB   06H ) Schaltmuster für
            DB   0AH ) Schrittmotortreiber
            DB   09H )


                    Unterprogramm TIME

TIME :      PUSH B     Register B&C retten in STACK
 ØLP :      MVI  C,85H Schleifengrenze setzen
 JLP :      DCR  C
            JNZ  ILP   Innere Schleife wird 85H x durchlaufen
            DCR  A
            JNZ  ØLP   Äußere Schleife wird so oft durchlaufen,
                       wie Register A zu Beginn anzeigt
            PØP  B     Rückspeichern
            RET        Rücksprung ins Hauptprogramm


                    Unterprogramm INPT

INPT :      PUSH B     B&C in Stack retten

            IN   0F5H  EIN/AUSGABE-Einheit Port B einlesen,
                       Ergebnis in A, zeigt Stellung beider
                       Impulsgeber
            RRC
            RRC
            RRC        4 x nach rechts schieben
            RRC
            DCR  B     B enthält 00 = ØW-Achse oder 01 = NS-Achse
            JM   XIMP  ØW-Achse gewählt
YIMP :      RRC        NS-Achse gewählt
            RRC        2 x nach rechts schieben


XIMP :      ANI  03H   Maskieren, letzten 2 Bit ausblenden
            CPI  03H
            JZ   LL
            CPI  02H   Umkodieren der Stellungen LL in LO
            JZ   LO    und der Stellung LO in LL
            JMP  ØN1   Stellungen 00 und 0L müssen nicht
                       umkodiert werden
LL   :      DCR  A
            JMP  ØNI
LØ   :      INR  A

ØNI  :      PØP  B     Ergebnis in A, B&C rückspeichern

            RET        Rücksprung ins Hauptprogramm
```

halten kann auch an einem federnden Anschlag auftreten. Als Kriterium für das Erreichen der Sollstellung wird deshalb gefordert, daß diese eine gewisse Zeit lang gehalten wird. In Bild 6 sind die Schaltpunkte des Impulsgebers mit eingetragen. Nach T_1 wird der Sollwert das erste Mal eingelesen und damit die 1. Schleife des Programms verlassen. Durch die Massenträgheitsmomente wird der Sollwert bei T_2 wieder verlassen.

Dies wiederholt sich bis T_N, wenn die Schwingung soweit abgeklungen ist, daß sie ganz innerhalb der Schaltgrenzen verläuft. Im Normalfall ist die Dämpfung durch das Getriebe allerdings so groß, daß selten mehr als ein Überschwinger auftritt.

Die 2. *Schleife* im Programm mißt, wie lange der Sollwert gehalten wird. Die Wartezeit sollte etwa $^3/_4$ der mechanischen Schwingungsdauer betragen, da innterhalb dieses Zeitraums mindestens ein Maximum liegt. Nach Herunterzählen des Zählers L auf Null ist diese Zeit verstrichen. Wird zwischendurch die Sollstellung verlassen, wird der Zähler L in *Schleife* 3 neu gesetzt und der Meßvorgang beginnt von neuem. Nach 8 Versuchen muß ein grundsätzlicher Fehler angenommen werden und das Programm wird abgebrochen. Das Programmlisting, *Bild 8*, zeigt die einzelnen Verarbeitungsschritte noch genauer. Sich wiederholende Teile, wie TIME, INPT und TRLT, sind als Unterprogramme organisiert. Alle Werteübergaben erfolgen über die Register A–L. Die simultane Ansteuerung beider Motoren ist notwendig, da sonst durch die einfache Wartezeitstruktur eine gegenseitige Beeinflussung der Motoren erfolgt. Das Schrittmotor-Timing-Problem könnte auch über eine Interruptsteuerung

Tabelle der technischen Daten des Kartenanzeigeräts

Abmessungen	330 x 610 x 40 mm³
Gewicht	6,2 kg
Temperaturbereich	0...+50 °C
Schüttelfestigkeit	5 g (20...100 Hz)
Feuchtigkeitsschutz	schwallwasserdicht
Anzeigefeld	310 x 460 mm²
Schrittweite	0,2 mm
Anzeigefehler, absolut	± 0,2 mm
Anzeigereproduzierbarkeit	±0,1 mm
Handverstellmöglichkeit	2 Geschwindigkeiten: 10 Schritte/s und 100 Schritte/s
Stromversorgung	24 V = (20...30 V)
Leistungsaufnahme ohne Kartenfeldbeleuchtung	ca. 43 W
Leistungsaufnahme mit Kartenfeldbeleuchtung	ca. 67 W
Maßstäbe	1:50 000, 1:100 000, 1:250 000
Kartenblattorientierung	umschaltbar
Kartenfeldbeleuchtung	einstellbar in 4 Stufen
Lichtkreuzhelligkeit	einstellbar in 3 Stufen
Elektronik	gesteuert mit Mikroprozessor 8080A
BITE-Funktion	Blinkanzeige im Bedienfeld, 2 Frequenzen
Anschlußstecker	PT 02 C – 16 – 26 S

gelöst werden, erfordert dann aber höheren Schaltungsaufwand.

Eine Realisierung dieser komplizierten Abfragen in konventioneller Technik hätte einen Aufwand von mindestens 30 integrierten Schaltungen erfordert, unter denen die Zeitglieder als besonders störanfällig anzusehen sind. *Hier zeigt sich die Leistungsfähigkeit des Mikroprozessors, der neben den Rechenaufgaben die Steuerungsaufgaben miterledigt.* Die Verarbeitungsgeschwindigkeit reicht aus, ohne Schwierigkeiten zu Schrittfrequenzen von 100 Hz bei voller Schrittüberwachung zu kommen. Bei Störungen, veränderlicher Spindelreibung usw. paßt sich die Schrittfrequenz dynamisch den Verhältnissen an, so daß immer ein maximales Antriebsmoment zur Verfügung steht.

5 Technische Daten

Die *Tabelle* gibt einen Überblick über die technischen Daten des Kartengeräts. Der Prototyp wurde in etwa elf Monaten entwickelt. Die Erstellung des etwa 1,2 K-Worte umfassenden Programms erforderte davon drei Monate. Zur Zeit befindet sich das Gerät in der Erprobung.

Der Autor ist Wiss. Mitarbeiter im Institut für Flugnavigation der Universität Stuttgart. Leitung: *Prof. Dr.-Ing. Dr.-Ing. E.h. K. Ramsayer.*

Literatur

[1] *McGrath, J. J.*: Geographic Orientation in Air Operations. Proceedings of a Symposium in Washington, D.C. November 18-20, 1969, under the auspices of the Working Group for Joint Army-Navy Aircraft Instrumentation Research.

[2] *Ramsayer K.* und *Wildermuth, E.*: Die automatische Koppelkarte AK4T, ein neuer Navigationsrechner zur Integration bordeigener und bodengestützter Navigationshilfen. Ortung und Navigation, 1967, Heft 1.

[3] Intel Data Catalog 1976.

[4] *Colbourne, E. Det al.*: Reliability of MOS-LSI Circuits. Proceedings of the IEEE, Bd. 62, H. 2, Febr. 1974.

[5] Semiconductor Reliability within the U.S. Department of Defense. Proceedings of the IEEE, Bd. 62, H. 2, Febr. 1974, S. 163...184.

[6] *Wakerly, J. F.*: Microcomputer reliability improvement using triple-modular redundancy. Proceedings of the IEEE, Bd. 64, Juni 76, S. 889...895.

[7] *Bazovsky, I.*: Reliability Theory and Practice. Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1961 (Buch).

[8] Technische Zuverlässigkeit. Herausgegeben von MBB. Springer Verlag, 1971 (Buch).

[9] *Jansen, D.*: Digital arbeitendes Positions-Nachlaufsystem. ELEKTRONIK 1976, H. 7, S. 47...52.

Dipl.-Ing. Helmut Klie,
Joachim Dziubiel

Analog-Schnittstellen arbeiten mit Mikroprozessoren zusammen

In der Praxis tritt häufig die Problematik auf, Meßwerte nach deren Erfassung zusätzlich auszuwerten, aufzubereiten und eventuell über längere Zeit abzuspeichern. In dieser Situation, und zwar besonders dann, wenn die Datenaufbereitung aufwendige Umrechnungen erfordert, bietet sich – wie in diesem Fall – der Einsatz eines intelligenten Datenerfassungssystems an, das selbstverständlich mit einem Analog-Interface zusammenarbeiten muß. Ziel dieses Artikels ist es nun, dieses wichtige Bindeglied einer mikroprozessorgesteuerten Meßkette, das A/D-Umsetzersystem, in der Anwendung vorzustellen. Den Autoren geht es darum, den Leser mit der System-Hardware vertraut zu machen und zugleich einige Software-Hinweise zu geben.

1 Hardware

1.1 Systemkomponenten

Die einzelnen Funktionsblöcke des mikroprozessorgesteuerten Datenerfassungssystems sind in *Bild 1* dargestellt. Die physikalischen Meßgrößen wie Temperatur, Druck, Dehnungswerte usw. werden von den Meßwertaufnehmern in analoge elektrische Signale umgeformt. Diese Signale liegen an den Analogeingängen des A/D-Umsetzersystems an. Ein über einen Adreßzähler gesteuerter Multiplexer führt die Analogsignale auf den Eingang eines Instrumentationsverstärkers, der mit seinem Ausgang über ein Sample-Hold-Glied an den A/D-Umsetzer gekoppelt ist. Multiplexer, Verstärker sowie Sample-Hold-Schaltung und A/D-Umsetzer stellen das Datenerfassungssystem dar. Es ist über den System-Bus direkt an den Mikroprozessor gekoppelt. Als Peripheriegeräte stehen ein Fernschreiber sowie ein Magnetplattensystem (Floppy Disk) zur Verfügung.

1.2 Systemaufbau

Bild 2 zeigt den Meßplatz. Als Mikroprozessor dient das MDS 800-Entwicklungssystem der Firma Intel. Die Grundausbaustufe besteht aus vier Basismoduln, der Busverdrahtung, einer Stromversorgung sowie einem Bedienungsfeld. Die vier Basismoduln umfassen die Zentraleinheit mit dem 8080-Mikroprozessorchip, eine 16-KByte-RAM-Speicherplatine, den Bedienungsfeldmodul und den Monitormodul. Hinzu kommen zwei Steuermoduln als Interface für die Floppy Disk. Die Zentraleinheit enthält den 8-bit-Mikroprozessor 8080, der mit einem 8-bit-Zweiweg-Datenbus zusammenarbeitet [1]. Die Peripherie besteht aus einem Teletype-Fernschreiber als Konsole und einer Floppy Disk mit zwei Laufwerken zu je 1/4 Million Byte Externspeicherplatz.

Als A/D-Umsetzersystem findet das von der Firma Burr-Brown entwickelte Datenverarbeitungssystem MP 8417 Anwendung [6]. Dieses System besteht aus der Platine MP 8416 und dem zugehörigen Flachkabelsatz zum Anschluß der analogen Meßkanäle. Auf die Funktion sowie die Eigenschaften des analogen Datenerfassungssystems wird im folgenden ausführlich eingegangen.

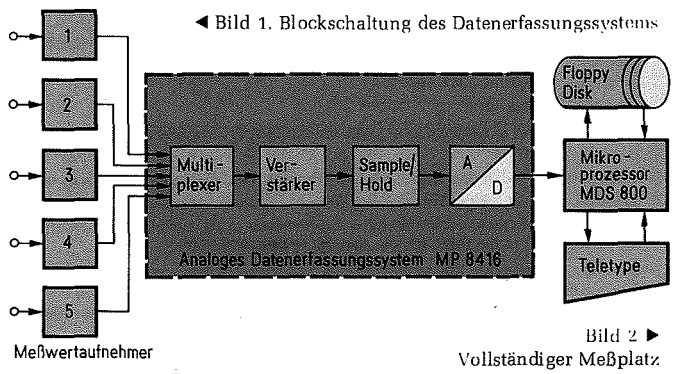
1.3 Analoges Datenerfassungssystem

Das A/D-Umsetzersystem ist elektrisch und mechanisch mit dem Mikroprozessor MDS 800 kompatibel. Die Platine ist vom Hersteller als 16-Kanal-Version massebezogen mit einem Eingangsmeßbereich von ±10 V geschaltet. Die analogen Eingangsdaten werden in 12-bit-Digitalsignale umgesetzt. Durch externe Beschaltung bzw. Verändern von Brükken kann die Grundkonzeption der Platine modifiziert werden [2].

Der Verstärkungsfaktor ist einstellbar in einem Bereich von 1...1000 durch externe entsprechende Widerstandsbeschaltung. Dabei ist allerdings zu beachten, daß bei einem Faktor von 1000 der Systemfehler bezogen auf vollen Skalenbereich von ±0,025 % auf ±0,1 % steigt. Die Meßfolgerate verringert sich dabei von 30 kHz auf 10 kHz. Die Einstellzeit steigt dabei entsprechend der *Tabelle* [3] an.

Das A/D-Umsetzersystem gestattet es dem Benutzer weiterhin, durch Ergänzung der entsprechenden Shunt-Widerstände eine einfache Eingabe von Stromsignalen von 4...20 mA und 10...5 mA vorzunehmen. Darüber hinaus gibt es die Möglichkeit, durch Abänderung einiger Brücken die 16 massebezogenen Eingänge als 8 Differenzkanäle zu betreiben. Diese Eigenschaft, gepaart mit der Möglichkeit der

◀ Bild 1. Blockschaltung des Datenerfassungssystems



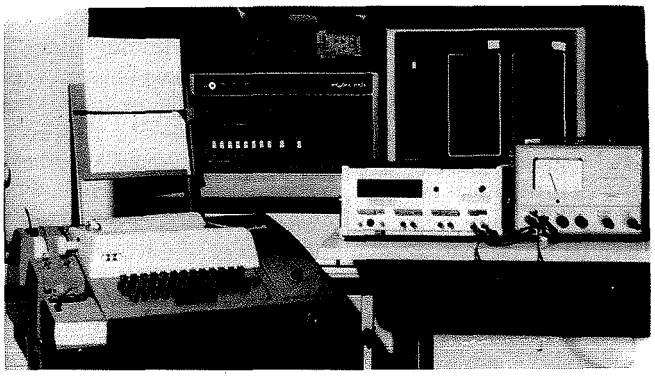


Bild 2 ▶
Vollständiger Meßplatz

Tabelle der wichtigen Betriebsdaten des A/D-Umsetzers

System-Verstärkg.	Typ. System-fehler	Durchsatz (Kanäle/s)		Verzögerungszeit (µs)	
V/V	% vom Vollaus.	Normal	Überlappend	Normal	Überlappend
1	±0,025 %	30 K	32 K	9	31
10	±0,035 %	25 K	32 K	18	31
100	±0,08 %	20 K	32 K	25	31
1000	±0,1 %	10 K	14 K	70	70

Wahl des hohen Verstärkungsfaktors, ist sehr wertvoll für die Verarbeitung von Kleinsignalen, wie sie z. B. bei Verwendung von Thermoelementen auftreten.

Das Kernstück der A/D-Umsetzerplatine, das System SDM 853, ist in einem magnetisch abgeschirmten Gehäuse eingebaut. Es besteht (*Bild 3*) aus einem analogen Multiplexer, einem Instrumentationsverstärker, einer Sample-and-Hold-Schaltung, dem 12-bit-A/D-Umsetzer sowie der nötigen Adreß- und Steuerlogik. Der CMOS-Multiplexer kann Eingangsspannungsbereiche von ±10 V bis herab zu ±10 mV Vollausschlag verarbeiten. Die einzelnen Kanäle werden durch einen 4-bit-Binär-Adreßzähler angesteuert. Der Null- und Endpunkt des A/D-Umsetzers lassen sich über Offsetpotentiometer von außen abgleichen. Zu diesem Zweck stellt der Hersteller ein Kalibrierprogramm zur Verfügung.

Eine Vergrößerung der Umsetzungsrate, die sich aus der Einstellzeit von Multiplexer und Verstärker, der Übernahmezeit des Analogspeichers sowie der Umsetzzeit des A/D-Umsetzers zusammensetzt, kann durch die sog. „überlappende Betriebsart" zum Teil kompensiert werden. Der Multiplexer wird dazu bereits auf den nächsten Kanal umgeschaltet, während der Analogwert eines Kanals vom A/D-Umsetzer digitalisiert wird.

2 Software

2.1 Verwendete Mikroprozessorsoftware

Das Mikroprozessorsystem MDS 800 in Verbindung mit dem Floppy-Disk-Speicher bietet bei der Grundausstattung des Rechners (16 KByte) bereits eine einigermaßen komfortable Software. Die Programmiersprache ist Assembler [4]. Der für dieses Gerät entwickelte PLM-Compiler ist erst bei Vollausbaustufe des Rechners (64 KByte) einsetzbar. Grundsätzlich sind alle Systemprogramme von Magnetplatten abrufbar – auf zeitaufwendigen Lochstreifenbetrieb über die Teletype-Konsole kann verzichtet werden.

Für die Aufbereitung des Programmtextes steht ein Texteditor mit recht komfortablen Befehlen zur Verfügung. Nach der Fertigstellung dieses Quellenprogramms erfolgt zunächst ein Zurückladen auf die Platte. Danach beginnt das Assemblieren, d. h. das Umsetzen des Quellenprogramms in

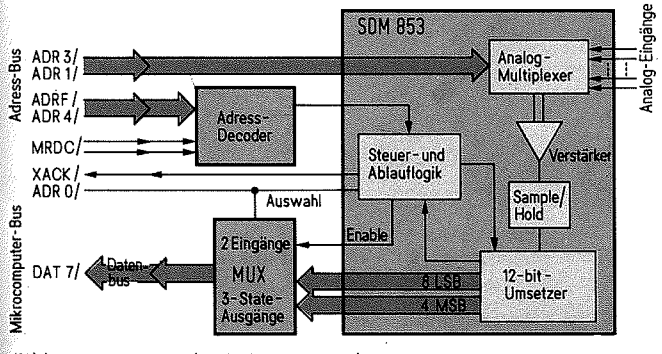


Bild 3. Baugruppen der A/D-Umsetzerplatine

die Maschinensprache. Dies geschieht mittels mehrerer Durchläufe, wobei gleichzeitig ein Listing des Quellenprogramms erstellt wird, aus dem die Speicherplatzzuordnung und der Maschinencode zu erkennen sind. Das Listing ist für das Aufsuchen von Programmfehlern erforderlich.

Nach der Assemblierung wird der vom Assembler erzeugte sedezimale (hexadezimale) Maschinencode durch ein Kommando in den Dualcode umgesetzt. Das Programm ist nunmehr ablauffähig und benötigt 1/3 weniger Programmspeicherplatz. Der Mikroprozessor arbeitet mit dem im Speicher residenten Monitorbetriebssystem. Durch den Einsatz der Floppy Disk steht eine zusätzliche Software als Programmschnittstelle zwischen Magnetplattenspeicher und MDS 800-System zur Verfügung. Dabei handelt es sich um den „Intel Systems Implementation Supervisor", kurz ISIS genannt.

Das Programm kann auf zwei Arten gestartet werden. Im ersten Fall geschieht dies durch einfachen Aufruf des Programmnamens auf der ISIS-Ebene. Im zweiten Fall erfolgt der Programmablauf im Debug-Betrieb. Der Vorteil dieser Methode ist, daß nach Ausführung des ISIS-Debug-Befehls die Kontrolle an den Systemmonitor übergeben wird, der es gestattet, Haltepunkte zu setzen und Register- und Speicherinhalte ausdrucken zu lassen. Auf diese Weise kann das Programm Bereich für Bereich ausgetestet werden [5].

2.2 Programmäßige Behandlung des A/D-Umsetzers

Die Eingangskanäle der A/D-Umsetzerplatine werden wie normale Speicherplätze adressiert. Da die Wortlänge der zu verarbeitenden Information 12 bit beträgt, benötigt jeder analoge Datenkanal zwei Speicherplätze. Der erste Kanal ist vom Hersteller auf die Adressen F720 H und F721 H fest verdrahtet. Diese Adressierung kann jederzeit vom Benutzer beliebig abgeändert werden [2].

Der eigentliche Datenerfassungsvorgang erfolgt durch Ausführung eines LHLD-Befehls gefolgt von der gewünschten Kanaladresse. Dieser Befehl leitet die A/D-Umsetzung ein und transportiert nach Abschluß der Umsetzung die in 2 Byte stehende Information in die H- und L-Register der CPU. Von hier aus kann die Information bequem weiterverarbeitet werden.

3 Anwenderprogramm – Programmerläuterung

Das erste Testprogramm dient zunächst einmal nur dazu, die nach der Erfassung binär vorliegenden Spannungswerte benutzerfreundlich auf der Konsole darzustellen. Der dazu erforderliche Aufwand sollte nicht unterschätzt werden. Das Programm, dessen Ablauf im Gesamtflußdiagramm (*Bild 4*) dargestellt ist, arbeitet im Dialogbetrieb. Der Benutzer kann auf diese Weise den gewünschten Spannungsbereich auch softwaremäßig, lediglich durch Eingabe einer von vier Ziffern, auswählen. Fehlerhafte Eingaben werden dabei erkannt und angezeigt. Nach diesen Vorbereitungsroutinen erfolgt die eigentliche Meßdatenerfassung. Die binären Werte werden, nachdem sie per Programm auf das richtige Vorzeichen überprüft wurden, nach einem Multibyte-Additionsverfahren umgerechnet (*Bild 5*). Das Verfahren basiert auf einem bitweisen Austesten des Binärwertes, wobei man davon auszugehen hat, daß 1 LSB identisch ist mit dem Wert des jeweiligen gewählten Spannungsbereiches dividiert durch $2^{12} = 4096$ bei 12 bit Auflösung.

Die Ausgabe der Meßwerte – angedeutet im Gesamtflußplan (Bild 4) – erfolgt in gut übersichtlicher Darstellung mit Vorzeichenangabe und Dezimalpunkt als ein Meßwert pro Zeile. Nach dem Ausdruck einer vom Benutzer vorzuwählenden Anzahl von Meßwerten stellt das Programm durch

Anfrage fest, ob gegebenenfalls eine Fortsetzung bzw. Programmende erwünscht ist. Bei Ende erfolgt die Übergabe der Steuerung zurück an das ISIS-Programm. Das vorliegende Programm umfaßt 340 Instruktionen; der erforderliche Programmspeicherplatz beträgt 850 Bytes in sedezimalem Maschinencode.

Bei diesem ersten Testprogramm wurde bewußt auf eine spezielle Problemstellung verzichtet, um zunächst einmal nur die für den Einsatz des A/D-Umsetzers grundlegende Software vorzustellen. Weitere interessante Applikationen mit Zeitbezug, speziellen Datenaufbereitungsproblemen, Datenspeicherung sowie Plotteransteuerung usw. werden z. Z. noch ausgearbeitet.

Das ausführliche Listing des ersten Testprogramms steht Interessenten auf Anfrage zur Verfügung (siehe Autorenadresse). Ein Abdruck ist wegen der Länge nicht möglich.

Literatur

[1] Hardware Reference Manual. Systembeschreibung Nr. B 98–132B der Firma Intel.

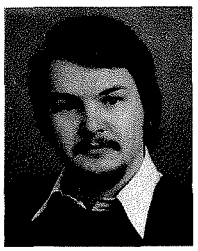
[2] Intel Microcomputer Analog I/O Systems. Datenblatt der Firma Burr-Brown.

[3] User's Manual SDM 853 Data Acquisition System. Systembeschreibung PDS-358 der Firma Burr-Brown.

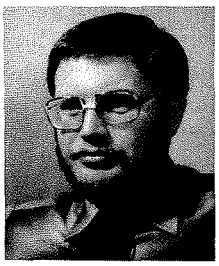
[4] *Pelka, H.*: Was ist ein Mikroprozessor? Franzis-Verlag, München, S. 76...79.

[5] Diskette Operating System, Microcomputer Development System. Handbuch Nr. 98-206B der Firma Intel.

[6] *Siebert, H.-P.*: Modul erfaßt und digitalisiert 16 Analogsignale. ELEKTRONIK 1974, H. 3, S. 85...89.



Dipl.-Ing. Helmut Klie studierte Elektrotechnik an der TU Hannover; sein Diplom erhielt er im Frühjahr 1975. Seit Herbst 1975 Assistent an der Abt. für Biomedizinische Technik sp. Krankenhaustechnik. Arbeitsgebiet: rechnergesteuerte Meßdatenerfassung und Prozeßsteuerung.
Telefon (priv.): 05 11/57 34 43
Hobbys: Segeln
ELEKTRONIK-Leser seit 1970



Joachim Dziubiel wurde im Jahr 1943 geboren. Er leitet derzeit die Programmier-Abteilung in der Firma Ernst Siegling, Hannover.
Telefon (priv.): 05 11/40 94 22
Hobbys: Leichtathletik
ELEKTRONIK-Leser seit 1973

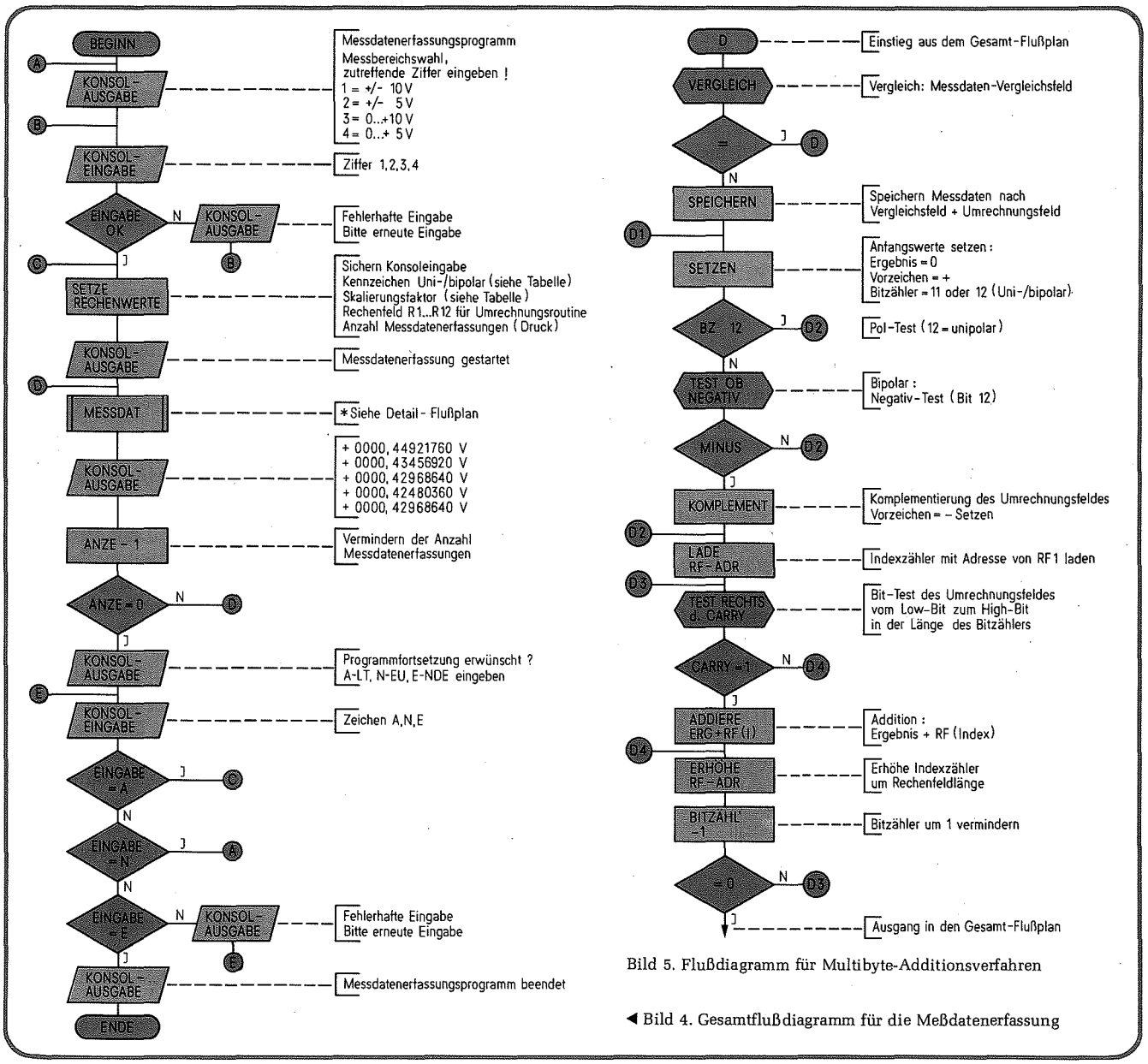


Bild 5. Flußdiagramm für Multibyte-Additionsverfahren

◀ Bild 4. Gesamtflußdiagramm für die Meßdatenerfassung

BAUELEMENTE aktuell

KURZ-
INFORMATIONEN
VALVO

Neue Interfaceschaltungen zum Valvo Signetics Mikroprozessor 2650

Die neuen Interfaceschaltungen zum Valvo Signetics Mikroprozessor-System 2650 haben nur eine Versorgungsspannung von 5 V, sind TTL-kompatibel und arbeiten in einem Temperaturbereich von 0 °C bis 70 °C.

MPCC 2652
Die Schaltung 2652 (Multi Protocol Communications Controller) ist eine programmierbare synchrone Empfänger-Sender-Schaltung.
Die vom Mikroprozessor kommenden parallelen Daten werden in serielle Daten umgewandelt und umgekehrt. Sender und Empfänger werden getrennt gesteuert.

PCI 2651
Die Schaltung 2651 (Programmable Communication Interface) ist eine universelle Steuerschaltung für die synchrone und asynchrone Verarbeitung von Daten. Sie wird direkt vom Mikroprozessor-System 2650 gesteuert und kann den seriellen Datenverkehr im Voll- oder Halb-Duplex abwickeln.

Informationen und Steuersignale werden vom D-Bus des Mikroprozessors 2650 übernommen und in internen Registern gespeichert. Eingehende serielle Information wird im Receiver-Buffer zwischengespeichert und in eine 8 bit-Information umgewandelt. Entsprechend empfängt der Transmitter-Buffer eine 8 bit-Information und wandelt diese in eine serielle bit-Folge um.

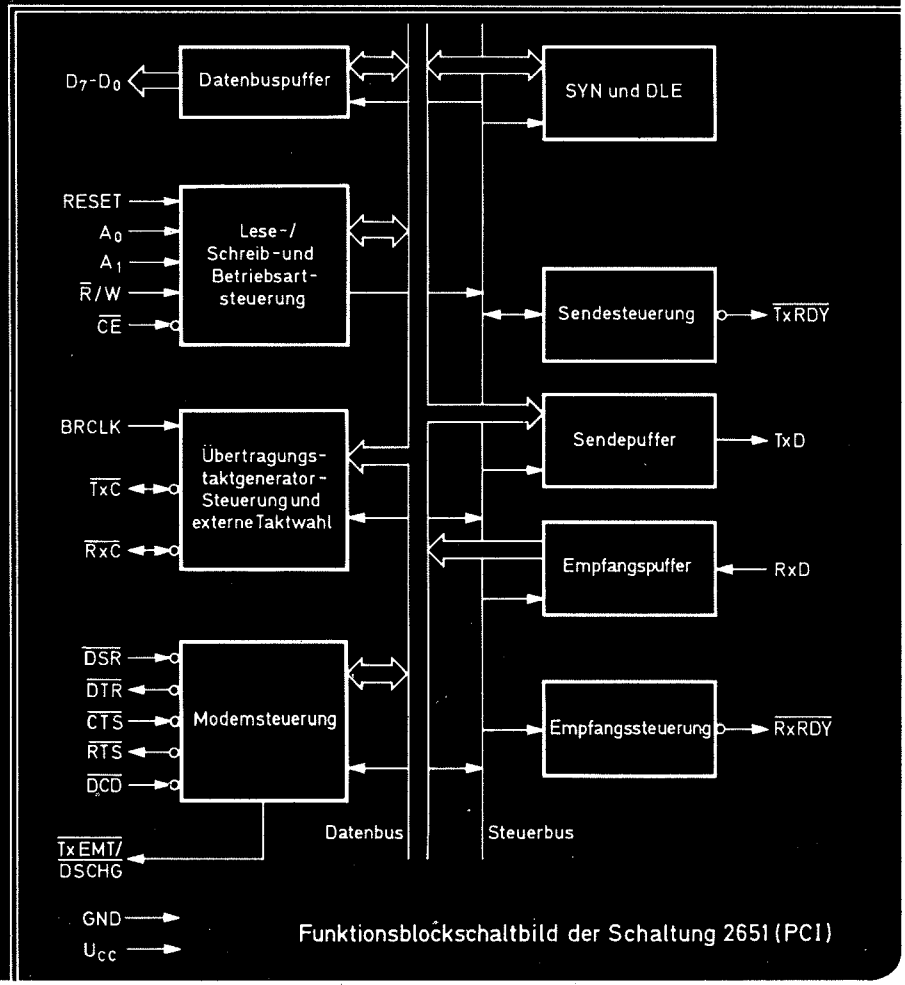
PPI 2655
Die Schaltung 2655 (Programmable Peripheral Interface) ist eine programmierbare Interfaceschaltung für das Mikroprozessor-System 2650. Sie wird mit dem System über einen bidirektionalen 8 bit-Datenbus verknüpft. Die drei Ausgänge mit insgesamt 24 Anschlüssen können individuell programmiert werden.

Die Schaltung 2655 kann auch als Timer oder Ereigniszähler bis zu 3 MHz eingesetzt werden.

Es sind fünf Betriebsarten möglich: Statische, getaktete, bidirektionale sowie serielle Ein-/Ausgabe und serielle Ein-/Ausgabe mit Timer.

SMI 2656
Die Schaltung 2656 (System Memory Interface) dient dazu, ein einfaches und kostengünstiges Mikroprozessor-System aufzubauen. Gemeinsam mit dem Mikroprozessor 2650 läßt sich ein 2-Chip-System realisieren, da die Schaltung 2656 alle Einheiten wie ROM, RAM, Takt- und Ein-/Ausgabe-Schaltung enthält.

In komplexeren Systemen arbeitet die Schaltung 2656 als zentrale Steuereinheit und liefert hierbei die ROM- und RAM-Anfangsadressen, den Grundtakt für den Mikroprozessor sowie vollständig decodierte, zeitangepaßte Chip-Auswahlsignale, so daß zur Auswahl externer Speicher- oder Ein-/Ausgabe-Schaltungen keinerlei zusätzliche Logik erforderlich ist.



Funktionsblockschaltbild der Schaltung 2651 (PCI)

MCS MICRONIC COMPUTER SYSTEME GMBH

Entwicklung – Fertigung – Beratung – Hardware – Software

Unser neues MC-Konzept

ALPHA 1

ein anschlußfertiger Mikrocomputer

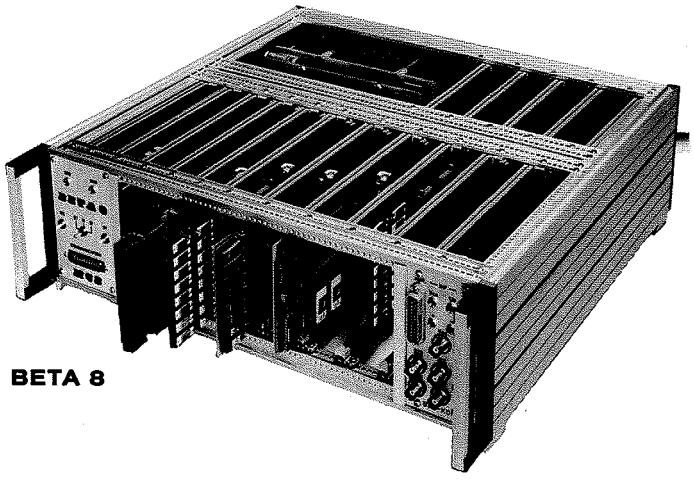
- umfangreiche eigene deutsche Dokumentation
- 2 K Betriebssystem **M O N A** (Monitor **A L P H A**) für 6502 von MOS, ROCKWELL, SYNERTEK
- 1,25 K RAM
- 16 freie I/O-Ports auf Stecker für Ihre Anwendung
- Netzteil eingebaut
- voll getestet
- BUS-kompatibel zu **B E T A 8**
- 8-stellige Siebensegment-Anzeige
- 27 Tasten / 6 Kontrollanzeigen (LED)
- Dateninterface für 2 Bandgeräte (auch Recorder)
- automatische Start-Stop-Steuerung der Bandgeräte
- V 24/RS 232-Schnittstelle } bis 4800 BAUD
- 20 mA Stromschnittstelle }

ALPHA 1

Diese Programmierhilfen bietet M O N A:

- Single-Step
- Slow-Step – zum Testen im Zeitlupentempo
- Disassembler!
- lesen und schreiben von Magnetbändern
- lesen und stanzen von Lochstreifen (8 Kanal ASCII)
- Terminal-Betrieb über ASCII bis 4800 BAUD
- Direkt-Anwahl von Accumulator, Statusregister und Programmzähler
- Vor- und Rückwärtslesen von Programmen
- Betrieb über eigene Tastatur und Anzeige
- alle Tasten und Anzeigen sind auch frei programmierbar
- **M O N A**-Routinen können von Anwender-Programmen mitbenutzt werden
- komplettes **M O N A**-Listing wird mitgeliefert.

A L P H A 1 ist voll integrierbar in **B E T A 8**



BETA 8

BETA 8

das Modulsystem und seine Merkmale:

- EURO-Karte (100 x 160 mm)
- 19-Zoll-System
- Processorunabhängige **B E T A 8** BUS-Struktur
- 8 Interruptprioritätsebenen
- DMA- und multiprocessfähiger BUS
- bis 16 Karten indirekt steckbar (64-pol. VG)
- BUS-Zentralplatine
- Netzteil 5 V/5,5 A – ± 12 V/0,8 A
- individuelle Frontplattengestaltung möglich (5 cm Bautiefe frei!)

und diese EURO-Karten sind schon verfügbar:

M 622 6502-CPU incl. 1 K RAM
M 620 2 K x 8 RAM (2112)
M 624 8 K x 8 RAM 2114 oder PROM 6353-1 o.ä.
M 623 4 (8) K x 8 EPROM-, PROM-, ROM-, RAM-Speicherkarte u. a. 2708, 2716 . . . (4 Bausteine, **jeder** voll dekodiert, beliebig kombinierbar)
M 628 I/O-Karte 2 x 6532 mit 2 x 25-pol. DSub-Buchse – Steckverbindungen für Aufsatzkarte für individuelle Peripherie –
M 629 Aufsatzkarte für I/O-Karte mit Lochraster
M 630 Aufsatzkarte für M 628 mit Betriebssystem M O N A 2
M 621 Experimentierkarte
M 631 I/O-Karte 2 x 6520 mit 2 x 25-pol. DSub-Buchse
alle Karten gepuffert und durch DIL-Schalter voll dekodiert.

in Vorbereitung: CPU 8080, 6800, Z 80, SCAMP
Floppycontroller für BASF-Floppy
IEC-Schnittstelle
E-PROM-Programmierkarte

MCS – Produkte werden in Berlin entwickelt und hergestellt
MCS – ist Ihr unabhängiger Partner für Microprocessor-Hard- und Software
MCS – für weitere Informationen wenden Sie sich an:

MCS Tennstedter Str. 18, 1000 Berlin 46,
Telefon (0 30) 7 11 30 11 (7 11 20 55), Telex 0184 187 voco d

Mikrocomputer verarbeiten parallel anliegende Daten wesentlich schneller, als sie von asynchron (seriell) arbeitenden Peripheriegeräten angeliefert und entgegengenommen werden. Um den Mikroprozessor nicht an diese langsame Arbeitsweise zu binden, läßt man die Serien/Parallel- und die Parallel/Serien-Wandlung vom Interface durchführen.

Karl Fronheiser

Asynchrone Mikrocomputer-Schnittstelle

1 Der ACIA-Baustein

Bei der Mikroprozessor-Familie M 6800 (Motorola) übernimmt diese Aufgabe der ACIA-Baustein (*Asynchronous Communication Interface Adapter*). Er ist auf der einen Seite direkt kompatibel mit dem Bus des Mikroprozessors, während die andere Seite mit Peripheriegeräten kompatibel ist, die ein asynchrones Datenformat benützen. Ein asynchrones Datenformat besteht aus aneinandergereihten seriellen Bits, wobei den Datenbits ein Startbit vorangeht und ein oder zwei Stoppbits nachfolgen. Da kein zuvor mit den Daten synchronisierter Taktimpuls vorhanden ist, werden vom ACIA die Kennzeichen des asynchronen Datenformats benützt, um Bits und Zeichen zu synchronisieren. Der ACIA wandelt ein Zeichen, das von einem Peripheriegerät seriell empfangen wurde, zu einem parallelen Byte um, wobei Start-, Stopp- und Paritätsbit wegfallen. Desgleichen werden vom Mikroprozessor stammende parallele Bytes in serielle Form gebracht, mit Start-, Stopp- und wahlweisem Paritätsbit, die dem Zeichen mitgegeben werden. Der ACIA besteht aus Steuer- und Statusregister und je einem Sende- und Empfangsregister für Daten, ferner aus Puffern für den Datenbus, Schieberegistern für Senden und Empfangen und aus peripheren Steuereinrichtungen, wie es aus dem Blockdiagramm in *Bild 1* hervorgeht.

Der ACIA benötigt zwei Adressen zur Adressierung seiner vier Register für Steuerung, Status, Senden und Empfangen. Um innerhalb des ACIA ein Register auszuwählen, sind die geeigneten logischen Pegel der Eingänge für Chip-Auswahl (CS0, CS1, $\overline{CS2}$), Registerauswahl (RS) und Lese/Schreibsteuerung (R/W) nötig. Das am Mikroprozessor vorhandene Ausgangssignal R/W steuert das Einlesen von Daten in den Speicher oder das Ausschreiben von dort auf periphere Einheiten. Zusätzlich wählt dieses Signal das Lese- oder Schreibregister im ACIA. Eine Kombination der Eingänge für Chip-Auswahl und des Einganges für Registerauswahl kann dazu dienen, um den Aufwand der für jedes Peripheriegerät benötigten Adreß-Decodierlogik gering zu halten:

Die vier Booleschen Kombinationen der Adreßleitungen A 14 und A 15 wählen z. B. Speicherblöcke aus, wie sie aus *Tabelle 1* hervorgehen. Durch die Zuordnung dieser Blöcke zu RAM, ROM, Peripheriegeräten usw. wird ein Speicheradreßverzeichnis des Systems gebildet. In diesem Beispiel sind die Peripheriegeräte den Adressen zwischen 8000 und BFFF zugeteilt. Falls das Adreßbit A 15 zu $\overline{CS0}$ und das Adreßbit A 14 zu $\overline{CS2}$ zugeordnet ist, wird ein Peripheriegerät gewählt, wenn der logische Pegel von A 15 „hoch" und der von A 14 „tief" liegt. Zur Auswahl der vier internen Register benötigt der ACIA aufeinanderfolgende Adressen. Das Verknüpfen des CS1-Einganges mit einer der andern Adreßleitungen gestattet die Wahl von 13 verschiedenen Peripheriegeräten ohne zusätzliche Decodierlogik.

Tabelle 1. Mögliche Einteilung von Speicherbereichen

Adressiertes Element	Speicherblock	Adreßleitungen A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
ROM	C000...FFFF	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X
Peripherie	8000...BFFF	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM	4000...7FFF	0	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X
RAM	0000...3FFF	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X
ACIA 1	8400...8401	CS0	$\overline{CS2}$	0	0	0	CS1	0	0	0	0	0	0	0	0	0	RS
ACIA 2	8020...8021	CS0	$\overline{CS2}$	0	0	0	0	0	0	0	0	CS1	0	0	0	0	RS

X = 1 oder 0, CS0 = 1, $\overline{CS2}$ = 0, CS1 = 1, RS = 1 oder 0

digkeit bis maximal 600 bit/s. Ein typisches System besteht aus einem lokalen Modem und einem ähnlichen Modem beim Fernterminal.

Da der Baustein MC6860 keine automatische Selbstwahl besitzt, muß die telefonische Verbindung von Hand oder mit Hilfe externer automatischer Selbstwahl-Vorrichtungen hergestellt werden. Die Verbindungsaufnahme zwischen dem lokalen und dem entfernt aufgestellten Modem geht, nachdem die telefonische Verbindung hergestellt ist, folgendermaßen vor sich: Vom Programm gesteuert wird das lokale Modem durch den ACIA-Ausgang für die Sendeanforderung $\overline{\text{RTS}}$ (*Request-to-Send*), der mit dem Eingang $\overline{\text{DTR}}$ (*Data Terminal Ready*) am Modem verbunden ist, freigegeben. Sobald mit dem entfernten Modem durch Erkennen des Rufzeichens die telefonische Verbindung hergestellt ist, sendet dieses eine Trägerfrequenz zur Verbindungsaufnahme mit dem lokalen Modem aus. Aufgrund der erkannten Trägerfrequenz gibt das lokale Modem mit seinem Ausgang $\overline{\text{CTS}}$ (*Clear-to-Send*) die Übermittlung frei. Der $\overline{\text{CTS}}$-Ausgang des Modems ist direkt mit dem $\overline{\text{CTS}}$-Eingang des ACIA verbunden, und der logische Zustand dieses Einganges ist als Statusbit erhältlich. Somit kann das Zustandekommen der Verbindungsaufnahme mit Hilfe des Programms durch Lesen des Statusregisters überprüft werden. Ist die Verbindungsaufnahme erfolgreich, können Daten über die Telefonleitung gesendet oder empfangen werden. Da ein langsam arbeitendes Modem nur einen $\overline{\text{CTS}}$-Ausgang besitzt, wurden in diesem Beispiel die Eingänge $\overline{\text{CTS}}$ und $\overline{\text{DCD}}$ (*Data Carrier Detect*) des ACIA so miteinander verknüpft, daß eine Unterbrechung in der Übertragung entweder in der Sende- oder Empfangs-Subroutine erfaßt werden könnte.

Systeme mit Modems mittlerer Geschwindigkeit können unabhängig sowohl die $\overline{\text{CTS}}$- als auch die $\overline{\text{DCD}}$-Eingänge benützen. In einem Vierleitersystem zeigt der $\overline{\text{CTS}}$-Eingang den Status einer nur für den Empfang bestimmten Zweidrahtleitung an. Dazu kommt, daß bei einem Vierleitersystem der Verlust eines Leitungspaares den einseitigen Datenverkehr auf dem verbliebenen Leitungspaar nicht beeinträchtigt.

Weder in Systemen mit Modems kleiner noch in Systemen mit Modems mittlerer Übertragungsgeschwindigkeit sollte der RTS-Ausgang des ACIA H-Pegel annehmen, bevor das letzte Zeichen vom entfernt aufgestellten System vollständig empfangen wurde. Der ACIA besitzt jedoch keinen Ausgang für „Wort fertig", der anzeigen würde, daß das letzte geladene Zeichen vollständig gesendet wurde, so daß das Modem dann gesperrt werden könnte. Die Funktion „Wort fertig" kann erzeugt werden, indem ein Zeichen zum Schein in den ACIA geladen und dann das Statusregister nach der Bedingung „Senderegister frei" abgesucht wird, woraus zu ersehen ist, ob das Scheinzeichen ins Schieberegister übertragen wurde. Dies ist eine Angabe dafür, daß das halbe Stoppbit des letzten Zeichens vollständig übermittelt wurde. Wird also der RTS-Ausgang auf H-Pegel gesetzt, kann das lokale Modem unwirksam gemacht werden. Es ist jedoch sicherzustellen, daß das letzte Zeichen vom entfernten System vor dem Sperren der Modems gelesen werden konnte.

3 Durch Unterbrechungen gesteuerte Systeme

Wird ein Mikrocomputer-System mit immer mehr peripheren Geräten, die immer mehr Verarbeitungszeit benötigen, erweitert, dann wird es zunehmend schwieriger, jedes Peripheriegerät in der erhältlichen Zeit zu berücksichtigen. Ein Verfahren, um eine wirksame Zeitausnützung zu erreichen, ist die Verwendung eines durch Unterbrechungen gesteuerten Systems. In einem solchen System besitzt jeder Interface-Adapter der Mikroprozessor-Familie einen Unterbrechungseingang IRQ (*Interrupt Request*), der mit anderen Interface-Adaptern durch eine verdrahtete ODER-Verknüpfung verbunden ist, womit für eine gemeinsame Unterbrechungsleitung zum Mikroprozessor gesorgt ist. Jede Unterbrechung von einem der Adapter veranlaßt den Sprung zu einem Unterprogramm. Es veranlaßt den Mikroprozessor, das Statusregister jedes Interface-Adapters abzufragen. Der ACIA besitzt ein $\overline{\text{IRQ}}$-Statusbit, das sich im Statusregister auf Position b 7 befindet (Vorzeichenposition für Zahlen), so daß nur ein einziger Mikroprozessorbefehl (BMI = *Branch if Minus*) genügt, um zu entscheiden, ob der Sende- oder Empfangsteil des betreffenden ACIA die Unterbrechung verursachte. Wenn einmal festgestellt ist, daß die Unterbrechung von einem ACIA stammt, können die Statusbits TDRE (*Transmit Data Register Empty*) und RDRF (*Receive Data Register Full*) in eigenen Unterprogrammen geprüft werden, um den spezifischen Grund für die Unterbrechung zu ermitteln. Das Steuerregister kann vorprogrammiert werden, um eine Unterbrechung entweder vom Sende- oder Empfangsteil des ACIA, je nach dessen Verwendung, zu verhindern.

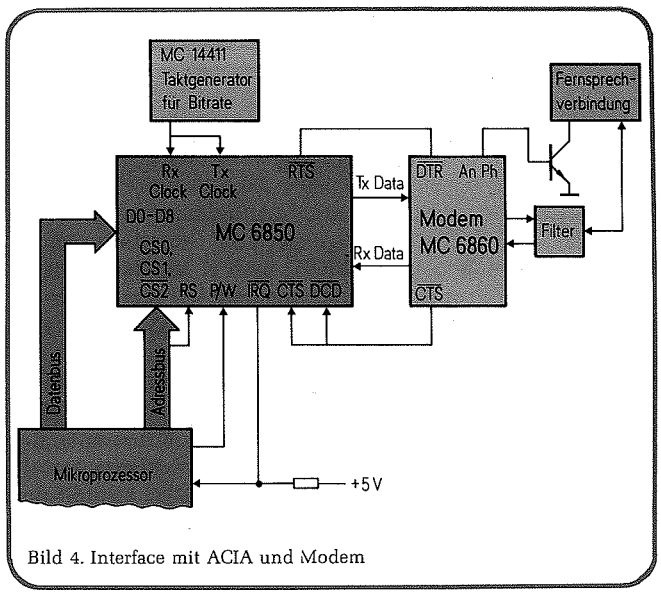


Bild 4. Interface mit ACIA und Modem

Ein CMOS-Impulsgenerator (MC14411), der über 16 Standard-Impulsfrequenzen für Übermittlungszwecke verfügt, bildet den Taktgeber für den ACIA. Der Sende- und Empfangsteil des ACIA besitzen getrennte Eingänge für den Taktimpuls, um, falls erwünscht, unabhängige Übertragungsgeschwindigkeiten erzielen zu können.

4 Die Software

Da die internen Register des ACIA und anderer Interface-Adapter für den Mikroprozessor wie Speicherzellen aussehen, sind bei Verwendung von Interface-Adaptern keine speziellen Befehle nötig. Die gebräuchlichsten, um Information in den ACIA einzuschreiben und von dort wieder auszulesen, sind die Befehle „Speichern" (STA) und „Laden" (LDA). Ein STA-Befehl bewirkt, daß die Lese/Schreibleitung

Tabelle 2. Auswahl der ACIA-Register durch die Signale R/W und RS

Adresse	STA-Befehl (R/W = 0)	LDA-Befehl (R/W = 1)
8400 (RS = A 0 = 0)	Steuerregister	Statusregister
8401 (RS = A 0 = 1)	Übertragungs-register	Empfangs-register

R/W des Mikroprozessors L-Pegel annimmt, ein LDA-Befehl hingegen bewirkt H-Pegel. Das Zuordnen aufeinanderfolgender Adressen durch die Verbindung eines Bits des Adreßbus mit den ACIA-Eingängen für Registerauswahl (RS) zusammen mit „Lesen/Schreiben" (R/W) erlaubt den Zugriff auf eines der vier ACIA-Register gemäß *Tabelle 2*. Der Befehl „STA in die Adresse 8400" z. B. bewirkt das Einschreiben ins Steuerregister des ACIA, während mit dem Befehl „LDA mit derselben Adresse" aus dem Statusregister ausgelesen wird.

Vor dem Senden und Empfangen von Daten muß der ACIA initialisiert werden. Dies wird gemacht, indem ein Wort ins Steuerregister eingespeichert wird, dessen Bits b 0 und b 1 den Zustand „L" aufweisen. Dann werden das Zähler-Teilungsverhältnis, die Wortlänge, Parität und Unterbrechungssteuerung ins Steuerregister einprogrammiert.

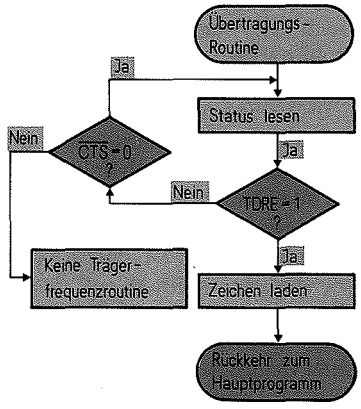


Bild 5. Flußdiagramm und Quellprogramm für die Sende-Routine

```
NEXT     LDA A STACON
         Status laden
         ASR A
         ASR A
         TDRE-Bit → Carry-Bit-Position
         BCS TX Data
         TDRE-Bit prüfen
         ASR A
         ASR A
         CTS-Bit → Carry-Bit-Position
         BCC NEXT
         CTS-Bit prüfen
         BR ERROR 1
         Sprung → „Keine Trägerfrequenz"
TX Data  STA B TXRX
         Zeichen → ACIA
         RTS
         Rückkehr vom Unterprogramm
```

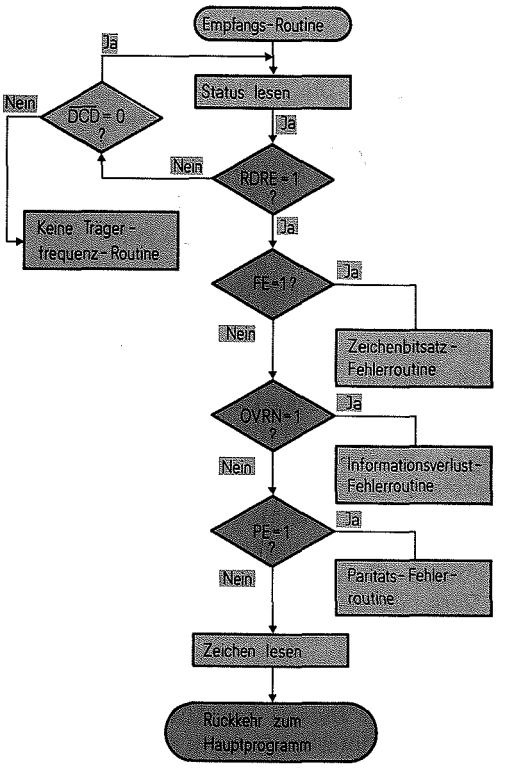


Bild 6. Flußdiagramm und Quellprogramm für die Empfangs-Routine

```
NEXT 1   LDA A STACON
         Status laden
         ASR A
         RDRF-Bit → Carry-Bit-Position
         BCS FRAM
         RDRF-Bit prüfen
         ASR A
         ASR A
         DCD-Bit → Carry-Bit-Position
         BCC NEXT 1
         DCD-Bit prüfen
         BR ERROR 2
         Sprung → „Keine Trägerfrequenz"
FRAM     ASR A
         ASR A
         FE-Bit → Carry-Bit-Position
         BCC OVRN
         FE-Bit prüfen
         BR ERROR 3
         Sprung → „Zeichenbitsatz-Fehler"
OVRN     ASR A
         OVRN-Bit → Carry-Bit-Position
         BCC PAR
         OVRN-Bit prüfen
         BR ERROR 4
         Sprung → „Informationsverlust"
PAR      ASR A
         PE-Bit → Carry-Bit-Position
         BCC R Data
         PE-Bit prüfen
         BR ERROR 5
         Sprung → „Paritätsfehler"
R DATA   LDA B TXRX
         Akku B mit Zeichen laden
         RTS
         Rückkehr vom Unterprogramm
```

Tabelle 3. Bedeutung der einzelnen Bits im Statusregister

b 0	RDRF	Signal „Empfangsregister besetzt" (*Receive Data Register Full*)
b 1	TDRE	Signal „Senderegister frei" (*Transmit Data Register Empty*)
b 2	DCD	Signal „Trägerfrequenz vorhanden" (*Data Carrier Detect*)
b 3	CTS	Signal „zum Senden bereit" (*Clear to Send*)
b 4	FE	Fehlersignal für Zeichenbitsatz (*Framing Error*)
b 5	OVRN	Fehlersignal für Informationsverlust (*Overrun Error*)
b 6	PE	Fehlersignal für Parität (*Parity Error*)
b 7	IRQ	Unterbrechungssignal (*Interrupt Request*)

Nach der Initialisierung ist der ACIA bereit zur Datenübertragung. Wegen der Länge der übermittelten Nachrichten wird die Datenübertragung in der Regel von Unterprogrammen ausgeführt, um die wiederholte Verwendung derselben Befehle zu reduzieren. Typische Beispiele von Flußdiagrammen und zugehörigem Quelltext für Sende- und Empfangs-Unterprogramme sind in den *Bildern* 5 und 6 dargestellt. *Tabelle* 3 zeigt die Bedeutung der einzelnen Bits im Statusregister.

Bei der Sende-Routine in Bild 5 wird der Inhalt des ACIA-Statusregisters in den Akkumulator des Mikroprozessors geladen. Mit Hilfe des Programms wird ein zu sendendes Zeichen in den ACIA eingespeichert, falls das Senderegister frei ist. Durch den Befehl RTS wird dann die Programmausführung vom Unterprogramm wieder auf das Hauptprogramm übertragen. Wenn der Pegel des Signals „Senderegister frei" (TDRE) auf L liegt und damit anzeigt, daß das Senderegister besetzt ist (oder der Pegel des $\overline{\text{CTS}}$-Einganges auf H liegt und damit das TDRE-Signal verhindert), sollte die $\overline{\text{CTS}}$-Statusinformation, die zuvor in den Akkumulator geladen wurde, auf ihre Aussage überprüft werden (dieser Schritt ist jedoch nicht nötig, wenn der Pegel des $\overline{\text{CTS}}$-Einganges dauernd auf L gehalten wird). Wenn in einem System Modems benützt werden, bedeutet H-Pegel am $\overline{\text{CTS}}$-Eingang, daß die Trägerfrequenz des Modems nicht vorhanden oder ausgefallen ist, was ein Wiederherstellen des Übertragungskanals erfordert. Ein auf L liegendes Statusregisterbit zeigt an, daß das Senderegister besetzt ist. Das Statusregister wird nun für dieses Bit in einer Programmschleife so lange wiedergelesen, bis das Senderegister abgearbeitet ist. Ist dies der Fall, wird das zu übermittelnde Zeichen im Senderegister gespeichert, und die Ausführung des Programms wird wieder vom Hauptprogramm übernommen.

Die Empfangs-Routine in Bild 6 ist ähnlich aufgebaut. Im ersten Schritt wird der Inhalt des Statusregisters in den Akkumulator des Mikroprozessors geladen. Wenn der Pegel des Signals „Empfangsregister besetzt" (RDRF) auf L liegt, wird damit angezeigt, daß das Empfangsregister frei ist oder die Empfangsschaltung durch den auf H liegenden $\overline{\text{DCD}}$-Eingang gesperrt ist. Deshalb sollte das $\overline{\text{DCD}}$-Statusbit, das zuvor in den Akkumulator geladen wurde, mit Hilfe des Programms auf seinen Zustand untersucht werden (dieser Schritt ist nicht nötig, falls der $\overline{\text{DCD}}$-Eingang dauernd L ist). In einem System mit Modems mittlerer Übertragungsgeschwindigkeit bedeutet ein während des Zeichenempfangs auf H liegender $\overline{\text{DCD}}$-Eingang, daß die zu empfangende Trägerfrequenz ausgefallen ist und daß der Übertragungskanal wieder hergestellt werden muß. Das $\overline{\text{DCD}}$-Statusbit wird auf L-Pegel zurückgesetzt, durch: 1. L-Pegel am $\overline{\text{DCD}}$-Eingang; 2. einen allgemeinen Reset; 3. Lesen des Empfangsregisters, nachdem das Statusregister gelesen wurde. Wenn das $\overline{\text{DCD}}$-Statusbit L ist, wird der Status für dieses Bit in einer Programmschleife so lange wiedergelesen, bis das Empfangsregister voll besetzt ist. Wenn der Pegel der Statusbit-Position b 0 auf H liegt (RDRF = H) und damit angezeigt wird, daß ein Zeichen empfangen wurde, ist der Status bezüglich des Zeichen-Bitsatzes, des Informationsverlusts und der Paritätsfehler des empfangenen Zeichens erhältlich. Ein auftretender Statusfehler könnte eine nochmalige Übermittlung der Nachricht oder fehlerkorrigierende Maßnahmen veranlassen. Wenn beim Zeichenempfang keine Fehler aufgetreten sind, wird das Empfangsregister gelesen, und mit einem RTS-Befehl wird die Führung des Programmverlaufs wieder dem Hauptprogramm übertragen.

Bei Systemen, die durch Unterbrechungen gesteuert werden, kann der ACIA so programmiert werden, daß voneinander unabhängige Unterbrechungen vom Sende- oder Empfangsteil auftreten. Eine beispielsweise nur vom Sendeteil ausgehende Unterbrechung kann erzeugt werden, indem die Sendeunterbrechung freigegeben (CR5 = H, CR6 = L) und die Empfangsunterbrechung gesperrt wird (CR7 = L). Damit tritt eine Unterbrechung auf, wenn das Senderegister frei ist (TDRE = H). Der Zustand des TDRE-Bit ist also bekannt, und eine Überprüfung dieses Zustandes ist unnötig, wie es aus dem Sende-Unterprogramm von Bild 5 ersichtlich ist.

Für den Zugriff auf eines der vier Register im ACIA sind also nur die Befehle STA und LDA nötig. Es sollte jedoch darauf geachtet werden, daß für den ACIA keine anderen Befehle verwendet werden, die ein automatisches Wiedereinschreiben bewirken. Dies würde zur Wahl von zwei Registern durch einen Befehl führen, da die Register im ACIA entweder nur zum Schreiben oder nur zum Lesen bestimmt sind. Die Mikroprozessor-Befehle, die für den Betrieb eines ACIA nicht benützt werden sollten, sind: ASL, ASR, COM, DEC, INC, LSR, NEG, ORA, ROR und ROL.

Literatur

[1] M6800 Mikrocomputer Handbuch. Druckschrift der Fa. Motorola.

[2] Applikationsschriften AN-731 und AN-747 der Fa. Motorola.

[3] M6800 Microcomputer System Design Data. Druckschrift der Fa. Motorola.

Karl Fronheiser, B.S.E., studierte an der Arizona State University und ist seit 1968 bei der Firma Motorola in Mesa, Arizona, beschäftigt. Zur Zeit arbeitet er im Applikationslabor für bipolare Systeme. Zuvor befaßte er sich mit MOS-Bausteinen für die Nachrichtentechnik.
Hobbys: Skifahren, Schwimmen und Holzarbeiten

Dipl.-Ing. Reinhard Gößler

Entwicklungshilfsmittel für die Mikrocomputer-Programmierung

Erst mit einem fertig entwickelten und ausgetesteten Anwenderprogramm im Speicher kann ein Mikrocomputer eine bestimmte Aufgabe erfüllen. Um ein solches Programm erstellen zu können, ist der Anwender auf umfassende Unterstützung durch die Herstellerfirmen angewiesen. Die hierzu angebotenen Entwicklungshilfsmittel zum Schreiben, Modifizieren und Prüfen eigener Programme reichen vom einfachen Mikrocomputer der Preisklasse um 1000 DM bis hin zum voll ausgebauten Entwicklungssystem, das mit umfangreicher Peripherie-Ausstattung bis in die Größenordnung von 100 000 DM kommt. Der nachfolgende Beitrag geht zunächst auf die grundlegenden Hardware- und Software-Eigenschaften der verfügbaren Entwicklungshilfsmittel sowie deren Anwendung ein und stellt dann repräsentative Entwicklungssysteme vor, ergänzt durch Beispiele von deren Leistungsfähigkeit. Der Entwickler von Mikrocomputer-Systemen erhält damit eine detaillierte Entscheidungsgrundlage für die Zusammenstellung einer geeigneten Gerätekombination.

Die Entwicklung von Mikrocomputer-Programmen unterscheidet sich ganz wesentlich von der Programmierung eines Minicomputers oder einer Großrechenanlage. Während bei den letztgenannten Rechnern der Bezug zur Hardware kaum oder gar nicht mehr gegeben ist, sind beim Mikrocomputer-Einsatz Hardware und Software so untrennbar miteinander verbunden wie auf keinem anderen Gebiet der Elektronik. Demzufolge muß bei der Geräte- und Systementwicklung außer dem Hardware-Entwurf ein daran angepaßter Programmentwurf erstellt werden bzw. umgekehrt – je nach Schwerpunkt der vorliegenden Aufgabenstellung. Zur Bewältigung dieser speziellen Entwicklungsaufgaben gibt es zahlreiche Hilfsmittel, deren Eignung für einen bestimmten Anwendungsfall in entscheidendem Maße von den Hardware- und Software-Eigenschaften abhängig ist. Erst durch den Einsatz derartiger Entwicklungshilfsmittel ist der Anwender in der Lage, eigene Programme für sein Mikrocomputer-System zu erstellen.

1 Programmentwicklung

Unter der Programmentwicklung ist derjenige Vorgang zu verstehen, der aus einem gegebenen Vorrat von Standardbefehlen diejenige Sequenz zusammenstellt, die zur Lösung einer bestimmten Aufgabe geeignet ist. Die Vorgehensweise dabei ist iterativ und umfaßt sechs wesentliche Schritte, zu deren Realisierung die Entwicklungshilfsmittel dienen:

- Programmerstellung
- Programmumsetzung
- Probelauf
- Programmänderung
- Fixieren des Programms
- Dokumentation.

Der Programmerstellung gehen die Problemanalyse und ein erster Entwurf für die Vorgehensweise bei der Programmierung voraus. Die Analyse erfolgt mit dem Ziel, die gestellte Aufgabe in Teilaufgaben zu untergliedern, um jede einzelne durch Hardware oder Software lösen zu können. Diese Zuweisung nimmt man entweder nach Wirtschaftlichkeitserwägungen vor, bei denen beispielsweise der Preis entscheidet, ob eine Zeitverzögerung durch eine Programmschleife oder eine Zählerkette realisiert wird, oder es sind hierfür übergeordnete Gesichtspunkte maßgebend, die etwa durch eine geforderte minimale Ausführungszeit oder die Hardware-Ausstattung des Mikrocomputers selbst festliegen. Beim Programmentwurf schlüsselt man die verschiedenen Software-Blöcke in einem Flußdiagramm auf, das als Detailkonzept zur Programmierung dient.

1.1 Programmerstellung

Das vorliegende Flußdiagramm soll auf eine möglichst einfache Weise in das sogenannte *Quellenprogramm* umgesetzt werden. Dazu gibt es eine Reihe von Programmiersprachen, die den Codierungsvorgang für den Programmierer durch symbolische Abkürzungen vereinfachen. Die Programmerstellung selbst ist auf verschiedenen Ebenen möglich; die unterste Ebene bildet die *Maschinensprache*, bei der Befehle, Adressen und Operanden in der binären Darstellungsform eingegeben werden. Die *Assemblersprache* erlaubt die Programmierung auf einer höheren Ebene, das heißt in einer der menschlichen Sprache näheren Form; Adressen und Operanden werden hierbei mit symbolischen Namen oder Abkürzungen dargestellt. Die Programmierung in Maschinen- oder Assemblersprache erfolgt maschinenbezogen, also an einen bestimmten Mikroprozessor-Typ gebunden. *Höhere Programmiersprachen* ermöglichen die Programmerstellung in einer Form, die losgelöst ist von einem spezifischen Prozessor und die sich auf keinen bestimmten Hardware-Aufbau, beispielsweise auf eine bestimmte Registerstruktur, mehr bezieht. Das entsprechende Programm, das die Befehlseingabe und Programm-Modifikationen ermöglicht, nennt man *Editor*.

1.2 Programmumsetzung

Die formale Umsetzung eines Quellenprogramms in das entsprechende *Objektprogramm*, das die Maschine unmit-

telbar verstehen kann, erfolgt in der Regel durch Umsetzprogramme, wenn man von der Möglichkeit der Handcodierung einmal absieht. Liegt ein Quellenprogramm in Assemblersprache vor, dient zur Umsetzung ein Programm mit gleicher Bezeichnung, nämlich der *Assembler*. Höhere Programmiersprachen werden vom sogenannten *Compiler* in die maschinenorientierte Darstellungsform umgesetzt. Unter *residenten* Programmen versteht man solche, die auf einer Maschine desselben Typs laufen, für die auch das Quellenprogramm geschrieben wurde; im anderen Fall spricht man von *Cross-Programmen* zur Umsetzung. Formfehler bei der Programmierung (*Syntax-Fehler*) werden bei der Umsetzung bereits automatisch erkannt und gelistet ausgedruckt. Nach der Korrektur dieser Fehler erfolgt eine erneute Programmumsetzung.

1.3 Probelauf und Programmänderung

Der Probelauf hat das Ziel, logische Fehler im Programm zu erkennen und nach deren Beseitigung eine Optimierung des Programms vorzunehmen. Zur ständigen Überwachung aller wichtigen Punkte in der Zentraleinheit während des Programmlaufs dient ein sogenanntes *Monitor-Programm*. Mit ihm lassen sich nach Vorgabe beliebiger Stoppadressen Register- oder Speicherinhalte anzeigen, um die korrekte Programmausführung im Detail nachvollziehen zu können. Eine Hardware-Anpassung zwischen Anwender-Zentraleinheit und Entwicklungssystem bezeichnet man als *In-Circuit-Emulator*, wofür man im deutschen Sprachgebrauch auch den Begriff *Emulations- und Testadapter* findet. Zur Beseitigung logischer Fehler oder für die Neuorganisation des Anwenderprogramms setzt man wiederum das Editor-Programm ein, dem ein erneutes Assemblieren und ein anschließender Probelauf folgen, bis das Programm zur Zufriedenheit arbeitet.

1.4 Fixieren des Programms

Um ein fertiges Programm im Anwender-System abzuspeichern, legt man es in einem PROM ab. Die entscheidende Aufgabe eines Entwicklungssystems besteht darin, das Objektprogramm so aufzubereiten, daß damit eine möglichst einfache PROM-Programmierung durchführbar ist. Im Idealfall nimmt das Entwicklungssystem selbst diese PROM-Programmierung vor, so daß separate Programmiergeräte nicht mehr erforderlich sind. Für die Beurteilung der Leistungsfähigkeit eines Entwicklungssystems ist diese Eigenschaft von wesentlicher Bedeutung, weil sie zusätzliche Investitionskosten und Arbeitsaufwand einspart.

1.5 Dokumentation

Zur Dokumentation zählen alle Vorgänge, die den detaillierten Programmausdruck ermöglichen oder die vollständige Programme auf externe Massenspeicher überschreiben, um sie dort zu archivieren. Das Entwicklungssystem muß derartige Interface-Schaltungen besitzen oder zumindest zum Anschluß solcher Peripheriegeräte vorgesehen sein.

2 Entwicklungshilfsmittel

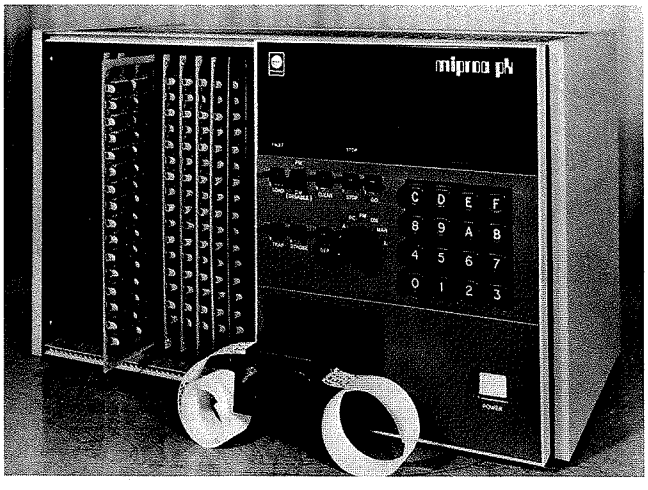
Entsprechend der im ersten Abschnitt geschilderten Vorgehensweise bei der Mikrocomputer-Programmierung besteht die wesentliche Unterstützung bei der Programmentwicklung auf der Software-Seite. Demzufolge ist es auch nicht richtig, nur die Hardware-Ausstattung eines Entwicklungssystems zu betrachten, weil die eigentliche Leistungsfähigkeit in der Organisation der Entwicklungs-Programme liegt. Eine direkte Verknüpfung zur Hardware besteht insofern, als externer Speicherumfang oder Zugriffsgeschwindigkeit zum Speichermedium davon beeinflußt werden, um nur zwei wesentliche Faktoren zu nennen.

2.1 Entwicklungs-Software

Die Entwicklungs-Software für einen bestimmten Mikrocomputer kann je nach Ursprungsfirma deutliche Leistungsunterschiede aufweisen. Es ist also keineswegs damit getan, irgendein Software-Paket zu einem bestimmten Computer zu kaufen, sondern es lohnt sich bei Konkurrenzangeboten ein Vergleich. Nur bei speziellen Mikrocomputern, für die oft auch kein Zweithersteller existiert, ist man auf das Unterstützungsangebot einer Herstellerfirma angewiesen, das man entsprechend sorgfältig prüfen sollte. Ein sehr wichtiger Gesichtspunkt bei der Software-Beurteilung ist deren Kompatibilität für eine ganze Mikroprozessor-Familie. In diesem Fall läßt sich ein Entwicklungsprogramm nicht nur für einen einzelnen Mikrocomputer einsetzen, sondern es ist beispielsweise auch zum Editieren und Assemblieren bei verwandten Modellen geeignet. Dadurch erspart man sich den Kauf mehrerer Entwicklungssysteme und bleibt von den Hardware-Einsatzmöglichkeiten her doch flexibel. Insbesondere bei den Mikrocomputern der sogenannten „Dritten Generation", aber auch bei umfangreich ausgebauten Mikroprozessor-Familien findet man heute eine derart leistungsfähige Software.

2.1.1 Betriebssystem

Ein Betriebssystem übernimmt innerhalb des Mikrocomputer-Entwicklungssystems die Verwaltung und den Aufruf der verschiedenen Entwicklungs-Unterprogramme. Am Beispiel des Intel-Betriebssystems ISIS (*Intel System Implementation Supervisor*), das fest auf einer Magnetplatten-Einheit gespeichert ist, soll zunächst die Arbeit mit dem



◀ Als eines der wenigen Entwicklungssysteme, die für schnelle TTL-Mikroprozessoren existieren, bietet die Firma Plessey für ihren 16-bit-Mikroprozessor MIPROC-16 das gezeigte Entwicklungssystem PK an ▶

Von der Firma Rockwell wird für die umfassend ausgebaute Mikroprozessor-Familie PPS-8 das kompakte Entwicklungssystem Assemulator angeboten, das bereits einen schnellen Lochstreifenleser enthält

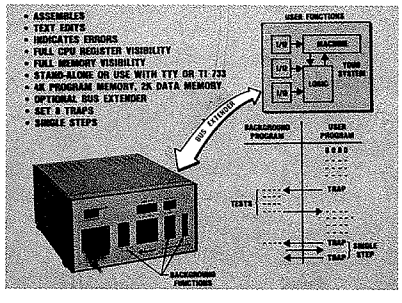


Tabelle 1. Funktion der fünf möglichen Durchläufe des Macro-Assemblers der Firma NEC Electronics

Pass 1	Der Assembler liest den Quellenstreifen ein und bildet im internen Arbeitsspeicher eine Symboltabelle
Pass 2	Der Assembler liest den Quellenstreifen ein und druckt über die Konsole das assemblierte Programm (*assembly listing*) aus
Pass 3	Der Assembler liest den Quellenstreifen ein und stanzt nach der schrittweisen Umsetzung den Objektstreifen aus
Pass 4	Der Assembler liest den Quellenstreifen ein und führt die Funktionen von Pass 2 und Pass 3 gleichzeitig aus (bei getrenntem Stanzer und Drucker)
Pass 5	Kann Pass 2 ersetzen, wenn nur solche Zeilen aufgelistet werden sollen, die Fehler enthalten

Text-Editor veranschaulicht werden (*Bild 1*). Der Text „ISIS, V1.0" ist die Bereitmeldung des Betriebssystems. Der Anwender gibt daraufhin „EDIT EXMPL. SRC" ein; dies ist die Anweisung an das Betriebsprogramm, den Editor zu laden und die anschließend eingegebenen Informationen in dem mit „EXMPL. SRC" bezeichneten Plattenbereich abzuspeichern. Daraufhin kommt die Bereitmeldung des Editors („ISIS TEXT EDITOR, V1.1, NEW FILE"), und der Programmierer spezifiziert mit dem INSERT-Operator „I", daß der nachfolgend eingegebene Text in den Speicherbereich (*Buffer*) des Editors eingeschrieben wird. Im Beispiel sind drei Fehler enthalten, die anschließend mit Hilfe des Editors beseitigt werden: Die Eingabe „STAET" enthält einen Schreibfehler und sollte stattdessen „START" lauten; zwischen der Spezifikation „SP" für den Stack-Pointer und dessen sedezimalem Anfangswert „13C0" fehlt ein Komma, und hinter dem Label „PULSE" müßte ein Doppelpunkt stehen (beides Syntax-Fehler). Der Ausdruck von zwei Umschaltzeichen, die als $ erscheinen, schließt die Texteingabe ab.

```
ISIS, V1.Ø
-EDIT EXMPL.SRC

ISIS TEXT EDITOR, V1.1
NEW FILE
*I      ORG    4ØØH
STAET:  MVI    A,8ØH
        OUT    13H
        LXI    SP 13CØH
PULSE   MVI    A,ØØØØØØØ1B
        OUT    11H
        MBI    A,1
        CALL   DLYP1
         .      .
         .      .
         .      .
        JMP    PULSE
        END
$$

*B$$
*SSTAET$START$$
*ØLT$$
START:  MVI    A,8ØH
*SSP 13CØH$SP,13CØH$$
*ØLT$$
        LXI    SP,13CØH
*SPULSE$PULSE:$$
*ØLT$$
PULSE:  MVI    A,ØØØØØØØ1B
*
```

Bild 1. Programmausdruck bei der Arbeit mit dem Editor

```
PDA-8Ø MONITOR VER 1.Ø
*IO-HL

*L

*G-Ø

PDA-8Ø MACRO ASSEMBLER VER 1.3

P=1


P=2

                    ;PRINT ALPHABET 11/19/76  (RMO)
                    START:  MVI A,'A'         ;(A)='A'
  ØØØØ 3E41         LOOP:   CALL CONO         ;PRINT (A)
  ØØØ2 CD47F7               CPI 'Z'           ;COMPARE TO 'Z'
  ØØØ5 FE5A                 INR A             ;INCREMENT
  ØØØ7 3C                   JC LOOP           ;WAS (A)='Z'
  ØØØ8 DAØ2ØØ               CALL CRLF         ;YES, END LINE
  ØØØB CDC9F7               JMP START         ;RESTART PROGRAM
  ØØØE C3ØØØØ       CRLF   EQU ØF7C9H        ;CR, LF ROUTINE
  F7C9               CONO   EQU ØF747H        ;CONSOLE OUTPUT
  F747                      END               ;END OF PROGRAM
  ØØØØ

P=3

:1ØØØØØØØ3E41CD47F7FE5A3CDAØ2ØØCDC9F7C3ØØA6
:Ø1ØØ1ØØØØEF
:ØØØØØØØ1FF
```

Bild 2. Ausdruck während der drei Assembler-Durchläufe

2.1.2 Text-Editor

Die Arbeit mit dem Editor spielt sich, genau wie bei den übrigen Entwicklungsprogrammen, im Dialogverkehr zwischen Benutzer und Maschine ab. Dazu gibt es eine Reihe fest vereinbarter Anweisungen an das Programm, die nach der Eingabe ausgeführt und anschließend quittiert werden. Danach ist das Programm zur Aufnahme weiterer Anweisungen bereit. Im Beispiel von Bild 1 springt nach Eingabe des Operators „B" der *Buffer-Pointer* des Editors an den Beginn des Text-Buffers; unter dem Pointer ist ein Register-Inhalt zu verstehen, dessen numerischer Wert auf eine bestimmte Stelle im Speicherbereich weist. Durch diese Adressierung entsteht in der Wirkung ein symbolischer Zeiger.

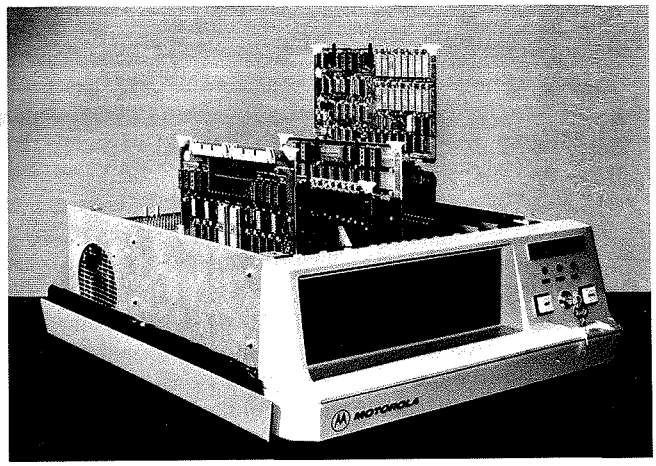
Die nachfolgende Anweisung „SDATEN1$DATEN2$" veranlaßt das Editor-Programm zum Suchen des alphanumerischen Textes „DATEN1" und dessen Austausch (*Substitute*) mit dem neuen Text „DATEN2"; hierdurch wird im gezeigten Beispiel der Schreibfehler im Wort „STAET" korrigiert. Mit der Eingabe „0LT", die aus den zwei Editor-Anweisungen „0L" (*0 line*) und „T" (*type*) zusammengesetzt ist, bringt man den Buffer-Pointer an den Zeilenanfang der laufenden Zeile und läßt diese anschließend ausdrucken; dies dient zur Verifizierung der Korrektur. Auf dieselbe Weise werden die übrigen Fehler beseitigt, soweit sie in diesem Stadium erkennbar sind.

2.1.3 Assembler

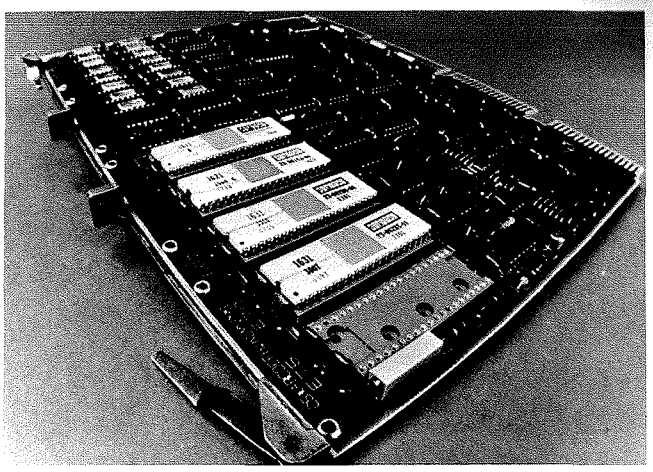
Der Umsetzvorgang des Quellenprogramms soll am Beispiel des Macro-Assemblers PDA-80 der Firma NEC Electronics nachvollzogen werden (*Bild 2*), der ebenfalls in einem Entwicklungssystem für den Mikroprozessor 8080 läuft. In diesem System erfolgen das Laden und der Aufruf des Assemblers über das übergeordnete Betriebsprogramm „MONITOR" (Zeilen 1...4 in Bild 2), was vom Prinzip her genauso abläuft wie im Abschnitt 2.1.2 beschrieben. Das Assemblieren vollzieht sich in mehreren Durchläufen (*passes*) des vom Editor ausgegebenen Quellencodes. Der hier betrachtete Assembler benötigt minimal zwei Durchläufe zur Erstellung des Objektcodes, kann darüber hinaus aber noch drei andere Durchläufe mit unterschiedlichen Funktionen ausführen (*Tabelle 1*). Üblicherweise spielt sich das Umsetzen in den drei Stufen ab, die in Bild 2 ersichtlich sind; nach der Bereitmeldung des Assemblerprogramms „PDA-80 MACRO ASSEMBLER VER 1.3" wird automatisch „P=" ausgedruckt. Der Assembler erwartet daraufhin die Benutzer-Eingabe 1...5 zur Spezifikation des gewünschten Durchlaufs. Die Reaktion erfolgt jeweils in der in Tabelle 1 beschriebenen Weise.

2.1.4 Simulator

Von einigen Herstellern werden sogenannte Simulator-Programme angeboten, die auf Systemen mit größerer Speicher-Ausbaustufe laufen und das Verhalten eines bestimmten Mikroprozessors softwaremäßig nachbilden. Diese Programme sind so gestaltet, daß sie Eingriffe in die simulierte Hardware erlauben und als Arbeitsgrundlage den Objektstreifen des Anwenderprogramms benutzen. Durch die Arbeit mit dem



Das Motorola-Entwicklungssystem „Exorciser" gehört mit seinem robusten Aufbau und der modularen Struktur zu den Standard-Systemen des Marktes



Diese Karte enthält den Mikrocomputer LSI-11 der Firma Digital Equipment, der bereits die Leistungsfähigkeit eines Minicomputers besitzt und als Zentraleinheit in einem Entwicklungssystem eingesetzt werden kann

Simulator lassen sich nur logische oder organisatorische Programmfehler aufdecken; eine schlüssige Aussage darüber, ob später auch die Hardware-Konfiguration arbeitet, ist durch eine Simulation nicht möglich. Das liegt erstens daran, daß die Nachbildung nicht immer im Echtzeit-Betrieb realisierbar ist, und zweitens können bestimmte Hardware-Eigenschaften kaum oder überhaupt nicht nachgebildet werden; dazu gehören beispielsweise Durchlaufzeiten im Anwender-System, die die Funktion entscheidend beeinflussen. Dennoch ist in der frühen Entwicklungsphase der Einsatz eines Simulator-Programms sehr hilfreich, um zielstrebig die ersten Voruntersuchungen durchführen zu können.

2.1.5 Ladeprogramme (Loader)

Bevor man irgendein Programm aufrufen kann, sei es ein Entwicklungs-Hilfsprogramm oder das übergeordnete Betriebsprogramm, muß man es in den Speicher des Betriebssystems laden, um darauf zugreifen zu können. Da für diesen Ladevorgang wiederum ein separates Ladeprogramm benötigt wird, stößt man hier auf das Problem der Ursprungs-Initialisierung; es kann durch fest abgespeicherte ROM-Routinen, sogenannte *Ur-Lader*, oder auch durch spezielle Hardware-Konfigurationen gelöst werden.

2.1.6 Fehlersuch-Programme (Debugging Routines)

Unter diesem Begriff faßt man solche Programme zusammen, die zur Inbetriebnahme oder Fehlersuche in Anwenderprogrammen geeignet sind. Die Sprachregelung hierbei ist keinesfalls einheitlich, so daß in jedem Fall zu klären ist, was ein angebotenes Programm im einzelnen leistet. Oftmals findet man bei der Entwicklungs-Software Kombinationen aus den beschriebenen Programmen, die unter einer firmenspezifischen Sammelbezeichnung (z. B. „Assemulator") zusammengefaßt sind. Derartige Zusammenstellungen macht man in der Regel deshalb, weil man aus Kostengründen kompakte Programme anbieten will, die dennoch möglichst viele der geschilderten Funktionen enthalten sollen.

2.2 Timesharing-Betrieb

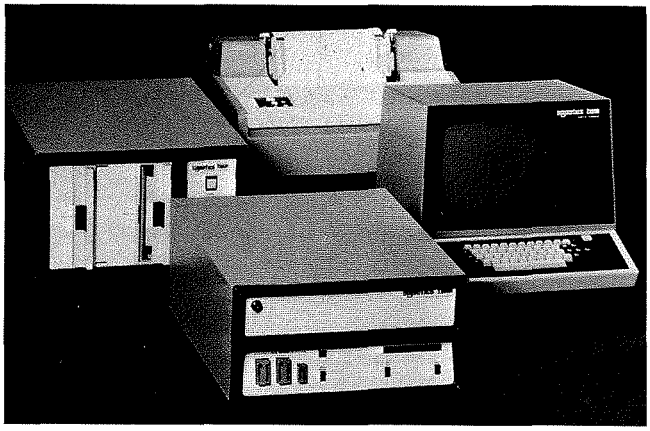
Es herrscht vielfach die Meinung vor, daß man die Entwicklungshilfsmittel selbst besitzen müsse, um eine Entwicklung durchführen zu können. Dies ist nur bedingt richtig, weil die Software-Unterstützung auf dem Wege der Datenfernverarbeitung vielfach auch im *Timesharing-Betrieb* zur Verfügung steht. Allerdings sind die entsprechenden Mietverträge recht teuer, so daß sie sich nur bei großen Projekten, kurzer Entwicklungszeit und ausgebildetem Personal rentieren.

2.3 Entwicklungs-Hardware

Die Hardware zur Entwicklungs-Unterstützung übernimmt diejenigen Hilfsaufgaben, die zum Speichern, Ausführen und Bedienen der Entwicklungs-Software erforderlich sind oder die den Anschluß des Entwickler-Systems an das Entwicklungs-System ermöglichen. Im allgemeinen sind diese Hardware-Konfigurationen um einen leistungsfähigen Mikrocomputer herum aufgebaut, der dem im Anwendungssystem eingesetzten entspricht. Die übrige Hardware-Ausstattung ist modular nachrüstbar, so daß man beispielsweise den Speicherausbau oder die Bestückung mit weiteren Interface-Schaltungen nach und nach vornehmen kann. Um aber die hierfür erforderlichen hohen Investitionskosten einsparen zu können, bieten verschiedene Hersteller oder Vertriebsfirmen Mikrocomputer in minimaler Ausbaustufe an, deren Leistungsfähigkeit zur Entwicklung kleiner und mittlerer Programme (500...1000 Befehle) durchaus genügt. Besonders für einen ersten Einstieg und für kleinere Firmen eignen sich solche *Entwicklungs-Kits*, bei deren Kauf man auf Erweiterungsmöglichkeiten, insbesondere für den Speicherausbau, achten sollte.

2.3.1 Mikrocomputer

Kernstück des Entwicklungssystems ist der zentrale Mikrocomputer, dessen Arbeitsspeicher sowohl die Entwicklungssoftware als auch das Anwenderprogramm aufnimmt. Für kleine Anwendungen, in denen keine umfangreichen Berechnungen auftreten, liegt die untere Grenze für den Arbeitsspeicher bei 1 KWorten, wenn parallel dazu ein fest gespeichertes Betriebsprogramm von mindestens der gleichen Kapazität zur Verfügung steht. Bereits bei Aufgaben mit mittlerem Umfang (Programme bis 4 KWorte) sollte der Arbeitsspeicher einen Bereich von 8 K umfassen. Für größere Aufgaben, beispielsweise bei der Echtzeit-Signalverarbeitung, sind noch umfangreichere Speicher erforderlich, die nicht selten bis 64 K heraufreichen. Neben der Speicherausbaumöglichkeit ist auf das Vorhandensein von Interface-Karten zu achten, über die periphere Geräte angeschlossen werden können.



Das Entwicklungssystem für den Signetics-Mikroprozessor 2650 trägt die Bezeichnung Twin und ist hier zusammen mit Floppy-Disk-Einheit, Matrix-Drucker und Datensichtgerät gezeigt

2.3.2 Peripherie-Geräte

Über die Peripherie-Geräte läuft die Kommunikation zwischen Benutzer und Maschine ab. Dazu gehören in erster Linie die Dateneingabe über eine Tastatur und die Ausgabe über einen Blattschreiber oder ein Datensichtgerät. Außerdem ist die Ein- und Ausgabemöglichkeit von Lochstreifen erforderlich, solange das Assemblieren und die Archivierung nicht auf andere Weise vorgenommen werden. Diese genannten Grundfunktionen kann die Fernschreibmaschine mit angebautem Leser/Stanzer übernehmen. Sie ist gemeinhin als das Standard-Peripherie-Gerät verbreitet, hat aber einige wesentliche Nachteile, durch die sie in Zukunft weitgehend verdrängt werden wird. Hierzu zählt insbesondere die langsame Arbeitsgeschwindigkeit, aber der hohe Geräuschpegel und die Unmengen an ausgeworfenem Papier wirken sich ebenfalls nachteilig aus; als Vorteil ist lediglich der relativ günstige Anschaffungspreis zu nennen.

Bedeutend umweltfreundlicher arbeitet das Datensichtgerät, das allerdings allein nicht ausreicht und zusätzlich die Anschaffung eines Druckers und gegebenenfalls einer schnellen Leser/Stanzer-Einheit erforderlich macht.

Wegen des schnellen Zugriffs, der problemlosen Handhabung und der großen Speicherkapazität bietet sich die Floppy-Disk-Einheit gleichermaßen als externer Programm- und Massenspeicher an. Allein durch diese Eigenschaften wird sich die Floppy-Disk-Einheit als Standard-Peripherie-Gerät etablieren, und bei der Anschaffung eines Entwicklungssystems sollte sie bereits zur Grundausstattung gehören.

In zunehmendem Maße gewinnen auch die Kassettenlaufwerke an Bedeutung, bei denen herkömmliche Magnetbandkassetten als Massenspeicher eingesetzt werden; der Datenzugriff ist hierbei etwas langsamer, doch steht dem der relativ günstige Anschaffungspreis gegenüber.

2.3.3 PROM-Programmiereinrichtungen

Bisher gehörte zur Standardausstattung eines Mikrocomputer-Labors ein separates PROM-Programmiergerät, das den Objektstreifen des ausgetesteten Anwenderprogramms zur PROM-Programmierung verwendet. Diese Geräte sind in der Regel mit Anpaßschaltungen (*Personality Modules*) für die verschiedenen PROM-Typen versehen, so daß beispielsweise auch löschbare PROMs eingesetzt werden können; dies spielt vor allem während der Entwicklungsphase und bei häufigen Änderungen eine Rolle, weil die Modifikationen problemlos durchführbar sind.

Aktueller Stand der Technik sind die elektrisch änderbaren PROMs (*EEPROMs, electrically erasable PROMs*), die dem nichtflüchtigen Speicher bereits sehr nahe kommen bzw. ihn bereits darstellen. Zur Programmierung dieser PROM-Typen gibt es spezielle Einschübe für das Entwicklungssystem, und der Programmiervorgang wird hierbei bedeutend vereinfacht, weil als Zwischenstufe kein separater Lochstreifen erstellt werden muß.

Auch Hersteller und Distributoren programmieren PROMs bereits sehr preisgünstig, so daß in der Anfangsphase nicht unbedingt eigene Programmiereinrichtungen benötigt werden.

2.3.4 Prototypen-Systeme

Zur Überbrückung der Hardware-Entwicklungsschwierigkeiten bieten einige Hersteller sogenannte Prototypen-Systeme an. Darunter sind modular aufgebaute Mikrocomputer zu verstehen, die sowohl zur Entwicklung als auch zum späteren Einsatz im Anwender-System vorgesehen sind. Durch die fertig entwickelte Hardware ergeben sich vordergründig zwar Vorteile, doch wird man bei größerem Produktionsumfang auf eigene Entwicklungen ausweichen müssen, um durch angepaßte Lösungen günstigere Herstellungspreise zu erzielen.

Bild 3. Das Entwicklungssystem SME-800 der Firma Siemens ist umfassend ausbaufähig und erlaubt den Anschluß aller herkömmlichen Peripheriegeräte

3 Entwicklungssysteme

Das heutige Marktangebot an Entwicklungssystemen läßt sich generell in vier Gruppen einteilen:

- Umfassend ausgebaute, modulare Systeme für universellen Einsatz
- Systeme für spezielle Mikroprozessor-Familien
- Leistungsfähige Kompaktsysteme
- Entwicklungs-Kits.

Die Abgrenzung dieser Gruppen gegeneinander ist unscharf, aber die nachfolgend aufgeführten Beispiele können eine Vorstellung von dieser Klassifizierung geben. Die ausgewählten Modelle kann man als repräsentativ betrachten, jedoch ist mit ihrer Nennung keineswegs eine Bewertung gegenüber Konkurrenzmodellen verbunden.

3.1 Mikrocomputer-Entwicklungssystem SME-800

Dieses Entwicklungssystem der Firma Siemens ist für den Mikroprozessor 8080 vorgesehen, doch erlaubt die interne Hardware-Struktur auch die Arbeit an späteren Modellen. Das Chassis der Zentraleinheit (*Bild 3*) kann maximal 18 Pla-

tinen aufnehmen; in der Grundausbaustufe sind vier davon erforderlich: Prozessor-Modul, Bedienungsfeld-Modul, Monitor-Modul, 16-K-RAM-Modul.

Der Prozessor-Modul enthält den Mikrocomputer auf der Basis des Siemens-Mikroprozessors SAB 8080. Dazu gehören die Hilfsbausteine zur Takterzeugung und Systemsteuerung sowie die Bus-Steuerlogik mit Adreßpuffern. Der System-Datenbus ist bereits mit 16 bit ausgeführt, von denen beim 8080-Einsatz nur die unteren 8 benötigt werden. Durch diese Konzeption ist sichergestellt, daß später u. U. auch ein Einsatz an 16-bit-Prozessoren möglich ist.

Vom Bedienungsfeld-Modul werden die Steuerfunktionen für die Anzeige-Einheiten und Schaltereingaben ausgeführt. Außerdem ist hierauf ein 256 Worte umfassendes Urlade-Programm fest abgespeichert. Ferner enthält dieser Modul die Logik zur Bus-Verwaltung und zur Erzeugung von Unterbrechungsanforderungen.

Auf dem Monitor-Modul befinden sich ein 2-K-Festwertspeicher für das Monitor-Programm und die Anpaßschaltungen für Fernschreiber, Datensichtgerät, schnellen Leser-Stanzer, Matrix-Drucker und externes PROM-Programmiergerät.

Der RAM-Modul ist mit dynamischen N-Kanal-Speichern vom Typ SAB 8107 bestückt; damit werden die Lesezyklen innerhalb von 735 ns und die Schreibzyklen in maximal 1,36 µs ausgeführt. Die Refresh-Logik ist ebenfalls auf der Platine enthalten.

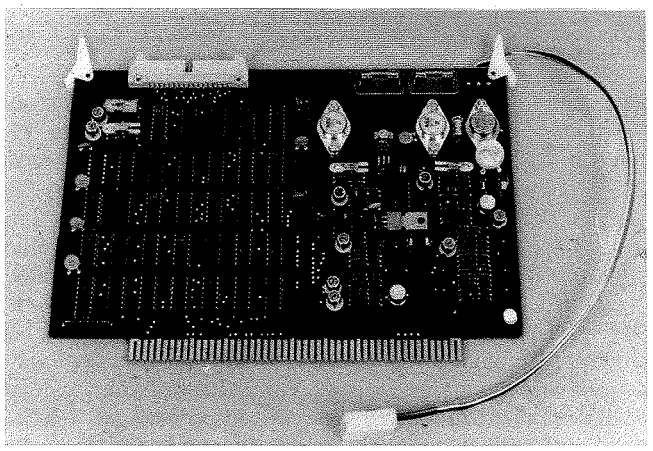
Zur Erweiterung dieses Systems können insgesamt vier der beschriebenen RAM-Moduln eingesetzt werden, wodurch sich der Arbeitsspeicher auf insgesamt 64 KWorte ausdehnt. Außerdem können maximal zwei PROM-Moduln nachgerüstet werden, von denen jeder 6 KWorte enthält und mit PROMs vom Typ SAB 8702A (bzw. 1702 oder 1302) bestückt ist. Als weitere Systemkomponenten werden Moduln angeboten, die den direkten Speicherzugriff ermöglichen (DMA-Modul) oder die eine Ein-/Ausgabe-Schnittstelle realisieren (E/A-Modul). Von wesentlicher Bedeutung ist außerdem die Platine zur Floppy-Disk-Steuerung, über die eine Zweifach-Platteneinheit anschließbar ist. Schließlich existieren weitere Moduln zum Anschluß und zum Datenverkehr mit dem externen Emulations- und Testadapter ETA.

In der Grundausbaustufe liegt dieses System in der preislichen Größenordnung von 15 000 DM, während es voll ausgebaut etwa das Dreifache kostet. In diesen Angaben ist selbstverständlich keines der genannten Peripherie-Geräte enthalten.

Die Software-Ausstattung enthält das Monitor-Programm, den Editor und Assembler sowie das Disketten-Betriebssystem ISIS. Die Software ist sehr gut ausgebaut, und vor allem das Betriebssystem erleichtert die Programmierung beträchtlich. Über eine Interface-Karte und das Modem 8331 ist außerdem ein Großrechner vom Typ 4004/151 oder 7.750 fernanschließbar, auf denen ein Cross-Assembler oder Simulator läuft. Hard- und Software-Organisation entsprechen denen der Firma Intel.

3.2 Entwicklungssystem für die 9900-Familie

Die Konzeption der Mikroprozessor-Familie 9900 der Firma Texas Instruments und die parallel dazu angebotene Entwicklungsunterstützung sind ein Beispiel für die hohe Flexibilität, die selbst mit einer speziellen Prozessor-Familie erreichbar ist. Dies kommt durch die Kompatibilität der Software zum Ausdruck, die nicht nur für die drei Mikroprozessoren SBP 9900 (I^2L), TMS 9900 und TMS 9980 (beide



Mit dieser Einschubkarte für das NEC-Entwicklungssystem PDA-80 ist die unmittelbare Programmierung von EPROMs möglich, ohne daß man den Umweg über einen Lochstreifen wählen muß

N-Kanal-MOS-Technologie) gegeben ist, sondern die sich auch auf die Mikro- und Minicomputer TMS 990/4 bzw. TMS 990/10 erstreckt (*Bild* 4). Diese letztgenannten Computer bilden die Zentraleinheit für das Entwicklungssystem, und je nach Anforderung kann man sich für den preisgünstigen Mikrocomputer TMS 990/4 oder den leistungsfähigeren Minicomputer TMS 990/10 entscheiden. Zu deren Bedienung ist lediglich das kompakte Peripherie-Gerät 733ASR erforderlich, das eine Tastatur, einen geräuschlosen Thermodrucker und ein Zweifach-Kassettenlaufwerk enthält. Dieses leistungsfähige Gerät ist eine Eigenentwicklung der Firma Texas Instruments, die damit eine angepaßte und dennoch vielseitige Hardware-Unterstützung liefert. Bei Einsatz des Minicomputers ist dieses System auch durch zusätzliche Peripherie-Geräte erweiterbar; dazu gehören Datensichtgerät, Floppy-Disk-Einheit, Lochkartenleser und Zeilendrucker. Der Arbeitsspeicher ist von 8 K...24 K ausbaufähig, und es können vom Anwender zusätzlich sowohl PROMs als auch EPROMs als nichtflüchtige Speicher eingesetzt werden. Die Software-Unterstützung umfaßt außer den Standard-Programmen einen Cross-Assembler, ein Simulations- und ein Ausgabe-Programm zum Überschreiben auf Kassette oder zum Programmieren von PROMs. Die letztgenannten Programme sind im Timesharing-Betrieb über Honeywell Bull oder das NCSS-Timesharing-System verfügbar und können auch in FORTRAN IV für Rechner mit Compiler bezogen werden.

3.3 Polyvalentes Entwicklungssystem PDS

Das kompakte Entwicklungssystem PDS, das die Firma Motorola als Entwicklungsunterstützung für ihren Mikro-

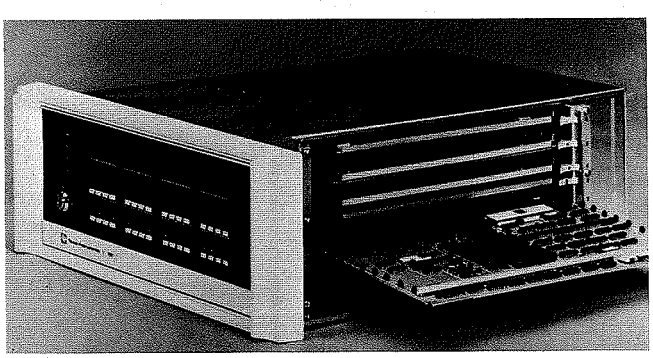


Bild 4. Die kompakten Minicomputer TMS 990/4 und TMS 990/10 der Firma Texas Instruments bilden die Zentraleinheit innerhalb des Entwicklungssystems für die 9900-Familie

prozessor 6800 anbietet, schließt die Lücke zwischen Ein-Karten-Mikrocomputer und umfassend ausgebautem Entwicklungssystem (*Bild 5*). In der Grundkonfiguration besteht es aus zwei Karten, nämlich der Hauptplatine M68SAC-1 mit dem Mikrocomputer selbst und einer Video-Karte M68DIM-1 zum Anschluß eines Datensichtgeräts. Die Kommunikation mit der Maschine erfolgt über eine Tastatur, und die gesamte Hardware ist ohne festen Aufbau in einem Handkoffer untergebracht. Dies ist nicht nur eine praktische Lösung, sondern sie spart erhebliche Herstellungskosten ein, die sonst für das Gehäuse anfallen würden. Das System wurde so konzipiert, daß es mit Baugruppen des großen Motorola-Entwicklungssystems „Exorciser" zusammenarbeiten kann. Dadurch kann man um die Anfangsausstattung herum mit wachsenden Anforderungen das System problemlos erweitern. Zur Software-Ausstattung der Erprobungsplatine gehört das Betriebsprogramm MINIBUG II, das fest in ROMs abgespeichert ist. Bereits mit dieser Ausrüstung ist der Entwickler in der Lage, kleine und mittlere Programme zu schreiben und in Betrieb zu nehmen. Darüber hinaus läßt sich direkt auch eine Fernschreibmaschine oder ein anderes serielles Peripheriegerät mit V-24-Schnittstelle anschließen.

3.4 Entwicklungs-Kit TK-80

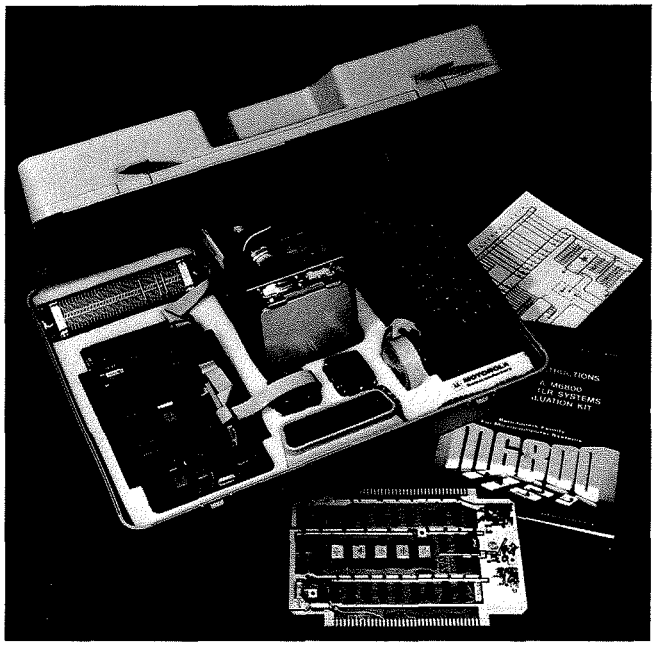
Prinzipiell sind die Programmentwicklung und das anschließende Austesten auch mit einem Ein-Karten-Mikrocomputer möglich, wie ihn beispielsweise die Firma NEC Electronics in seiner Minimalkonfiguration anbietet (*Bild 6*). Es handelt sich hierbei um einen vollständigen 8080-Mikrocomputer einschließlich programmierbarem Ein-/Ausgabe-Baustein, einer achtstelligen Anzeige zur sedezimalen Darstellung von Adressen und Daten sowie einer Tastatur zur Dateneingabe. Der Aufbau sieht bereits die Erweiterungsmöglichkeit für den Speicherausbau bis insgesamt 64 K vor. Es ist sogar die schrittweise Befehlsausführung möglich, und das 1-K-CMOS-RAM auf der Platine kann mit zwei Mignon-Zellen wochenlang gepuffert werden. Das in drei ROMs abgespeicherte Monitor-Programm enthält die wesentlichen Funktionen eines Editors und ermöglicht auch das Überschreiben und Einlesen des Speicherinhalts auf Bandkassetten. Ähnlich wie bei dem Motorola-System PDS ist auch hier der umfangreiche Hardware-Ausbau ohne weiteres möglich; insofern kann ein solches System die Keimzelle einer umfangreicheren Lösung bilden, die zur spezifischen Anpassung an die eigenen Bedürfnisse führt.

4 Software-Unterstützung

Es ist ein altes Problem bei der Programmentwicklung, daß jeder für sich „das Rad neu erfindet", d. h., er setzt sich mit der Lösung von Standard-Problemen auseinander, die in vielfacher Ausführung längst existieren. Dadurch gehen Unmengen an Kapital verloren, weil die Entwicklungszeit nicht sinnvoll genutzt wird. Einen Ausweg aus dieser Situation bilden die nach und nach entstehenden Programmbibliotheken der Hersteller, die eine Vielzahl von Standard-Routinen enthalten. Als Beispiel einer gut ausgebauten Programmbibliothek kann diejenige der Firma Intel dienen; unter der Bezeichnung INSITE (*Intel Software Index and Technology Exchange*) ist diese Organisation zusammengefaßt, die ausschließlich dem Benutzer zugute kommt, der sich dadurch langwierige Eigenentwicklungen ersparen kann. Insofern ist ein solches Software-Angebot mitunter die effektivste Entwicklungsunterstützung, weil sie zu einem minimalen Preis bereits fertige Problemlösungen anbietet.

5 Marktübersicht

Die Entscheidung zum Kauf eines bestimmten Mikroprozessors hängt in großem Maße von der dazu verfügbaren Entwicklungsunterstützung ab. Diese soll bereits viele fertige Lösungen für häufig auftretende Standard-Probleme enthalten und außerdem die Programmierung und Fehlersuche so weit wie möglich erleichtern. Die fundierte Beurteilung einer derartigen Leistungsfähigkeit ist durch bloßes Literaturstudium natürlich nicht möglich. Selbst nach einer Systemvorführung kann man nur einen oberflächlichen Eindruck gewinnen, so daß man vor dem Kauf eines größeren Entwicklungssystems unbedingt selbst Erfahrungen daran gesammelt haben sollte. Unter diesen Gesichtspunkten ist die Übersicht der *Tabelle 2* nur als eine Informationshilfe zu betrachten, auf die allein sich eine Kaufentscheidung nicht stützen kann.



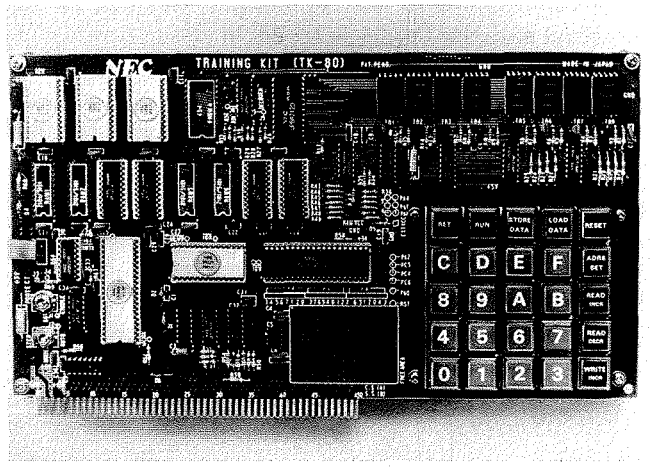


Bild 6. Am Beispiel des Ein-Karten-Mikrocomputers der japanischen Firma NEC Electronics wird deutlich, daß zur Programmentwicklung bei kleineren Aufgaben bereits diese Minimalkonfiguration genügt

◀ Bild 5. Motorola bietet mit diesem anpassungsfähigen Entwicklungssystem PDS eine kostengünstige Lösung für kleinere und mittlere Programme

Tabelle 2. Übersicht über die angebotenen Entwicklungssysteme

Hersteller	Bezeichnung	Mikro-prozessor	ausbau-fähig	spez. An-wendung	Kompakt-system	Entwick-lungs-Kit	Bemerkungen
AMI	Development Center	6800	●				Massenspeicher mit IBM-kompatibler Formatierung anschließbar
DEC	PDP-11	LSI-11	●				leistungsfähiger 16-bit-Minicomputer
Electronic Arrays	EASE 2000	EA 9002			●		handliches System mit übersichtlichem Aufbau
Fairchild	Formulator	F 8	●				umfangreicher Ausbau einschließlich Prototypen-Karten
General Instrument	GIMINI	CP 1600	●				enthält ROM-Betriebs-system; Raum für eigene Hardware-Erweiterungen vorgesehen
Harris	SIMON	6100			●		hohe Speicherausbaustufe trotz kompakten Aufbaus
Intel	MDS-800	8080	●				leistungsfähigstes Betriebssystem auf Diskette
	Intellec 8/MOD 8	8008	●				gute Software-Unterstützung
	Intellec 4/MOD 40	4004/4040	●				
	SDK-80	8080				●	erweiterbarer Kit
Mostek	Development-System	Z 80					leistungsfähigster 8-bit-Mikrocomputer
	Software-Development-Board	F 8					Zentrale Karte für ein Entwicklungssystem
	GEMS 8	MK 5065		●			Prototypen-Aufbau
Motorola	MEK 6800	6800				●	Version D 2 mit Kassetten-Recorder-Anschluß
	PDS	6800			●		flexibel erweiterbar
	Exorciser	6800	●				robustes und umfangreich ausbaufähiges System
MOS-Technology	MDT-650	MCS 6500		●			Kompatibilität zum 6800
NEC Electronics	PDA-80	8080	●				EPROM-Programmiereinschub
	TK-80	8080				●	Monitorprogramm mit Kassetten-Ein-/Ausgabe
National Semiconductor	angepaßte Entwick-lungssysteme	IMP-Serie	●				gute Hardware-Ausstattung und preiswerte Systeme
		PACE		●			
		SC/MP		●			
Plessey	MIPROC PK	Miproc-16		●			System für schnellen 16-bit-TTL-Prozessor
Rockwell	PPS-4MP Assemulator	PPS-4		●			schneller Leser und Eingabe-Kanäle integriert
	PPS-8MP Assemulator	PPS-8		●			
Texas Instruments	MM-80	8080				●	Prototypen System
	EP 1/2	TMS 1000		●			handliches und preis-wertes System
	990/4, 990/10	9900-Fam.		●			Software-Kompatibilität vom Mikroprozessor bis zum Minicomputer
	AMPS	9900-Fam.		●			Universelles System, um den Minicomputer 990/10 herum aufgebaut
Signetics	Twin	2650	●				Minicomputer-Leistungs-fähigkeit mit guter Hardware-Unterstützung
Siemens	SME-800	8080	●				breites Angebot an Erweiterungsmoduln

Literatur

[1] *Gößler, R.* und *Schwerte, J.*: Auf dem Weg zur Mikroprozessor-Praxis. ELEKTRONIK 1976, H. 3, S. 74...85.

[2] *Timm, V.*: Im Blickpunkt: ROMs, PROMs und PLAs. ELEKTRONIK 1976, H. 5, S. 38...47.

[3] *Schlenther, M.*: Bipolare und MOS-Schreib-/Lesespeicher (RAMs). ELEKTRONIK 1976, H. 10, S. 57...68.

[4] *Gößler, R.*: Ein-/Ausgabe-Bausteine für Mikroprozessoren. ELEKTRONIK 1976, H. 11, S. 62...68.

[5] Microcomputer LSI-11, PDP-11/03, PDP-11V03. Druckschrift der Firma Digital Equipment, November 1976.

[6] EASE 2000 Emulator. Datenblatt der Firma Electronic Arrays, Oktober 1976.

[7] Formulator User's Guide. Handbuch der Firma Fairchild, März 1976.

[8] Series 1600 Microprocessor System, Documentation. Handbuch der Firma General Instrument, Dezember 1976.

[9] SIMON Prototyping System. Produktbeschreibung der Firma Harris.

[10] Data Catalog 1976. Datenbuch der Firma Intel, 1976.

[11] Intellec Microcomputer Development System. Druckschrift der Firma Intel, 1976.

[12] Microcomputers Hardware Manual. Handbuch der Firma MOS-Technology, Januar 1976.

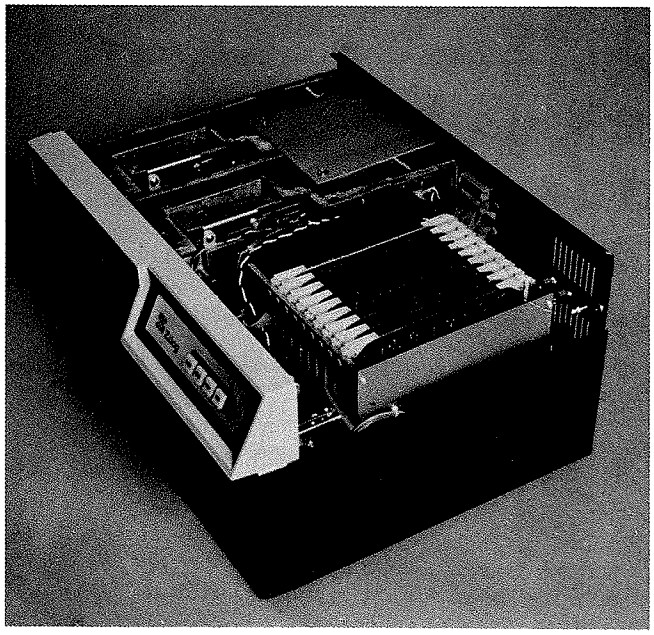
[13] Microprogram Development System. Datenblatt der Firma National Semiconductor, 1975.

[14] PDA-80 Systems Software Manual. Handbuch der Firma NEC Electronics, 1977.

[15] Operator's Manual for PPS-MP Assemulator. Handbuch der Firma Rockwell, 1975.

[16] Microcomputer Prototype Development System Twin. Produktbeschreibung der Firma Signetics/Valvo, November 1976.

[17] 990 Computer Family, Systems Handbook. Handbuch der Firma Texas Instrument, Mai 1976.



Das Zilog-Z80-Entwicklungssystem ist kompromißlos auf Floppy-Disk-Technik aufgebaut und verfügt über besonders leistungsfähige Echtzeit-Testhilfen. Dadurch wird die Entwicklung von Anwendersystemen in technisch minimal möglicher Zeit mit maximaler Kostenersparnis durchgeführt

Programme ohne Umwege erstellt und getestet

Die Hersteller von Mikroprozessoren bieten auch Programmiersysteme für jeden µP-Typ an. Die meisten dieser Systeme lassen sich per Assemblersprache ansprechen und benutzen einen Lochstreifen (oder ein anderes Speichermedium) zur Zwischenspeicherung des Programms, bevor es endgültig in ein PROM geladen wird. Ein System, das diesen Umweg vermeidet und vergleichsweise billig aufgebaut werden kann, hat die Münchner Firma Adcomp Datensysteme für den Eigenbedarf entwickelt. Inzwischen wurden damit innerhalb von drei Monaten 8 KByte an Programmen erstellt. Ein gewisser Nachteil liegt darin, daß direkt im Maschinen-Code programmiert wird, deshalb ist die Anwendung hauptsächlich bei kleineren und mittleren Programmlängen sinnvoll.

Forderungen

Die wesentlichen Forderungen für das Programmerstellen sind folgende Punkte:

- Einen Befehl soll man durch möglichst wenig Tastendrücke eingeben können, sonst nimmt das Suchen der Tasten zuviel Zeit in Anspruch. Der eingegebene Befehl soll, sofern es sich um eine Änderung handelt, sofort am Prototyp getestet werden können. Dazu muß das zu entwikkelnde System direkt und jederzeit Zugriff zu dem Speicher haben, in den die Befehle eingegeben werden.
- Der Speichervorrat darf nicht, wie bei einigen Systemen, schon bei 1 K enden. Eine sinnvolle Kapazität sind 4 oder besser 8 KByte, um auch komplexere Geräte programmieren zu können. Bei plötzlichem Stromausfall sollte man sicher sein, daß der Speicher nicht gelöscht wird. Das ist möglich mit einem Kassettengerät oder einer Floppy Disk, billiger jedoch kommt eine NC-Sinter-Batterie.
- Erstellte und bereits mit dem Programmierplatz getestete Programme müssen irgendwann in PROMs „eingefroren" werden. Üblicherweise stanzt man dazu einen Lochstreifen, der dann in ein PROM-Programmiergerät eingelesen wird. Wirtschaftlicher und schneller ist es, wenn der Programmierplatz die PROMs direkt ohne den Umweg über Streifen füllen kann.
- Zur Kontrolle der eingetasteten Adressen oder Daten muß eine optische Anzeige vorhanden sein. Billiger als ein Fernsehschirm sind alphanumerische LED-Anzeige-Einheiten.

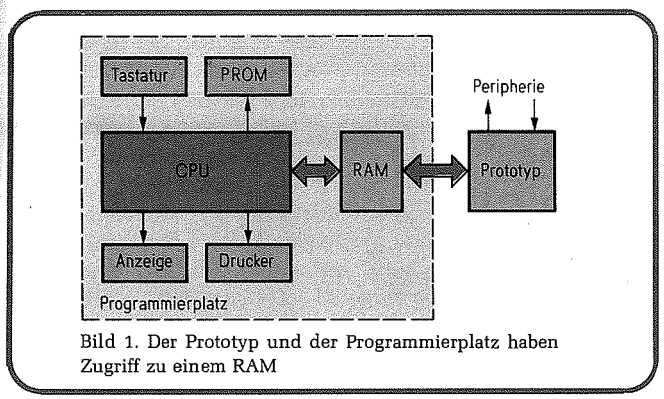


Bild 1. Der Prototyp und der Programmierplatz haben Zugriff zu einem RAM

– Für eine bessere Übersicht bei späteren Änderungen oder Erweiterungen soll man Programme ausdrucken können. Das ist keine Notwendigkeit, jedoch eine gewisse Vereinfachung. Ein Druckeranschluß für inzwischen preiswert gewordene Matrix-Drucker muß deshalb vorhanden sein.
– Erstellte Programme, die sich bereits in PROMs befinden, müssen ohne den Umweg über Lochstreifen wieder in das RAM geladen werden können. Damit können auch später schnell Änderungen an einem Mikroprozessor-System durchgeführt werden.

Arbeitsweise

Zwei Mikroprozessor-Systeme haben direkten Zugriff zu einem RAM: der Programmierplatz und das Prototypensystem (*Bild 1*). Das bedeutet, daß das RAM des Programmiersystems die späteren PROMs oder ROMs mit dem Anwenderprogramm ersetzt. Dazu liefert der Prototyp eine 13-bit-Adresse und erhält Befehle über den Datenbus. Eine einfache Adreß-Umschaltlogik regelt den Zugriff der beiden Systeme.

Die Tastatur des Programmierplatzes hat 16 Tasten von 0...F. Zwei Tastendrücke ergeben ein Byte. Die Adresse erhöht sich selbständig. Mit Steuertasten lassen sich folgende Funktionen auslösen:

1. Adressenänderung
2. Programmabschnitt drucken
3. PROM programmieren
4. PROM-Inhalt in RAM einlesen
5. Zur optischen Inhaltskontrolle Adresse vor- oder zurückzählen.

Das Display zeigt jeweils eine 2-Byte-Adresse auf vier Sedezimal[1])-Anzeigen und das im RAM gefundene Byte auf zwei weiteren Sedezimal-Anzeigen an. Nach dem Füllen wird die Adresse automatisch um eins erhöht, der Inhalt der nächsten RAM-Position ist dann sichtbar. Der Adreßbereich geht von 0000...1FFF.

1000	62	LU	2	
1001	6F	LL	7	
1002	70	LAU	0	0 → Stack
1003	5E	SAR	⟨ ⟩ –	
1004	8F	BR7		
1005	FE			
1006	6B	LL	3	→ 2
1007	20	LAL		
1008	99		99	
1009	5E	SAR	⟨ ⟩ –	
100A	5E	SAR	⟨ ⟩ –	fülle mit 99
100B	5E	SAR	⟨ ⟩ –	
100C	29	JMP		
100D	09		09	springe → fertig
100E	0F		0F	

Bild 2. Auf einem Programmierbogen werden (von links nach rechts) die Adresse, der Hexadezimal-Code für einen Befehl, der mnemonische Code und – wenn nötig – ein Kommentar eingetragen

[1]) Korrektere Bezeichnung für hexadezimal (vgl. Arbeitsblatt Nr. 103, H. 4/77).

Als PROMs werden die handelsüblichen Typen 2708 verwendet, deren Preis durch die Anzahl der Hersteller und durch die produzierte Menge inzwischen reduziert wurde und wohl den Typ 1702 A verdrängen wird.

Als RAMs werden die 1-Kbit-Typen 2102 verwendet, die inzwischen Industriestandard sind. Der Programmierplatz selbst hat noch ein einfaches Betriebssystem in drei PROMs (1702 A), also mit einem Speicherbedarf von weniger als 768 Byte.

Der Druckeranschluß arbeitet im Handshake-Verfahren, er paßt sich an die maximale Geschwindigkeit des Druckers an. Einsetzbar sind Drucker mit einer 7- oder 8-bit-Parallelschnittstelle.

Bedienung

Aufgrund des einfachen Aufbaus ergeben sich zwei Nachteile, die gewisse Voraussetzungen verlangen:

- Das zu programmierende Mikroprozessor-System muß eine Befehlsarchitektur haben, bei der sich Befehle in zwei Tetraden teilen lassen. Diese Voraussetzungen bietet z. B. der Typ F 8. Andere Systeme erfüllen diese Voraussetzung weitgehend.
- Der Programmierer muß eine ganze Reihe von Befehlen auswendig lernen. Empfehlenswert ist es, eine Tabelle aller vorkommenden Befehle auf einem Blatt zu benutzen, um das dauernde Wälzen von Handbüchern zu vermeiden. Geht man einen Schritt weiter, dann sollte man sich sogar von mnemonischen Befehlen lösen, die mehr als drei Buchstaben haben. Dieser Code besteht aus der Kurzerklärung eines Befehls, z. B. CLA (*lösche Akkumulator*) oder LAR (*lade Akkumulator von Register*). Da diese Befehle nicht eingetastet, sondern in einem Programmierbogen lediglich mitgeschrieben werden, kann man die vom Hersteller angegebenen als Empfehlung betrachten und kürzen. Erfahrungsgemäß beherrscht man schon nach den ersten 100 Bytes die entsprechende Sedezimal-Schreibweise für die meisten Befehle. Nach 1000 Bytes wird die Befehlsübersicht nur noch selten zur Hand genommen.

Programmiert wird nicht an der Maschine, sondern mit dem Bleistift. Ein Programmierbogen (*Bild 2*) enthält am linken Rand fortlaufende Adressen. In die übernächste Spalte werden die (gekürzten) mnemonischen Befehle eingetragen, daneben gegebenenfalls ein Kommentar. Bei dieser Arbeitsweise hat man den Ablauf ständig vor Augen. Im Sinne einer guten „Buchführung" werden außerdem die Befehle im Sedezimal-Code eingetragen. Nachdem ein Abschnitt so programmiert ist, wird er über die Tastatur in das RAM getastet. Sofort kann das Prototypensystem die neuen Befehle ausführen – der Programmtest läuft.

Erfahrungen

Mit dem System wurden inzwischen ein MDT-Computer mit 4 KByte Betriebssystem, eine Waage mit 2,5 KByte und ein medizinisches Gerät mit 1 KByte Anwenderprogramm programmiert. Ein vergleichbares System, allerdings mit komfortablem Assembler und Kassettensystem, kostet etwa 55 000 DM. Das beschriebene Gerät ist bereits für etwa 13 000 DM (ohne Drucker) zu erstellen.

Helmut Holighaus

Dipl.-Ing. Jochen Müller

Logik-Analyse im Datenbereich

Grundlagen – Möglichkeiten – µP-Anwendungen

Die Entwicklung der Digitaltechnik in den letzten zehn Jahren war zwar atemberaubend, die Einführung der Mikroprozessoren jedoch verlief im Vergleich dazu explosionsartig. Mit dem rasanten Anwachsen der Anwendungsmöglichkeiten des Mikroprozessors verbreitete sich die Kunde von diesem neuen Digital-Baustein. Zwangsläufig mußte auch die Meßtechnik neue Wege einschlagen: Es gilt nun, die Dimension des Datenbereiches zu bewältigen. Als ein unentbehrliches Hilfsmittel erweist sich hier der Logik-Analysator; die folgenden Ausführungen geben Einblick in seine vielfältigen Verwendungsmöglichkeiten.

1 Mikroprozessoren erfordern eine neue Meßtechnik

Die neue Technik hat die Entwicklung neuartiger Meßgeräte und Meßverfahren nach sich gezogen. Sowohl der Entwicklungs- als auch der Testingenieur hatten bisher überwiegend Hardware-Probleme mit herkömmlichen Meßgeräten zu lösen, während jetzt in der *Software* eine andere Problemart auf sie zukommt.

Software ist, vereinfacht gesagt, der Inhalt eines Festwertspeichers, z. B. eines ROM. Der Speicherinhalt bestimmt die Aufeinanderfolge von elektrischen Signalen, wobei der zeitliche Ablauf dabei nicht kontinuierlich, wie beispielsweise die Strahlablenkung eines Oszilloskops, sondern schritt- oder taktweise wie bei einer Digitaluhr ist. Dieser Vorgang wird dadurch unübersichtlich, daß mit jedem Schritt viele gleichzeitig im System vorhandene Signale bestimmte logische Pegel aufweisen müssen.

Der im Speicherinhalt festgelegte Programmablauf, der über die Zentraleinheit (CPU) sämtliche Funktionen steuert, wäre recht einfach zu überprüfen, gäbe es nicht die Bedingungsverknüpfungen, die eine Systemfunktion zwar komplex, dafür aber flexibel gestalten. Die Verknüpfung kann beispielsweise so aussehen, daß in einem Programmschritt mehrere Signalleitungen auf ihren logischen Zustand („1" oder „0") abgefragt werden. Abhängig von der gelesenen Kombination wird dann an eine andere Stelle im Programm gesprungen. Es ist somit verständlich, daß die Abweichung eines einzigen Signales zu einem beliebigen Zeitpunkt katastrophale Folgen für das Funktionieren eines Mikroprozessor-Systems haben kann.

Der Programmablauf wird auch durch Parameter bestimmt, die beispielsweise vom Bediener manuell ausgelöst oder automatisch von Meßeinrichtungen (als Meßdaten) geliefert werden. Der Vorteil eines „Software-orientierten Systems" besteht darin, daß durch Auswechseln des Festwertspeichers dasselbe System einen anderen Programmablauf haben kann: Die Software kann je nach Anwendungsbedarf für die gleiche Hardware entwickelt, und durch Austausch der Speicher verändert werden.

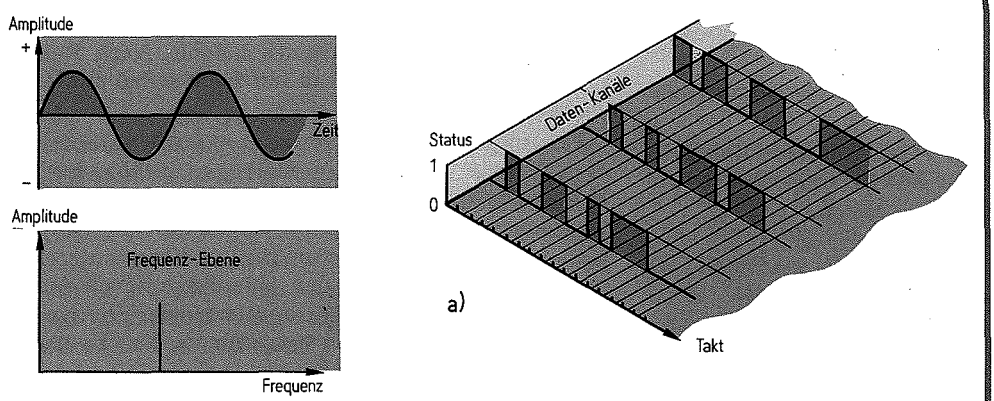


Bild 1. Grafische Darstellung einer Sinusschwingung in der Zeit- bzw. Frequenzebene

Bild 2a. Drei Datenkanäle stellen den räumlichen Zusammenhang zwischen Takt- und Statusebene (Übergang zum Datenbereich) dar

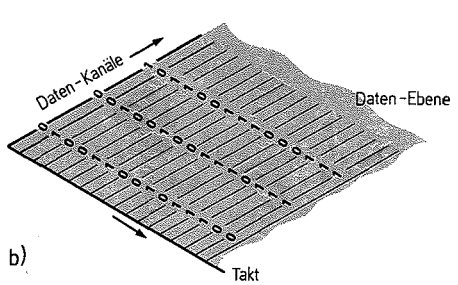


Bild 2b. Reduktion der räumlichen Darstellung in eine zweidimensionale tabellarische (ähnlich der Anzeige eines Logik-Analysators)

Zum Entwickeln von Programmen müssen die vom Hersteller des Mikroprozessors festgelegten „Spielregeln" eingehalten werden. Dazu gibt es Hilfen, die sogenannten Software-Entwicklungs-Systeme, die Eingriffe in das Programm

erlauben: Programm-Instruktionen oder Speicher-Inhalt können leicht geändert werden, da der Festwertspeicher hier noch durch einen Schreib-Lese-Speicher ersetzt ist. Ist ein Programm erstellt, folgt der Probelauf. Dieser kann mit erweiterten Entwicklungs-Systemen, den sogenannten Simulatoren oder Emulatoren, oder mit einer neuen Generation von Meßgeräten, den Logik- und/oder Mikroprozessor-Analysatoren, durchgeführt werden. Die Analysatoren ermöglichen Realzeit-Darstellung, was bedeutet, daß das Programm mit der geplanten Geschwindigkeit ablaufen kann. Ein Teil davon wird in einen Speicher gelesen und auf einer Elektronenstrahlröhre lange genug dargestellt, um die Anzeige interpretieren und den Programmablauf damit analysieren zu können.

2 Der Datenbereich, eine neue Dimension in der Meßtechnik

Die Definition des Datenbereiches stützt sich auf die beiden bekannten Bereiche, den Zeit- und Frequenzbereich [15]. Im *Zeitbereich* wird ein elektrisches Signal als Spannung in Abhängigkeit von der Zeit dargestellt (Oszilloskop-Darstellung). Messungen im Zeitbereich betreffen also die analogen Parameter elektrischer Signale.

Im *Frequenzbereich* wird das Leistungsspektrum eines elektrischen Signals dargestellt, wie es vom Spektrum-Analysator her bekannt ist. Vertikal ist die Leistung (bzw. die Amplitude), horizontal die Frequenz wiedergegeben. Messungen im Frequenzbereich enthüllen die Frequenz-Komponenten eines Signals, seinen Unterwellen- und Oberwellen-Gehalt, als Spektrallinien angezeigt.

Den Zusammenhang zwischen Zeit- und Frequenzbereich zeigt *Bild 1*. In einem kartesischen Koordinaten-System ist eine Sinuswelle dargestellt, und sie erscheint auf der Amplituden-Frequenz-Ebene als senkrechte Linie unter der Bedingung, daß sie keine Oberwellen beinhaltet.

In *Bild 2* wird die räumliche Darstellung von Impulsfolgen auf drei parallelen Kanälen gezeigt. Die Amplituden-Achse wurde zur *Status-Achse*, da im Datenbereich nur die beiden logischen Zustände „0" und „1" interessieren. Die Zeit-Achse wurde zur *Takt-Achse*, weil nur noch diskrete Zeitintervalle für den funktionellen Ablauf relevant sind. Darunter ist die räumliche Darstellung auf eine zweidimensionale Ebene übertragen; diese zweidimensionale Darstellung entspricht der binären Zustands-Anzeige von Logik-Zustands-Analysatoren.

Messungen im Datenbereich erfassen also gleichzeitig vorhandene Zustände auf mehreren Signalleitungen, den zeitlichen Ablauf von Zuständen oder Zustands-Änderungen, oder allgemein den funktionellen Datenablauf.

3 Aufgaben von Entwicklungs-Systemen und Logik-Analysatoren

Entwicklungs-Systeme und Logik-Analysatoren haben unterschiedliche Aufgaben; es gibt auch Überlappungen in den Einsatzmöglichkeiten. *Bild 3* soll am Beispiel eines Entwicklungsablaufs einer Mikroprozessor-Schaltung zeigen, zu welcher Zeit und an welchem Ort welches Gerät oder System zweckmäßig eingesetzt wird.

Die linke Seite dieses Flußdiagramms zeigt die Software-, die rechte Seite die Hardware-Entwicklung. Dieser Ablauf entspricht zwar den meisten Entwicklungsaufgaben, doch ist der gerätetechnische Aufwand sehr unterschiedlich. Kleinere Programme (< 2000 Worte) rechtfertigen nicht die Investition eines Software-Entwicklungs-Systems. Beim

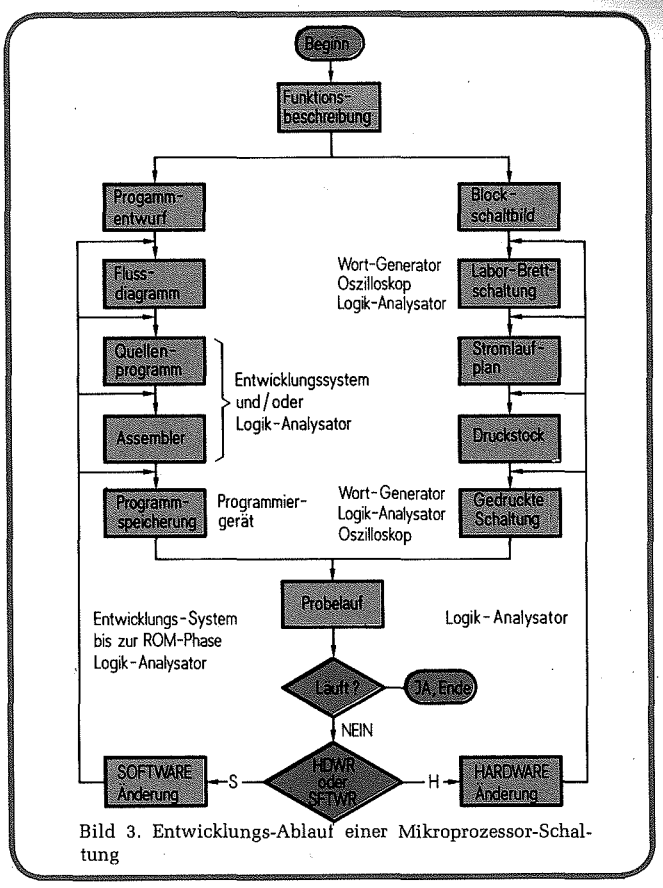
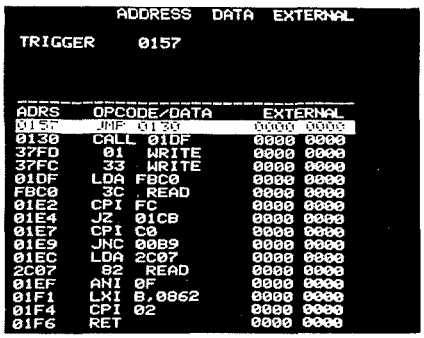


Bild 3. Entwicklungs-Ablauf einer Mikroprozessor-Schaltung

Einsatz einfacher Mikroprozessoren für kommerzielle Anwendungen ist die Beschaffung eines meist recht teuren *Entwicklungs-Systems* ebenfalls wirtschaftlich abzuwägen. Dort kann der Einsatz eines Programmier-Gerätes und eines Logik-Zustands-Analysators ausreichend sein. Im Gegensatz dazu wird die Software-Entwicklung bei größeren Programmen (> 4000 Worte) ohne Entwicklungs-System hoffnungslos. Derartige Systeme reichen von der *Firmware* (Unterprogramm, um geschriebenes Programm zu speichern, auszudrucken und zu starten) bis zum *Compiler* (Übersetzer-Programm auf hoher Ebene), dessen Ergebnis ein Programm in Maschinensprache ist, somit verständlich für den Mikroprozessor bzw. das Programmiergerät.

Die Hardware-Entwicklung verwendet sowohl das „klassische" Oszilloskop als auch Logik-Analysatoren. Vor dem Probelauf dient als Datenquelle beispielsweise ein *Wortgenerator*, dessen Wiederholrate kontinuierlich verändert werden kann, was zum Bestimmen der im System maximal möglichen Programmablauf-Geschwindigkeit von Vorteil ist. Außerdem lassen sich die Datenausgänge des Wortgenerators zeitlich gegeneinander und gegen den Taktausgang in beide Richtungen (Vorlauf, Nachlauf) kontinuierlich steuern, um Zeitprobleme zu simulieren, so daß kritische Grenzen im frühen Hardware-Entwicklungs-Stadium erkennbar werden.

Wenn Hardware und Software funktionieren, findet der schon erwähnte *Probelauf* statt. In diesem Entwicklungs-Stadium ist die Unterscheidung zwischen Hardware- und Software-Fehlern schwierig, da beide zunächst als funktionelle Fehler angezeigt werden. Die Anwendung des Software-Entwicklungs-Systems zur Fehlersuche ist bis zur ROM-Phase geeignet. Nach dieser Phase, wenn also das Programm vom Simulator in einen Festwertspeicher übernommen wurde, ist ein Eingriff in den Programmablauf (also direkt in



▲ Bild 4. Logik-Zustands-Analysator Hewlett-Packard 1600 S

◀ Bild 5. Mnemonic-Anzeige des Logik-Analysators HP 1611 A

die Software) nur noch bedingt möglich. Von diesem Zeitpunkt an wird die Software- und Hardware-Analyse dynamisch mit Mikroprozessor- oder *Logik-Analysatoren* durchgeführt. Bei einem Hardware-Fehler (Verdrahtung, Zeitprobleme von elektrischen Signalen, Störimpulse, Bauteile-Fehler; Störimpulse und Zeitprobleme sind die häufigsten Fehlerursachen) lokalisiert der Analysator zwar den Fehler, die elektrische Ursache hierfür zeigt jedoch oft erst das vom Analysator synchronisierte Oszilloskop.

Elektrische Fehler stellen sich auf dem Analysator wie Software-Fehler dar. Dynamische oder intermittierende Fehler müssen erst lokalisiert werden, bevor eine Realzeit-Darstellung möglich ist.

Das Beispiel des Entwicklungsablaufes einer Mikroprozessor-Schaltung zeigt, daß Entwicklungs-Systeme und Analysatoren sich zeitlich und funktionell eher ergänzen als verdrängen.

4 Logik-Analysatoren und ihre Anwendung im Datenbereich

4.1 Unterscheidungsmerkmale und Auswahlkriterien

Das neue Meßkonzept ist durch die schon erwähnte neue Geräteklasse der Logik-Analysatoren verwirklicht; die Namensgebung reicht vom „Digiscope" über „Digital-Analysator" und „Logik-Scope" bis zum „Mikroprozessor-Analysator". Das Prinzip, Daten digital zu erfassen und auf einem Bildschirm anzuzeigen, ist allen Ausführungen gemeinsam. In der Verwirklichung, besonders im Preis, der zwischen 1000 DM und 70 000 DM liegt, unterscheiden sich die Geräte jedoch sehr stark.

Unter den Analysatoren unterscheidet man drei Gruppen, die Logik-Zeit-Analysatoren (*Logic Timing Analyzer*), die Logik-Zustands-Analysatoren (*Logic State Analyzer*) und die Mikroprozessor-Analysatoren.

- Die *Logik-Zeit-Analysatoren* arbeiten mit einer intern variablen Taktrate bis zu 200 MHz. Sie tasten die binären Signale ähnlich dem Sampling-Verfahren ab, speichern die Daten und rufen dann den Speicherinhalt zur Anzeige ab. Die Anzeigeform ist eine Pseudo-Puls-Darstellung, d. h., es gibt nur die beiden Pegel „0" und „1" (bzw. „L" und „H") und die dazwischenliegenden Übergangsflanken, jedoch ist dies keine analoge Impuls-Darstellung im Sinne eines herkömmlichen Oszillogramms.

  Der wesentliche Vorteil der Logik-Zeit-Analysatoren gegenüber dem Oszilloskop besteht in ihrer Fähigkeit, mehrere parallel verlaufende Ereignisse gleichzeitig darstellen zu können. Zwei- oder Mehrkanal-Oszilloskope verwenden entweder die nacheinander folgende Strahlablenkung (alternierend) oder die zerhackte Ablenkung (Chop-Betriebsart). Die Zeitauflösung des Logik-Zeit-Analysators hängt von seiner internen maximalen Taktrate ab. Um parallele Ereignisse hinreichend aufzulösen, muß die Abtastfrequenz des Logik-Zeit-Analysators um mindestens den Faktor 10 größer als die höchste Signalfrequenz gewählt werden; es lassen sich auch kurze Impulse erkennen und speichern.
- Die *Logik-Zustands-Analysatoren* erfassen die Daten nach dem gleichen Verfahren wie die Logik-Zeit-Analysatoren, sie verwenden jedoch einen datensynchronen Takt, also den Takt des Systems, das die Daten erzeugt. Der Speicherinhalt wird bei diesen Geräten als „0" und „1" auf dem Bildschirm dargestellt (Binäre Zustandsanzeige; *Bild 4*). Zur Verfolgung von Computer- oder Mikroprozessor-Programmen ist diese Darstellungsweise sehr gut geeignet, weil die Anzeige der Maschinensprache des Systems entspricht.
- Die *Mikroprozessor-Analysatoren* sind Meßgeräte, die als spezielle, auf Mikroprozessoren zugeschnittene Logik-Zustands-Analysatoren bezeichnet werden können. Sie lassen meist die Vielseitigkeit der Zustands-Analysatoren im Hinblick auf allgemeine digitale Logik vermissen, sind aber andererseits komfortabler für die Untersuchung von Mikroprozessor-Programmabläufen ausgestattet. Der Analysator Hewlett-Packard Modell 1611A assembliert die Meßwerte rückwärts (Umsetzung vom Maschinencode in Assemblersprache) und stellt sie als Befehlsfolge in Mnemonics auf dem Bildschirm dar (*Bild 5*).

4.2 Eigenschaften und praktische Anwendungen

Die erste Anforderung an Meßgeräte im Datenbereich ist eine Vielzahl von Eingangskanälen. Die heutigen 8-bit-Mikroprozessoren bestimmen das absolute Minimum, nämlich acht Kanäle. Mit diesen lassen sich dann aber nur die Datenleitungen (*Data Bus*) beobachten. Wichtiger für die Verfolgung eines Programmablaufs ist die Anzeige des Programmzählers oder des Adressen-Register-Zählers, der meist 16 bit (parallel) hat. Geräte, die besonders für die Fehlersuche in Mikroprozessorschaltungen konzipiert sind, haben 25 parallele Eingänge, wie das Biomation-Modell 168-D, und sogar 32 Eingangskanäle, wie die Modelle Hewlett-Packard 1600S und 1611A (Bild 4 und 5).

Der Vorteil einer großen Anzahl von Eingängen sei am Beispiel in *Bild 6* erläutert; es zeigt die Minimalkonfiguration eines Motorola-Mikroprozessor-Systems. Die Bildschirmdarstellung der Adressen (Adreß-Bus $A_0...A_{15}$), Daten oder Instruktionen (Daten-Bus $D_0...D_7$) und der Steuersignale (VMA, R/W, CB 2) ist in *Bild 7* wiedergegeben. Es ist leicht zu erkennen, daß nur der Überblick über eine Vielzahl von Signalen eine genaue Beurteilung des Systems erlaubt.

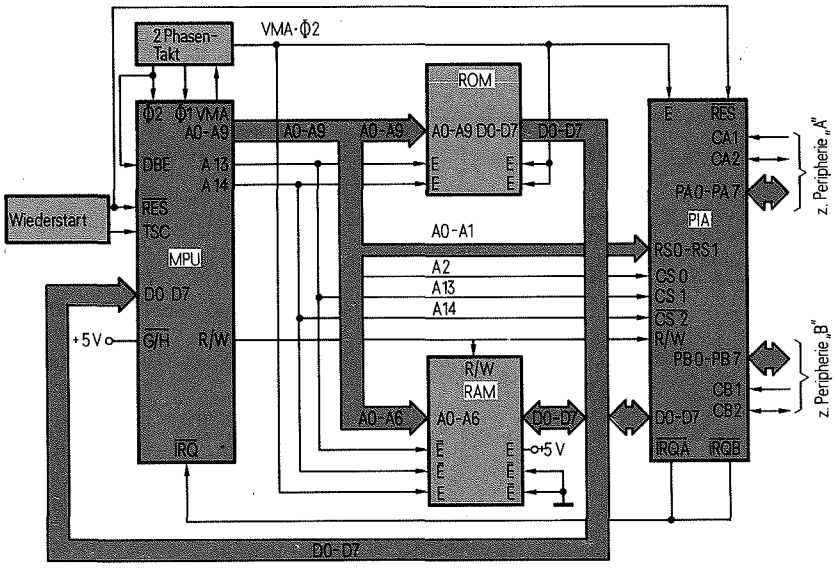


Bild 6. Blockschaltung eines Mikroprozessor-Systems Motorola MC 6800

Ein anderes wesentliches Kriterium für Logik-Analysatoren ist ihre Speicherfähigkeit. Logik-Zeit-Analysatoren brauchen eine beträchtliche Zahl von Speicherzellen; Logik-Zustands-Analysatoren haben eine geringere Speichergröße, um die Wiederholrate für die Datenerfassung erhöhen zu können und damit der Realzeit-Darstellung von Programmabläufen näher zu kommen. Dieser Nachteil gegenüber den Zeit-Analysatoren wird durch selektive Datenspeicherung ausgeglichen: Um beispielsweise aus einem 1000 Schritte umfassenden Programmteil die ungleichmäßig auftretenden Schreibbefehle analysieren zu können, verwendet man den Zustand der Schreib-Lese-Steuerleitung als Auswahl-Kriterium zur Datenspeicherung. Als Ergebnis dessen werden nur solche Daten in den internen Speicher des Analysators eingelesen, die gleichzeitig auf der Schreib-Lese-Steuerleitung die Schreibbedingung erfüllen.

Eine notwendige Eigenschaft der Analysatoren ist die kombinatorische oder Boole'sche Triggerung: Eine vorgegebene Kombination von gleichzeitig auftretenden Bits ergibt eine eindeutige Triggerreferenz für das Gerät. Die Triggerworterkennung ermöglicht die Anzeige von Daten, die mit dem Triggerwort enden bzw. anfangen. Die erste Anzeigeform ist die Pre-Trigger- oder End-Darstellung, die zweite die Post-Trigger- oder Start-Darstellung, welche der Oszilloskop-Triggerung entspricht. Die Pre-Trigger-Darstellung läßt sich nur mit einem digitalen Speicher verwirklichen (*Negative-Zeit-Darstellung*). Die eindeutige Triggerreferenz ist notwendig für die digitale Verzögerung. Sie erlaubt es, das begrenzte Anzeigefenster (Bild 4) um beliebige Taktzyklen über den Datenablauf zu verschieben, ähnlich der analogen Verzögerung oder zweiten Zeitbasis des Oszilloskops. Die Kombination von End-Darstellung und digitaler Verzögerung ermöglicht es schließlich, das vorgewählte Triggerwort irgendwo auf dem Bildschirm erscheinen zu lassen. Darüber befinden sich zeitlich vorhergehende, darunter zeitlich nachfolgende Ereignisse.

Die Schnelligkeit der Analysatoren ist meist hinreichend dimensioniert; die oberen Grenzfrequenzen liegen zwischen 5 und 200 MHz. Die asynchronen Logik-Zeit-Analysatoren arbeiten wegen der höheren zeitlichen Auflösung schneller als die Systemtakte. Logik-Zustands-Analysatoren, die 16 und mehr Kanäle speichern und anzeigen, arbeiten bis 20 MHz; Logik-Analysatoren für Mikroprozessoren sind in der oberen Grenzfrequenz den schnellsten Mikroprozessor-Taktfrequenzen angepaßt.

Ein weiterer Faktor ist die mechanische und elektrische Anpassung der Signalaufnehmer an die zu untersuchende Schaltung, wobei die elektrische Schnittstelle kritischer ist. Bei CMOS-Bausteinen muß die Impedanz der Tastköpfe sehr hoch (> 1 MΩ), und die Parallelkapazität sehr niedrig (< 15 pF) sein. Es gilt also auch hier das klassische Prinzip, daß ein Meßgerät das Meßobjekt möglichst nicht beeinflussen soll.

Da das elektrische Signal digitalisiert wird, muß der Eingang des Analysators einen *Komparator* besitzen, der den logischen Pegel des Signals erkennt. Eine Triggerschwelle (fest oder variabel) dient hierbei als Referenz. Zwei Triggerschwellen pro Eingang erlauben die Anzeige von Pegeleinbrüchen (Störpulse).

Die meisten Analysatoren besitzen *Synchronisations-Ausgänge*, die ein Referenzsignal für die externe Triggerung eines Oszilloskops anbieten; man kann dies Synchronisation des Zeitbereichs mit dem Datenbereich nennen.

Wichtig im Datenbereich ist der Vergleich von Ist-Daten mit Soll-Daten, was zwei Speicher erforderlich macht: Ein digitaler Komparator führt den Vergleich von gemessenen Daten mit vorgegebenen Referenz-Daten durch. Das Ergebnis wird bei Übereinstimmung als „0", bei Fehler als strahlintensivierte „1" an der zugehörigen Stelle angezeigt.

Auch das Anzeigebild soll als Auswahl-Kriterium herangezogen werden: Die *Leuchtdiodenanzeigen* sind wenig aufwendig, aber auch wenig flexibel, da nur eine binäre Anzeige möglich ist. Eine *Elektronenstrahlröhre* ermöglicht alle wünschenswerten Anzeigeformen, die von der Pseudo-Impuls-Anzeige bis zur mnemomischen Programmwiedergabe reichen. Die meisten Analysatoren verwenden eingebaute Röhren; die übrigen besitzen Signalausgänge, die entweder an ein Oszilloskop oder an ein Sichtgerät zur Datenanzeige angeschlossen werden.

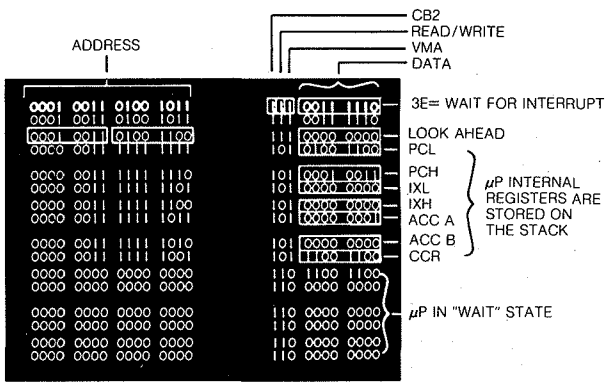
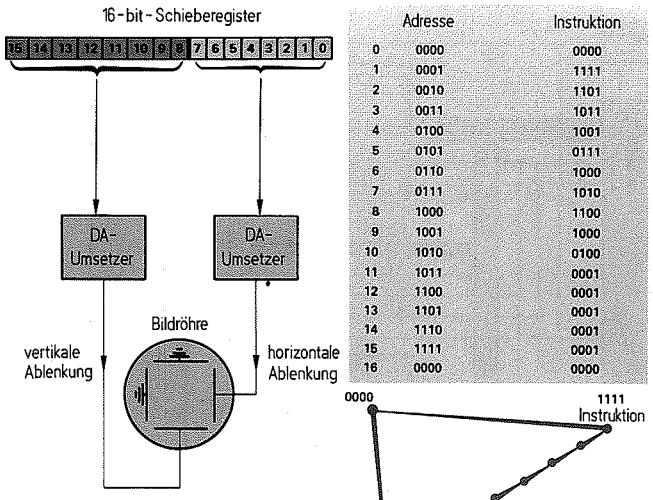


Bild 7. Anzeige des HP-Modells 1600 A bei einer Messung in der Schaltung nach Bild 6



	Adresse	Instruktion
0	0000	0000
1	0001	1111
2	0010	1101
3	0011	1011
4	0100	1001
5	0101	0111
6	0110	1000
7	0111	1010
8	1000	1100
9	1001	1000
10	1010	0100
11	1011	0001
12	1100	0001
13	1101	0001
14	1110	0001
15	1111	0001
16	0000	0000

▲ Bild 8. Prinzip der Karten-Darstellung (*mapping*)

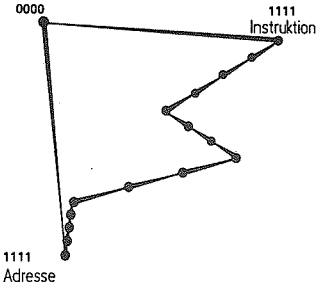


Bild 9. ▶ Karten-Darstellung einer Programmfolge

Die Darstellung im Sedezimal-Code (bekannt als Hexadezimal-Code) ist für die Anzeige von Mikroprozessor-Programmen bequemer, da die Instruktionen vom Hersteller so im Datenblatt angegeben werden. Daher bieten viele Analysatoren die Wahl von Oktal- und Sedezimal-Code an.

Eine neuere Anzeigeform ist die *Speicher-Darstellung* (Biomation Modell 168D). Aus einem Programm werden 256 Schritte gespeichert und in verschiedenen Formen dargestellt: Die gespeicherten Adressen (16 bit parallel) werden in die höherwertigen 8-bit und niederwertigen 8-bit unterteilt. Erstere werden als Seiten-Adressen in Matrix-Form (8x8) dargestellt; letztere als Seiteninhalt in der gleichen Form.

Die dritte Anzeigeart stellt den Inhalt des internen Speichers als Schreib- oder Lese-Befehl ebenfalls in *Matrix-Form* dar. Eine sehr gute alphanumerische Anzeige ist die des Logik-Analysators HP 1611A, der im mnemonischer Darstellung den untersuchten Programmteil anzeigt. Der interne Speicherinhalt kann auch im Oktal- oder Sedezimal-Code dargestellt werden.

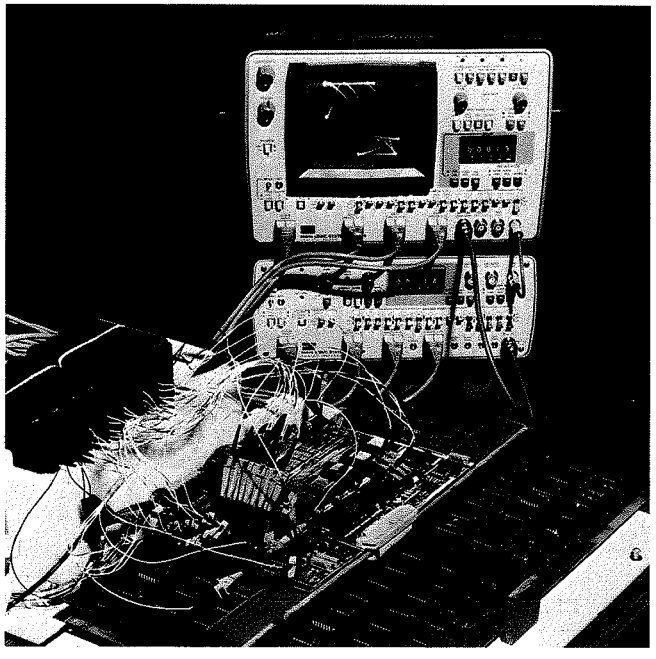


Bild 10. Karten-Darstellung des HP 1600 S in der Praxis

Die bisher beschriebenen Anzeige-Arten sind *quantitative* Darstellungen: Man kann konkrete Werte ablesen. Die aus den Anforderungen im Datenbereich entstandene grafische Anzeigeform, bestehend aus Punkten und Verbindungslinien, stellt Meßgrößen *qualitativ* dar; dies erlaubt eine Gesamt-Übersicht über Programmaktivitäten. Die quantitative Anzeigeform ermöglicht eine definierte Aussage über eine begrenzte Datenmenge. Das Prinzip der grafischen Darstellung (auch *Karten-* oder *mapping-Darstellung*) besteht in der Digital-Analog-Umsetzung von 16-bit-Worten. *Bild 8* zeigt ein Datenregister, in das jedes Programmwort eingelesen wird. Entsprechend der Bit-Konfiguration des Datenregister-Inhalts werden zwei analoge Ablenkspannungen erzeugt. Jeder 16-bit-Kombination entspricht eine bestimmte Lage des zugehörigen Punktes auf dem Bildschirm.

Bild 9 zeigt das Beispiel einer Programmschleife mit 16 Worten zu 8 bit. Nachdem ein Wort in einen Punkt umgewandelt worden ist, wird nicht nur das folgende Wort als neuer Punkt, sondern auch eine Verbindungslinie zwischen beiden dargestellt, die sich zum zeitlich später abgebildeten Punkt hin verbreitert. Diese Vektordarstellung dient dazu, die zeitliche Folge der Punkte (Programmschritte) zu erkennen. Die grafische Zustands-Darstellung ist also ein wesentliches Hilfsmittel bei der Fehleranalyse von Programmabläufen (*Bild 10*). Im Prüffeld können selbst angelernte Kräfte einen fehlerhaften Ablauf bei dieser Darstellungsweise sofort erkennen.

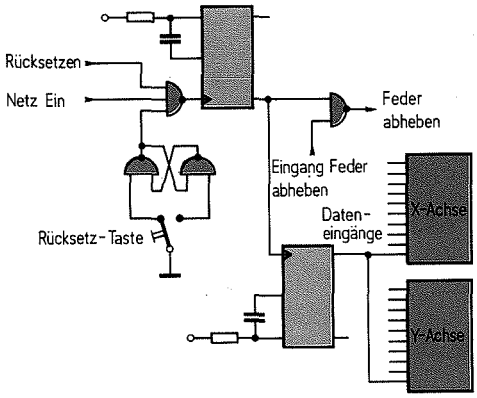
5 Logik-Analysatoren bei der Fehlersuche in Mikroprozessor-Schaltungen

5.1 Vergleich zur herkömmlichen Fehlersuche

Die Entwicklung vom einfachen Logik-Tester hin zum Logik-Analysator wird am Beispiel der Steuerschaltung eines Kurvenschreibers (Plotter) deutlich. Der gleiche Kurvenschreiber arbeitet in *Bild 11* mit diskreter digitaler Logik, in *Bild 12* mit einem Mikroprozessor.

Bild 11 zeigt den Teil der Logik des Kurvenschreibers, der, von der Rücksetz-Taste gesteuert, die Schreibfeder zum Abheben und in die Ausgangsposition bringen soll. Das Signal „Rücksetzen" kann von Knoten zu Knoten mit einem Logik-Tastkopf oder mit einem Oszilloskop verfolgt werden. Dieser Fall zeigt zwei wesentliche Merkmale: Erstens ist das Signal von der Rücksetz-Taste, auf Ort und Zeit bezogen, isoliert. Zweitens ist das Schaltbild für jemand, der mit Digital-Logik vertraut ist, hinreichend, um die Funktion zu verstehen. Das heißt, das Schaltbild und die Daten eines IC lassen die Funktion eines kombinatorischen logischen Systems erkennen.

Bild 12 zeigt die Blockschaltung des gleichen Kurvenschreibers, der hier jedoch einen Mikroprozessor enthält. Nun ist es nicht mehr möglich, ein einzelnes Signal von der Rücksetz-Taste bis zum X-Achsen- und Y-Achsen-Register zu verfolgen. Auch sagt das Schaltbild nichts über den Rücksetz-Vorgang. *Es gibt für niemanden, selbst wenn er mit Logik und Mikroprozessoren vertraut ist, eine Chance, über das Verhalten der Schaltung etwas auszusagen.* Die fehlende Information wird mit dem Flußdiagramm, *Bild 13*, gegeben. Für eine detaillierte Aussage, warum der Kurvenschreiber beispielsweise nicht auf die Rücksetz-Taste reagiert, muß nun der Programmablauf verfolgt werden.



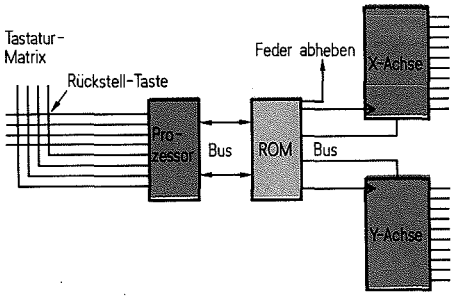


Bild 12. Prinzip eines Kurvenschreibers mit Mikroprozessor

◀ Bild 11. Blockschaltung eines Kurvenschreibers (*Plotter*)

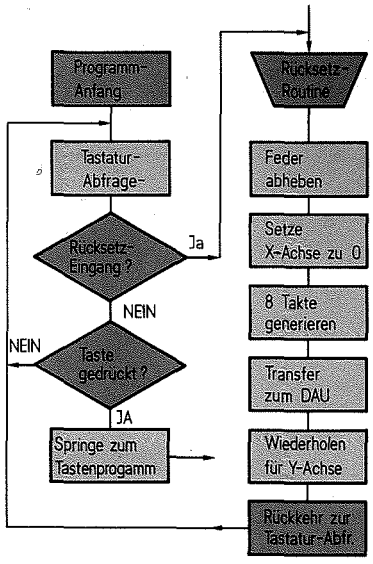


Bild 13. ▶ Flußdiagramm zum mikroprozessor-gesteuerten Kurvenschreiber

Zur Lösung solcher Probleme sind Logik-Zustands-Analysatoren entwickelt worden. Mit ihnen lassen sich Programmfehler zunächst lokalisieren. Danach können herkömmliche Geräte, wie Logik-Tastkopf und Oszilloskop, durchaus geeignet sein, die elektrische Ursache des Fehlers zu finden.

5.2 *Darstellung von Programmabläufen*

Jedes Programm kann in zwei Gruppen untergliedert werden, von denen die eine aus geradlinigen Programmabläufen, die andere aus Schleifen besteht. Zu den Schleifen kann man in diesem Sinne auch die Unterprogramme (Subroutinen) zählen, selbst wenn sie nur einmalig durchlaufen werden. Da jedes laufende Programm sich zumindest eine Zeitlang in einer Schleife befindet, läßt die grafische Darstellung sofort erkennen, was das Programm augenblicklich macht; eine sich wiederholende Schleife ergibt hierbei ein konstantes Polygon auf dem Bildschirm (Bild 10).

Da die Karten-Darstellung nur eine qualitative Aussage bietet, wird mit Hilfe eines Zeigers optisch ein interessierender Teil ausgewählt, der darauf in der quantitativen Darstellung zur genauen Analyse betrachtet werden kann. Dieses Verfahren, nämlich vom allgemeinen Programmüberblick auf einen eingeengten Bereich zu gelangen, bietet sich an, wenn der Anwender am Systemverhalten einen Fehler erkennt, ihn aber keinem Programmteil zuordnen kann.

5.3 *Bestimmen der Zugriffszeit eines ROM-Systems*

Ein ROM-System, bestehend aus vier ROMs, einem Adreßbus-Puffer, einem Datenbus-Puffer und einer ROM-Anwahl-Logik soll mit der maximalen Wiederholrate arbeiten, bei der die Daten noch fehlerfrei ausgelesen werden. Die maximale Wiederholrate ist nicht nur eine Funktion der ROM-Zugriffszeit, sondern auch des Zeitverhaltens aller logischen Bauelemente.

In *Bild 14* simuliert ein Wortgenerator den Adreßbus einer Zentraleinheit (CPU). Der Logik-Analysator zeigt den Signalfluß auf dem Datenbus des ROM-Systems an. Bei niedriger Wiederholrate der ROM-Adressierung werden gültige Daten ausgelesen und in den Sollwertspeicher des Analysators als Referenz-Daten eingelesen. Der Vergleich mit den Ist-Daten, die dann mit wachsender Wiederholrate (steigender Frequenz) in den Ist-Daten-Speicher eingelesen werden, zeigt nach Überschreiten der maximalen Rate einen oder mehrere Fehler (*Bild 15*). Das Zeitintervall der maximalen Wiederholrate ist die *maximale Zugriffszeit* des in Bild 14 abgebildeten Systems.

5.4 *Gleichzeitige Darstellung der Software- und Peripherie-Aktivitäten*

Ein Mikroprozessor-System arbeitet immer mit einer Schnittstelle, über die Daten in zwei Richtungen übertragen werden. *Schnittstellen verursachen die häufigsten Probleme in Mikroprozessor-Schaltungen.* Nachteilig ist, daß es zwei Datenabläufe gibt, die gleichzeitig beobachtet werden müssen; sie können unabhängige Taktraten haben oder sogar asynchron verlaufen. Um sie gleichzeitig darzustellen, muß eine gemeinsame Triggerreferenz vorhanden sein. Eine mögliche Lösung für die Darstellung solcher asynchronen Vorgänge zeigt *Bild 16.*

Die linke Seite der Anzeige stellt die Software (Adressen und Instruktionen) dar; auf der rechten Seite werden Peripherie-Signale dargestellt, die etwa achtmal langsamer sind als die Software-Signale. Das einzige gemeinsame Kriterium für die Darstellung der asynchronen Vorgänge besteht im Start der Anzeige links und rechts.

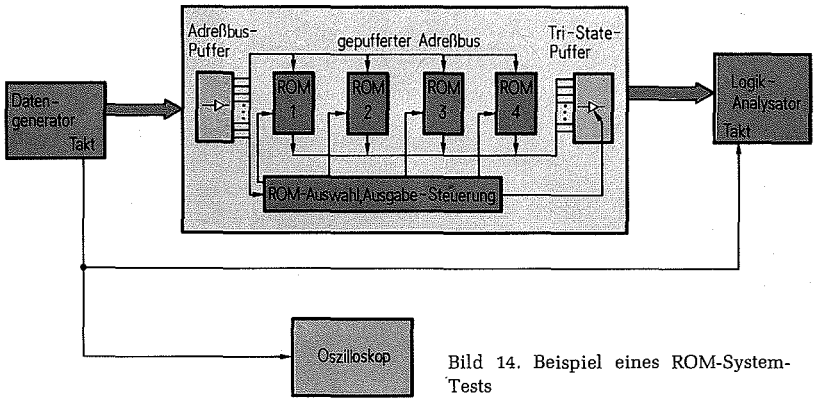


Bild 14. Beispiel eines ROM-System-Tests

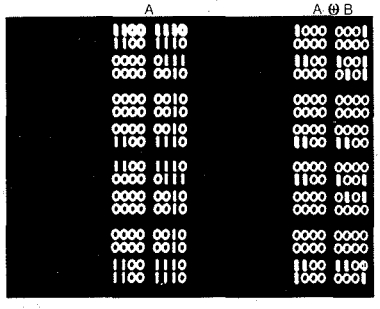


Bild 15. Vergleich zwischen Ist- und Soll-Daten; Fehler werden, die beiden rechten Spalten zeigen dies, durch die strahlintensivierte „1" angezeigt

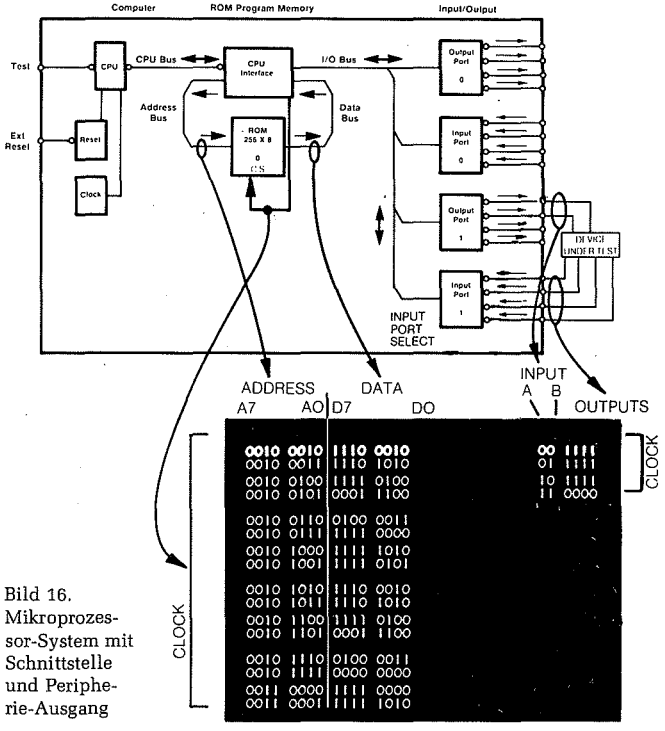


Bild 16. Mikroprozessor-System mit Schnittstelle und Peripherie-Ausgang

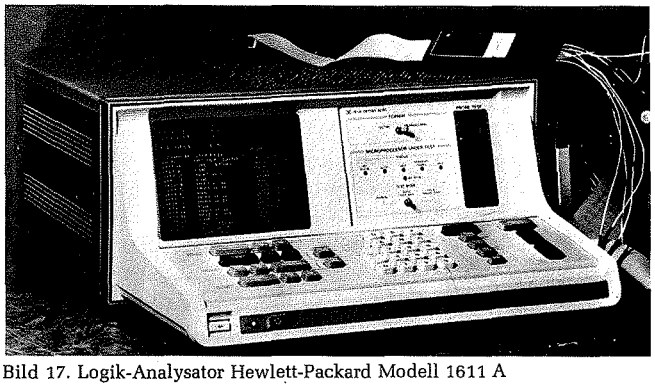


Bild 17. Logik-Analysator Hewlett-Packard Modell 1611 A

5.5 Funktionelle Tests an Zählschleifen

Zählschleifen gehören zum Standard-Repertoire jedes Programmschreibers. Sei es, daß eine bestimmte Wartezeit aufgebracht werden muß, bevor das Programm weiterlaufen kann, oder, daß Ereignisse gezählt werden; beim Test der Software interessiert den Entwickler, ob die Schleife wirklich so oft, wie vorher programmiert, durchlaufen wurde. Der Logik-Analysator Hewlett-Packard Modell 1611A (*Bild 17*) hat einen eingebauten Zähler: Bedingungen für den Ereigniszähler sind die Tasten-Anwahl einer Start- und Stop-Bedienung, sowie die Eingabe, welches Ereignis (Triggerwort) gezählt werden soll. Die alphanumerische Anzeige stellt das Ergebnis sofort dar. Für Warteschleifen ist es auch wesentlich, die absolute Zeit für das Durchlaufen einer Schleife zu kennen. Der erwähnte Analysator zeigt die Zeit absolut an, mit einem internen quarzstabilen Oszillator gemessen.

6 Ausblick

Weder für Entwicklungs-Systeme und Simulatoren noch für Logik- bzw. Mikroprozessor-Analysatoren ist mit der jetzigen Generation ein Endzustand erreicht, wie das etwa bei Oszilloskopen zu sehen ist. Die Entwicklung wird dem Trend der Mikroprozessoren folgen.

Die Electronica 1976 in München zeigte, daß die 8-bit-Mikroprozessoren zur Zeit im Mittelpunkt stehen. Auf der Messe waren aber auch Tendenzen in zwei Richtungen zu erkennen: Die eine führte zu den einfacheren und preiswerteren Mikroprozessoren für kommerzielle Anwendungen, wie etwa Waschmaschinen- und Küchengeräte-Steuerung. Die andere wies in Richtung der 16-bit-Mikroprozessoren, die in gleicher Zeit mehr Programmarbeit erledigen als die heutige Generation. Abzusehen ist die Entwicklung der Meßgeräte schon heute: Die Logik-Analysatoren, generell einsetzbar, werden sowohl für kombinatorische und sequentielle Logik als auch für Computer und Mikroprozessoren konzipiert.

Die Mikroprozessor-Analysatoren werden sich speziell auf ihren Anwendungsbereich konzentrieren. Ihre Flexibilität wird in der Anpassung an verschiedene Modelle liegen. Sie werden komfortabler im Hinblick auf Bedienung, Darstellung und Programmeingriff sein.

Die Entwicklungs-Systeme schließlich werden auch in einfachen und preisgünstigen Ausführungen angeboten werden, um die Anfangsinvestition gering zu halten, und damit die praktische Erfahrung mit Mikroprozessoren einem noch größeren Interessentenkreis zugänglich zu machen.

Dipl.-Ing. Jochen Müller studierte Nachrichtentechnik an der TH Aachen. Als Entwicklungsingenieur bei der Firma AEG-Telefunken in Ulm konzentrierte er sich auf die Digitaltechnik. 1970 übernahm er im Marketing der Firma Hewlett-Packard in Böblingen Aufgaben zur Verkaufsunterstützung und -förderung des europäischen Meßgeräte-Vertriebs. In letzter Zeit führte er das von Hewlett-Packard entwickelte neue Meßkonzept des Datenbereiches in Europa ein. In Seminaren für Verkaufsingenieure und Kunden erwarb er spezielle Kenntnisse über Probleme in Mikroprozessor-Schaltungen.
Hobbys: Fotografie einschließlich Dunkelkammer; Skifahren; moderne und klassische Musik.
Telefon: (0 70 31) 6 67 16 74
ELEKTRONIK-Leser seit 1967

Literatur

[1] *Runyon, S.*: Logic Analyzers: A new Force in Digital Troubleshooting. Electronic Design, Bd. 22 (1974), H. 24, S. 16...24.

[2] *House, Chuck*: Engineering in the Data Domain calls for a new kind of Digital Instrument. Electronics, Bd. 48 (1975), H. 9, S. 75...81.

[3] Programmieren lernt man spielend. ELEKTRONIK 1976, H. 11, S. 11

[4] *Small, C.T.und Morrill, Jr., J.S.*: The Logic State Analyzer, A Viewing Port for the Data Domain. Hewlett-Packard Journal, August 1975.

[5] Logikanalysator für Mikroprozessorsysteme. ELEKTRONIK 1976, H. 12, S. 106, (Elektronik-Markt).

[6] *Hill, J.C. und Fiedler, C.*: Logic Analyzers in System Debugging Make Time Run Backward. Computer Design, Bd. 14 (1975), H. 12, S. 67...72.

[7] *Farnbach, W.A.*: Logic State Analyzers – A New Instrument for Analyzing Sequential Digital Processes. IEEE-Transactions on Instrumentation and Measurement, H. 4, Dezember 1975.

[8] *Farnbach, W.A.*: Bring up your µP „bit-by-bit". Electronic Design, Bd. 24, (1976), H. 15, S. 80...85.

[9] Analyzers for microprocessors coming on strong. Electronics, Bd. 49 (1976), H. 17, S. 31...32.

[10] *Quick, P.*: Logikrecorder – unentbehrlich in der Digitaltechnik. messen + prüfen/automatik (1976), H. 11, S. 658...666.

[11] *Santoni, A.*: Tester are getting better at finding microprocessor flaws. Electronics, Bd. 49 (1976), H. 26, S. 57...66.

[12] *Smith, J.H.*: A Logic State Analyzer for Microprocessor Systems. Hewlett-Packard Journal, Januar 1977.

[13] Sonderausgabe ELEKTRONIK 1977: Mikroprozessoren. Franzis-Verlag, München.

[14] *Runyon, S.*: Focus on Logic and µP Analyzers. Electronic Design, Bd. 25 (1977), H. 3, S. 40...50.

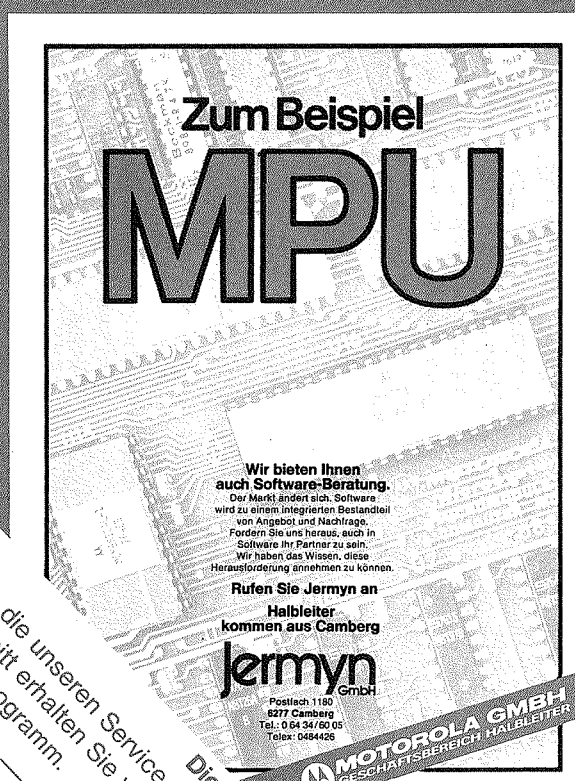
[15] Signaldarstellung im Zeit- und Frequenzbereich. ELEKTRONIK-Arbeitsblatt, 1971, H. 8, S. 293...294.

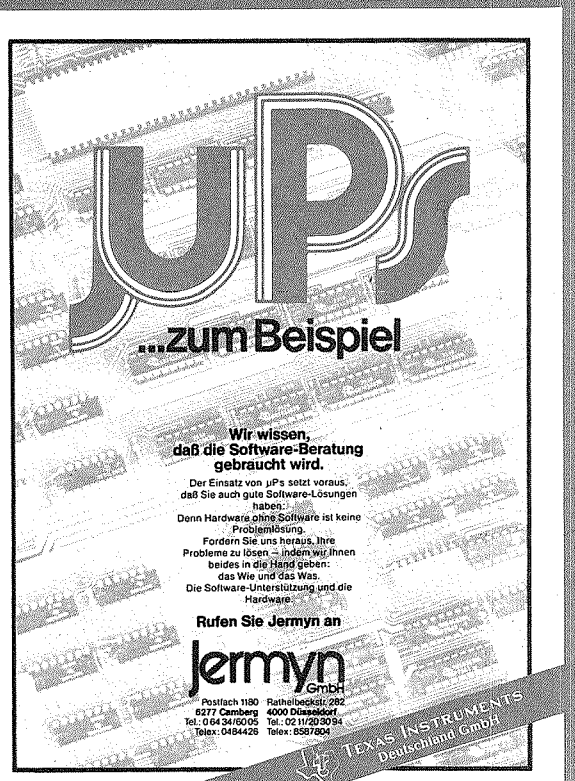
[16] *Blomeyer, H.-P.*: Logik-Tester für Mikroprozessor-Systeme. ELEKTRONIK 1976, H. 9, S. 66...68.

[17] Data Domain-Datenbereichsdarstellung, Report 4/76 der Firma Rhode & Schwarz Vertriebs GmbH, Köln.

Der Service von Jermyn in Camberg

Zum Beispiel: Mikro Prozessor-Service





Die Service-Ecke

lle, die unseren Service besser kennenlernen wollen. Mit diesem hnitt erhalten Sie weitere Informationen über unser rogramm.

e:

se:

on vorher bei
n bestellt:
ein

Rufen Sie Jermyn an. Halbleiter kommen aus Camberg

Jermyn GmbH

Postfach 1180, 6277 Camberg/Ts.

Mikroprozessoren in der Prozeßlenkung

Über dieses Thema fand Ende 1976 im *Kernforschungszentrum Karlsruhe* eine Fachtagung statt, zu der das Projekt Prozeßlenkung mit DV-Anlagen (PDV) [1] eingeladen hatte. Der 454 Seiten umfassende Tagungsband liegt jetzt vor [2] und kann bei der Literaturabteilung der Gesellschaft für Kernforschung (Postfach 3640, 7500 Karlsruhe) bestellt werden.

Den etwa 300 aus Instituten und Industriefirmen erschienenen Fachleuten wurden in 17 Referaten vor Augen geführt, daß die Prozeßautomatisierung ein beträchtliches Anwendungspotential für Mikroprozessoren darstellt. Mehr und mehr Förder-, Regal- und Transportanlagen, Kran- und Antriebssteuerungen, Bestückungsautomaten, Abfüllmaschinen, Extruder zur Herstellung von Kunststofformteilen, Werkzeugmaschinen, chemische Anlagen...usw. werden in Zukunft mit Mikroprozessoren ausgerüstet werden.

Das war auch den zahlreichen im Tagungsband nicht abgedruckten Diskussionsbeiträgen zu entnehmen, die im folgenden zusammenfassend wiedergegeben werden:

Am ersten Tag galt das besondere Interesse den *Speicherbausteinen*, wie z. B. den in Kürze erhältlichen N-MOS Chips für 64 Kbit.

Auch das Thema *Zuverlässigkeit* wurde angeschnitten. Neben den Test- und Prüfverfahren, die off-line und vor dem Einbau der ICs angewandt werden, sollten on-line Testverfahren zur Verfügung stehen, die eine schnelle Fehlerdiagnose von Mikroprozessor-Systemen ermöglichen und damit die MTTR reduzieren helfen. Die von den Herstellern angebotenen Entwicklungssysteme sind dafür nicht geeignet.

Die heute verfügbare Mikroprozessor-*Software*, so war zu hören, läßt noch viele Wünsche offen. Bezüglich rationeller Softwaregenerierverfahren besteht gegenüber der kommerziellen EDV ein jahrelanges Nachholbedürfnis. Die meisten Mikroprozessor-Entwicklungssysteme sind auf Assembler-Programmierung zugeschnitten und setzen beim Anwender sehr detaillierte Systemkenntnisse voraus. Der These, daß nur durch höhere Programmiersprachen das ungünstige Verhältnis zwischen Soft- und Hardwareentwicklungskosten von ca. 10 : 1 verbessert werden kann, wurde nicht widersprochen.

Allerdings – und auch darüber war man sich einig – müssen für Mehr-Mikroprozessor-Systeme noch geeignete Sprachelemente entwickelt werden, die die Synchronisation von „realtime tasks" in örtlich verteilten Prozessoren erleichtern. Die Projektleitung PDV bemerkte dazu, daß bei der Implementierung der Prozeßprogrammiersprache PEARL [3] in geförderten Vorhaben die sich anbahnende „Intelligenzverteilung" berücksichtigt wird.

Ein weiteres Thema waren die *Kommunikationseinrichtungen* zwischen örtlich verteilten Mikroprozessoren. Die für Prozeßanwendungen typischen Anforderungen, wie geringe Reaktionszeiten, rückwirkungsfreie Ankopplungsmöglichkeit der einzelnen Stationen, hohe Sicherungseffizienz der zu übertragenden Informationen und nicht zuletzt einheitliche Schnittstellen und Leitungsprotokolle werden durch vorhandene Standards nicht erfüllt, und bekannte Firmensysteme [4] sind zu technologieabhängig.

Die Projektleitung PDV wies auf die gemeinsamen Bemühungen um ein Standardprozeßbussystem hin, das zur Verbindung von verteilten Mikroprozessoren geeignet ist. Gemeint war der *PDV-Bus* [5] der zur Zeit in geförderten Vorhaben von vier großen Herstellern und vier weiteren Institutionen implementiert wird.

Am zweiten Tag stand der Einsatz von Mikroprozessoren in Regelsystemen im Mittelpunkt der Diskussion. Die Programmierbarkeit von Mikroprozessoren führt hier zu einer *Verbesserung der Regelgüte*, weil nun durch fortgeschrittene Regelalgorithmen Sollwerte näher an Grenzwerte gelegt werden können, weil mit vertretbarem Aufwand selbsteinstellende Regler mit adaptivem Verhalten möglich werden und weil Planungsfehler durch Änderung der Reglerstruktur (z. B. bei Kaskadenreglern) vor Ort leichter zu korrigieren sind [4]. Die Reglercharakteristik kann statisch durch den Austausch steckbarer ROMs oder dynamisch durch das Umladen von RAMs geändert werden.

Bei der abschließenden Podiumsdiskussion wurde herausgestellt, daß der Mikroprozessor nicht nur zur Übernahme von Regel- und Steuerfunktionen prädestiniert ist. Er läßt sich bereits am Beginn des Signalwegs, d. h. beim Erfassen von Prozeßgrößen einsetzen. Hier kann er nichtlineare Kennlinien von Sensoren linearisieren, Meßwerte skalieren oder Interfacefunktionen an der Schnittstelle zwischen lokalen Subsystemen und einer Kommunikationseinrichtung (z. B. Bussystem) übernehmen. Er kann vor Ort Redundanzreduktion vornehmen und auf bestimmte Prozeßalarme autonom reagieren.

Auf einer hierarchisch höheren Systemebene lassen sich Mikroprozessoren für Überwachungs-, Optimierungs- und Estimationsaufgaben einsetzen.

In Prozeßwarten können sie mit der gezielten Informationsaufbereitung, -verdichtung und -auswahl betraut werden und dadurch den Informationsaustausch zwischen Mensch und System verbessern helfen. Beispiele dafür sind mit Mikroprozessoren aufgebaute Grafiksysteme, die außer alphanumerischen Informationen auch Prozeßschemata, Balken- und Flußdiagramme, Kurven usw. farbig darstellen können.

Am Schluß der Veranstaltung wurde das *Ausbildungsproblem* angesprochen. Der Regelungstechniker, der bisher an analogen Kompaktreglern P-I-D-Anteile mit dem Schraubenzieher einzustellen gewöhnt war, muß mit ROMs, digitalen Signaldarstellungen...usw. vertraut gemacht werden. Der dafür notwendige Umschulungsprozeß sollte sobald als möglich begonnen werden, um dem abschreckenden Beispiel „Uhrenindustrie" [6] kein weiteres folgen zu lassen.

Die PDV-Fachtagung hat den mit der Prozeßautomatisierung befaßten Fachleuten veranschaulicht, was auf sie zukommt. Diese neue DV-Technologie muß nun rechtzeitig und nicht nachempfindend in die breite Anwendung gebracht werden, um auch international auf diesem Gebiet wettbewerbsfähig zu bleiben. Das Projekt PDV leistet durch gezielte Fördermaßnahmen seinen Beitrag dazu [7].

H. Walze, Projekt PDV

Literatur

[1] *Eckert, H.:* Projekt Prozeßlenkung mit DV-Anlagen. Regelungstechnische Praxis, rtp, Januar 1976.

[2] *Barthel, F. u. a.:* Einsatz von Mikroprozessoren zur Prozeßlenkung, KFK-PDV-Bericht 101, Januar 1977.

[3] *Timmelsfeld, K.H., u.a.:* PEARL – Vorschlag für eine Prozeß- und Automatisierungssprache. KFK-PDV-Bericht 1, April 1973.

[4] Dezentralisiertes digital arbeitendes Automatisierungssystem TDC 2000. ELEKTRONIK 1976, H. 3, S. 30.

[5] *Walze, H.:* PDV-Bus löst Kopplungsprobleme bei der industriellen Prozeßautomatisierung. ELEKTRONIK 1977, H. 1, S. 77...80.

[6] Darunter kommt alles ins Rutschen. Der Spiegel. 1976, H. 51, S. 87.

[7] PDV-Arbeitskreis TP71: Anwendung von Mikroprozessoren im Rahmen von PDV-Vorhaben. PDV-Entwicklungsnotiz 84, Oktober 1976.

Mit dem Aufkommen des Mikroprozessors wird der Elektronikfachmann mit einer Aufgabe konfrontiert, für die er bislang kaum zuständig war: die Erstellung von Software. Der folgende Beitrag gibt einen Überblick über heute übliche Programmiertechniken, jedoch keine Einführung in das Programmieren selbst. Kern der Ausführung ist die Darstellung von Programmiersprachen für Mikroprozessoren. Eine Analyse der Wirtschaftlichkeit zweier Programmiermethoden weist den Weg zukünftiger Verfahren.

Dipl.-Inform. G. R. Koch

Stand und Trends der Programmierung von Mikroprozessoren

Wenn Mikrocomputer vorgestellt werden, so stehen die systemtechnologischen Bewertungsmaßstäbe immer obenan. Die Charakterisierung orientiert sich an den Struktur- und Architekturelementen der Hardware und ihrer Leistungsdaten. Eine zweite gängige Klassifizierung ist die phänomenologische Beschreibung ihres Einsatzes. Hier lassen sich zwei Bereiche unterscheiden:

- Der Mikrocomputer als Ersatz für festverdrahtete Logik z. B. als Steuereinheit für Datenperipheriegeräte oder als „eingebaute Intelligenz" in technischen Geräten wie Waagen, deren Fertigungszahlen sie als Massenprodukte ausweisen
- der Mikrorechner als eigenständiger Nachfolger der Mini- und Prozeßrechner.

Der ELEKTRONIK-Report [1] über Mikroprozessoren berücksichtigt auch die Programmiertechnik als ein wesentliches Kriterium für die Verwendungsfähigkeit. Er erläutert auch die wichtigsten Grundbegriffe der Programmiertechniken.

Bei allem Trubel um die technologische Novität „Mikroprozessor" muß gefragt werden, ob überhaupt eine spezielle, für den Mikroprozessor besonders geeignete Programmierung existiert. Aus der Sicht des Softwareingenieurs nämlich unterscheidet sich vorläufig die Hardware/Software-Schnittstelle in nichts von dem, was man von konventionellen Computern her kennt.

1 Gegenwärtiger Stand des Mikrorechnerprogrammierens

Im Augenblick erlebt der Softwarespezialist die Wiederentdeckung seines „Rades": Die Kunst des Programmierens, die parallel mit der Entwicklung der Mega- und Minicomputer der 50iger und 60iger Jahre einen hohen Stand erreicht hat, muß beim Mikrocomputer von Grund auf neu erarbeitet werden. Die Stufen werden chronologisch die gleichen sein wie bei den Megas und Minis, aufwendige und teure Entwicklungen wiederholen sich. Indizien hierfür sind:

1. Eine ursprünglich magere Hardware wird funktionell ständig erweitert. Das läßt sich einmal an der Zunahme der spezielleren Befehle z. B. zur Behandlung von ganzen Registerfeldern, von Stapelspeichern, zur Beeinflussung einzelner Bits und zur Manipulation von Zeichenketten, zum anderen an den immer komfortableren Interruptbehandlungen ablesen. Ein Rückblick in die von Neumannsche und Zusesche Ära lehrt, daß seinerzeit Hardware in ähnlicher Weise sukzessiv angereichert wurde.
2. Die Anwenderprogramme werden immer größer. Eine Umfrage in Großbritannien ergab, daß nur noch etwa 15 % der Anwender Mikrorechnerprogramme erstellen, die kleiner als 1 KByte sind. Zur Zeit typische Programmlängen sind 0,5...4 KByte [4].
3. Die Anwendungserfordernisse nehmen an Komplexität zu. Der Ruf nach kommunikationsunterstützender Standard-Software, nach Programmbibliotheken und Betriebssystemen wird lauter. Einige Hersteller bieten schon Floppy-Disk-Betriebssysteme an.
4. Softwarehäuser und Hersteller erstellen immer komfortablere Programmierhilfsmittel und nicht zuletzt höhere Programmiersprachen.

Im augenblicklichen Zustand besteht der eigentliche und einzige Unterschied der Mikrorechnerprogrammierung nur im „Drumherum" des Programmierens. Mikrosystementwicklungen finden in der experimentellen Atmosphäre eines Elektroniklabors statt. Es liegt nahe und ist den Kenntnissen eines Hardwareingenieurs angemessen, Programme in einem Binärcode oder in einem mnemonischen Maschinencode zu formulieren. Dies lohnt sich bei Einmalanwendungen für Programmlängen bis 256 Byte, da sich hier der Aufwand für das Erlernen komplizierterer Programmiermethoden nicht rechtfertigen ließe.

Als eine erste Hilfe benutzt der Programmierer ein Editorprogramm, das Erstellung und Behandlung eines Programmtextes unterstützt: Es realisiert das Aufsuchen von Zeilen, bestimmten Befehlen und Symbolen wie Sprungziele, ausgezeichnete Operandennamen und Adressen. Unter Aufsicht dieses Texterstellungsprogramms ist es möglich, problemlos Programmzeilen zu löschen, einzufügen und anzufügen.

Zum „Drumherum" der Programmierung gehören auch die überall verfügbaren Großrechner, sei es als hauseigene Anlage oder als per Telefon anwählbarer Timesharing-Computer. Auf diesen Gastrechnern lassen sich durch soge-

nannte Cross-Programme und mit Unterstützung der komfortablen Peripherie und der Systemsoftware Mikrocomputerprogramme schnell erstellen und testen. An Cross-Software sind im wesentlichen zu nennen:

- der *Editor* als Texterstellungshilfsmittel,
- der *Assembler* zum Übersetzen von in Assemblersprache geschriebenen Programmen in den Maschinencode (unglücklicherweise wird dieses Übersetzungsprogramm ebenso mit Assembler bezeichnet wie die zu übersetzende Sprache),
- der *Compiler* zum Übersetzen einer höheren Programmiersprache in Maschinencode,
- der *Simulator* zum zeitlich unechten Nachvollziehen und Testen des erzeugten Maschinenprogramms.

Auf dem Zielsystem, also dem verwendeten Mikrocomputer selbst, muß an sogenannter *residenter* (anwesender) Software vorhanden sein:

- ein *Lader*, der z. B. den vom Gastrechner auf einem Lochstreifen ausgestanzten Maschinencode in den Programmspeicher einliest,
- ein *Testprogramm*, das eine Untersuchung des Echtzeitverhaltens der Zielhardware beim Ablauf des „crosserzeugten" Programms erlaubt.

Der „Cross"-Programmierprozeß in seinem gesamten Umfang ist in *Bild 1* dargestellt. Die Gastrechnerprogrammierung ist zwar ein sehr angenehmes Verfahren, doch mit Sicherheit nicht das billigste. Nach [5] muß bei Benutzung eines Timesharing-Computers pro Befehl etwa 4,50 DM veranschlagt werden, also 1800 DM pro Programmierer und Monat. Diese beachtlichen Kosten waren für die Hersteller bald Anlaß, eigenständige Entwicklungssysteme anzubieten. Das folgende Blockdiagramm in *Bild 2* stellt eine mögliche Konfiguration eines solchen „Stand-alone-Systems" dar. *Bild 3* zeigt die dazu nötige und erwünschte Software.

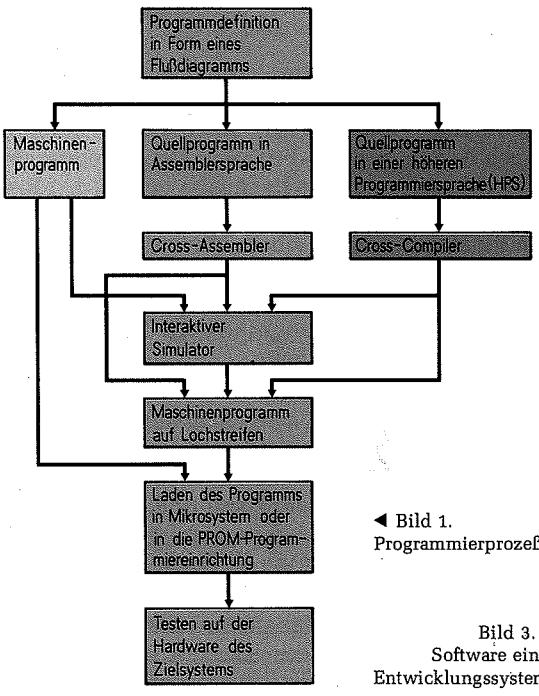
Die Vorteile eines Entwicklungssystems sind: Programme können in Echtzeit in einer lebendigen, der zukünftigen Anwendung gemäßen Umgebung getestet werden. – Die Aufgabenteilung zwischen Hardware und Software läßt sich konkret optimieren. – Trotz hoher Anfangsinvestitionen sind die Rechnerkosten summiert über eine längere Zeit erheblich geringer als bei Timesharing-Entwicklungen. – Bei einer Erstinvestition von 25 000 DM und einer angenommenen dreijährigen Amortisationszeit liegen hier die monatlichen Aufwendungen bei ca. 250 DM pro Programmierer [5].

Die Kehrseite dieser Medaille ist die höchst umständliche Handhabungsprozedur, wie sie *Bild 4* zeigt. Durch das mehrfache Einlesen des Quellprogramms und das Ausgeben des Zielprogramms bringt die Verwendung einer Floppy Disk als Programmträger gegenüber Lochstreifen eine Beschleunigung des reinen Handhabungsablaufes etwa um den Faktor 10...50.

Der konstruktivste Ansatz, die Vorteile der Softwareentwicklung auf Gastrechnern und die der Programmierung auf einem Entwicklungssystem zu vereinigen, ist die Verkopplung des Zielrechners mit dem Gastrechner (*Bild 5*). Diese Ankopplung kann erfolgen: über die normale V-24-Standardschnittstelle, über die Verknüpfung der jeweiligen Systembusse bzw. der nichtstandardisierten Parallelinterfaces oder über den direkten Speicherzugriff des Mikrocomputers auf den Arbeitsspeicher des großen Rechners.

Einen anderen, verkaufspolitisch sehr klugen Weg gingen die Firmen DEC und Data General mit den Mikrocomputern LSI 11 und „micronova": Der Befehlsvorrat ist dabei eine Untermenge der schon vorhandenen Minisysteme, der PDP-11- bzw. Nova-Familie. Letztlich sind diese Kleinstrechner hochintegrierte Nachahmungen schon vorhandener Rechner; die Programme können auf den konventionellen Systemen erstellt werden.

Ein letztes sehr zukunftsträchtiges Verfahren ist die Emulation der Befehle des Zielsystems auf einem anderen Rechner. Unter Emulation wird die Anpassung der vorgegebenen Architektur eines z. B. größeren Rechners an den Befehlsvorrat des Zielrechners verstanden. Dies ist im wesentlichen bei Computern zu erreichen, die ein mikroprogrammierbares Steuerwerk besitzen. Mit Hilfe der Mikroprogrammierung – das hat nichts mit Mikroprozessorprogrammierung gemein – läßt sich der Gastrechner zum Zielrechner „ver-



◄ Bild 1.
Programmierprozeß

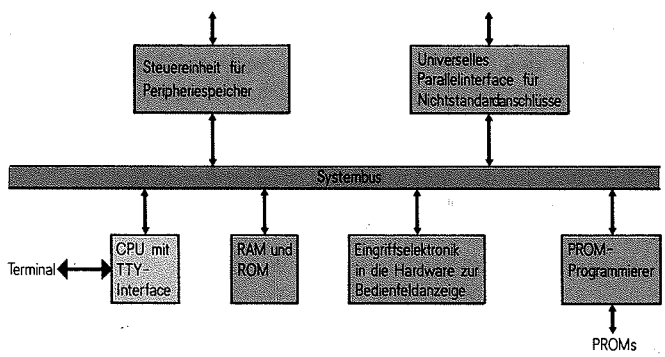


Bild 2. Konfiguration eines Entwicklungssystems

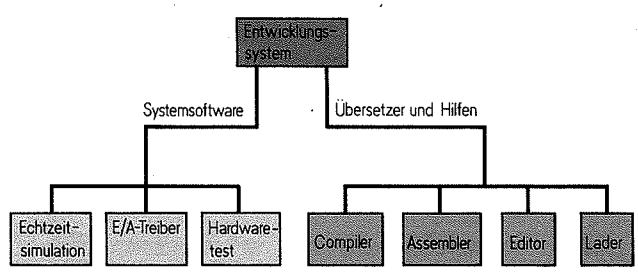
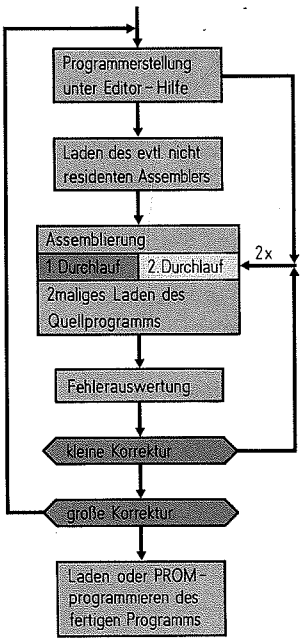


Bild 3. ►
Software eines
Entwicklungssystems

Tabelle 1. Beurteilung einiger Assemblersprachen

Kriterium / Assemblersprache	Intel 8080	Motorola 6800	Signetics 2650 (Cross)	Fairchild F 8	National IMP-16	Rockwell PPS-8
Leicht lesbares, sauberes u. strenges Format	Große Formatunabhängigkeit. Vorteil: Flexibilität Nachteil: Gefahr unstrukturierter Darstellung	Spaltenunabhängig. Einschr. bzgl. Spalte 1. Automatische Feldausrichtung	Spaltenunabhängig. Einschr. bzgl. Spalte 1	Spaltenunabhängig. Einschr. bzgl. Spalte 1	Spaltenunabhängig. Verwendung von Feldindikatoren	Spaltenabhängig
Vielfältige Möglichkeit der Definition von Namen durch d. Benutzer	5 Zeichen, 1. Zeichen Buchstabe, od. ? 1) Def. per Marke 2) Def. per Gleichsetzung 3) Def. per Zellenreservierung 4) Def. per Feldvorbesetzg.	6 Zeichen, 1. Zeichen Buchstabe 1) Def. per Marke 2) Def. per Gleichsetzg. 3) Def. per Zellenreservierung 4) Def. per Feldvorbesetzg.	4 Zeichen, 1. Zeichen Buchstabe Def. wie b. Intel 8080	6 Zeichen, 1. Zeichen Buchstabe Def. wie b. Intel 8080 Ausnahme: Zellenreserv.	Def. wie b. Intel 8080	8 Zeichen, Mind. 1 Zeichen = Buchstabe Def. wie b. Intel 8080
Vielfalt von Datentypen	Hexadezimal Oktal Dezimal Binär ASCII Strings	Hexadezimal Oktal Dezimal Binär ASCII (Strings)	Hexadezimal Oktal Dezimal Binär ASCII Strings	Hexadezimal Oktal Dezimal Binär ASCII Strings	Hexadezimal Dezimal ASCII Strings	Hexadezimal Dezimal Binär
Formulierung komplexer Adreßausdrücke auch mittels logischer Operatoren	Komfortabel; mit log. Operationen u. Shiften	Alle 4 arithm. Grundoperat.	Nur Addition u. Subtrakt.	Alle 4 arithm. Grundoperat. u. Exp.	Arithm. Grundoperat. u. log. Operat.	Nur Addition u. Subtrakt.
Option eines Symboltabellenausdrucks	Alphab. sortiert	Unsort.	–	Alphab. sortiert; Zuordn. auch oktal od. dezim.	Alphab. sortiert	
Option einer Kreuzreferenztabelle				vorhanden		vorhanden
Verständliche Fehlerdiagnose	Subjektives Kriterium; große Variationsbreite					
Option der bedingten Assemblierung	vorhanden				vorhanden	
Verschiebbarkeit der Programme					möglich	
Bildung von parametrisierbaren Makros	möglich			(Bei Cross-Version möglich)		



◀ Bild 4.
Programmierprozeß auf einem Entwicklungssystem

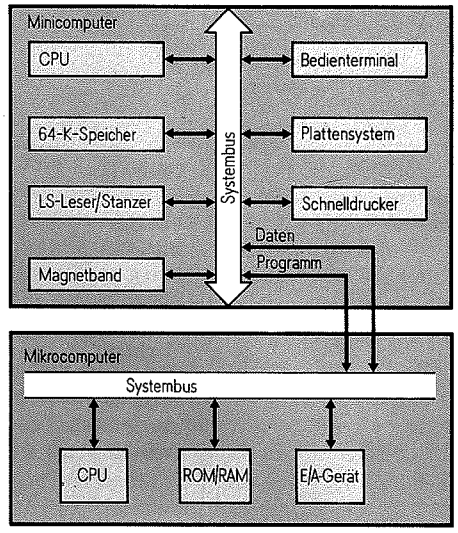


Bild 5. ▶
Konfiguration einer Kopplung Gastrechner/Mikrocomputer

```
Markenfeld   Operationsfeld   Operandenfeld   Kommentarfeld

LOO          Load             RØ   TABL+3     Adresse aus Sprung-
                                                tabelle holen
```

◀ Bild 6.
Beispiel einer Assembleranweisung

wandeln", indem ihm beigebracht wird, die Zielrechnerbefehle zu interpretieren. Die Firma Intel bietet eine interessante Entwicklungshilfe an, die sich das Prinzip dieser Methode zunutze macht. Das zunächst auf dem Entwicklungssystem erstellte Programm muß nicht notwendigerweise auch auf der endgültigen Zielkonfiguration korrekt ablaufen. Um nun die Fehlerquellen im Zielsystem bequem aufspüren zu können, wird durch eine „In-Circuit-Emulation" der Zielsystemprozessor durch die Zentraleinheit des Entwicklungssystems per Hardwareverbindung ersetzt. Dies ermöglicht den Programmtest unter Aufsicht eines Fehlersuchprogramms des Entwicklungssystems.

2 Assemblersprachen

Zwar ist die Handhabung das sichtbar Typische der Mikrorechnerprogrammierung, jedoch ist damit noch nichts über die Qualität der Programme gesagt. Diese interferiert auf vielfältige Weise mit der verwendeten Programmiersprache und der damit möglichen Programmierdisziplin.

Der heute übliche Standard ist die Programmierung in einer Assemblersprache oder Symbolsprache. Die von den Herstellern angebotenen Symbolsprachen variieren in ihren Eigenschaften jedoch so sehr, daß hier nur versucht werden kann, anhand eines Kriterienkatalogs einen groben Überblick zu geben (*Tabelle 1*).

Das erste Kriterium verlangt eine saubere Formatierung. *Bild 6* zeigt den typischen Aufbau eines Assemblerbefehls in vier Feldern. Im Markenfeld kann dem darauf folgenden Befehl bzw. dem Datum ein frei wählbarer Name gegeben werden. Im Operationsfeld befinden sich entweder ausführungsbezogene Operationsanweisungen für den Prozessor oder Direktiven an das Assemblerübersetzungsprogramm. Im Operandenfeld schließlich sind die zu manipulierenden Speicher- oder Registerinhalte oder die Adressen dieser Zellen, gegebenenfalls auch in einer noch zu berechnenden arithmetischen Ausdrucksform, notiert. Generell ist vereinbart, daß in jeder Assemblerprogrammzeile genau ein Befehl zu stehen hat. Innerhalb einer Zeile ist man nicht immer an eine feste Spaltenordnung gebunden. *Bild 7* beweist jedoch, daß eine strenge Formatierung wesentlich übersichtlicher wirkt. Insofern ist die völlige Zeilen- und Spaltenungebundenheit, wie sie z. B. Intel anbietet, nur dann sinnvoll, wenn sich der Programmierer an die Disziplin einer anschaulichen optischen Gestaltung hält. Zu erwähnen ist hier eine Option des 6800-Übersetzers, die eine automatische Feldausrichtung bewirkt.

```
BUFA EQU H'0800'  SET THE VALUE OF SYMBOL BUFA
BUFB EQU H'08A0'  SET THE VALUE OF SYMBOL BUFB
 ORG H'0100'
 DCI BUFA SET DCO TO BUFA STARTING ADDRESS
 XDC STORE IN DCI
 DCI BUFB SET DCO TO BUFB STARTING ADDRESS
 LI H'80' LOAD BUFFER LENGTH INTO ACCUMULATOR
 LR I,A SAVE BUFFER LENGTH IN SCRATCHBAD BYTE I
LOOP LM LOAD CONTENTS OF MEMORY BYTE ADDRESSED BY DCO
 XDC EXCHANGE DCO AND DCI
 ST STORE ACCUMULATOR IN MEMORY BYTE ADDRESSED BY DCO
 XDC EXCHANGE DCO AND DCI
 DS I DECREMENT SCRATCHPAD BYTE I
 BNZ LOOP IF SCRATCHPAD BYTE I 15 NOT ZERO, RETURN TO LOOP
```

```
BUFA  EQU   H'0800'   SET THE VALUE OF SYMBOL BUFA
BUFB  EQU   H'08A0'   SET THE VALUE OF SYMBOL BUFB
      ORG   H'0100'
      DCI   BUFA      SET DCO TO BUFA STARTING ADDRESS
      XDC             STORE IN DCI
      DCI   BUFB      SET DCO TO BUFB STARTING ADDRESS
      LI    H'80'     LOAD BUFFER LENGTH INTO ACCUMULATOR
      LR    I,A       SAVE BUFFER LENGTH IN SCRATCHPAD BYTE I
LOOP  LM              LOAD CONTENTS OF MEMORY BYTE ADDRESSED BY DCO
      XDC             EXCHANGE DCO AND DCI
      ST              STORE ACCUMULATOR IN MEMORY BYTE ADDRESSED BY
                                                              DCO
      XDC             EXCHANGE DCO AND DCI
      DS    I         DECREMENT SCRATCHPAD BYTE I
      BNZ   LOOP      IF SCRATCHPAD BYTE I IS NOT ZERO? RETURN TO LOOP
      END
```

Bild 7. Die strenge Formatierung (rechts) ist für den Programmierer wesentlich übersichtlicher

Tabelle 2. Die vier wichtigsten Möglichkeiten, im Assemblerprogramm frei wählbare Namen zu definieren

1. Definition per Markenfeld LOOP LOAD COUNTER, A	Der Name (LOOP) steht im Markenfeld
2. Definition per Gleichsetzung LOOP = 100	Dem Namen wird explizit ein Wert zugeordnet
3. Definition per Reservierung COUNTER RES 1	Mit jeder Platzreservierungsanweisung kann ein Name eingeführt werden. Dem Namen COUNTER z. B. weist der Assembler als Wert die Adresse eines freigehaltenen Speicherplatzes zu
4. Definition per Vorbesetzung TABELLE DC 1 DC 2 DC 4 DC 8 DC 16	Wenn mehrere Speicherzellen mit konstanten Werten vorzubesetzen sind, so gibt man diesem Konstantenbereich einen eigenen Namen. Der Zugriff zu den Zellen geschieht im Beispiel über die Bezeichnungen TABELLE bzw. TABELLE + 1 usw.

Tabelle 3. Vorgesehene logische Operationen in Adreßausdrücken

	Fairchild F8	Intel 8080	Motorola 6800	Rockwell PPS-8	National IMP-16	Signetics 2650
Subtraktion	–	–	–	–	–	–
Addition	+	+	+	+	+	+
Division	/	/	/		/	
Multiplikation	*	*	*		*	
Modulo		MOD				
Arithmetik Exponentiation	**					
Logisches NICHT		NOT			%	
Logisches UND		AND			&	
Logisches Exkl. ODER		XOR				
Logisches Inkl. ODER		OR			[	
Shift rechts		SHR				
Shift links		SHL				

Die zweite Forderung ist, daß es auf vielfältige Art möglich sein muß, im Programm frei wählbare Namen zu definieren. *Tabelle 2* gibt die vier wichtigsten Möglichkeiten an.

Das dritte Kriterium ist, auf wieviel Arten eine Datenkonstante geschrieben werden kann. Neben der natürlichen Dezimalform oder der maschinengemäßen Binärschreibweise sollten auch die Typen „oktal", „hexadezimal", „ASCII" und „Zeichenketten" gestattet sein.

Eine weitere Eigenschaft ist die Möglichkeit, im Operandenfeld komplexere Ausdrücke zur Adressenberechnung z. B. zur Gewinnung von Sprungadressen zu bilden. *Tabelle 3* gibt an, welche Operationen hier bei den diversen Assemblern erlaubt sind.

Die Forderung nach einer Symbol- bzw. Marken- bzw. Namenstabelle ist eine der wichtigsten überhaupt. Anhand einer solchen Tabelle kann der Programmierer feststellen, welche Namen von ihm definiert wurden, ob nicht etwa Doppeldefinitionen auftreten und welchen Wert diese Namen nach der Übersetzung besitzen. Komfortabler ist eine Kreuzreferenztabelle, die zusätzlich Auskunft erteilt, an welchen Stellen im Programm die definierten Namen verwendet oder ihre Werte verändert werden. Zwei weitere Kriterien seien hier nur kurz erläutert: Unter „bedingter Übersetzung" ist die Option einer Anweisung an das Übersetzungsprogramm zu verstehen, die festlegt, welche Programmteile aus dem gesamten Quellprogramm zu assemblieren sind. Dies erlaubt z. B. ein systemangepaßtes Erzeugen von Maschinencode aus Teilen eines generellen, umfassenden Assemblerprogramms für bestimmte Einsätze (*Bild 8*).

Verschiebbarkeit eines Maschinencodes bedeutet, daß mit einer Assemblierung noch nicht festgelegt ist, in welchen Speicherbereich das Maschinenprogramm zu liegen kommen wird. Alle Programmadressen sind zunächst bezogen auf den Programmanfang. Erst beim Laden des Programms durch ein spezielles residentes Programm (Lader) im Mikrocomputer werden die endgültigen Absolutadressen relativ zu einer vom Benutzer zu definierenden Anfangsadresse berechnet.

3 Höhere Programmiersprachen

Ein weitaus qualifizierteres, von den Hardwarefachleuten nur zögernd anerkanntes Hilfsmittel sind die höheren Programmiersprachen (HPS). Über ihre Anwendung bei Mikrocomputern werden nach wie vor leidenschaftliche Auseinandersetzungen geführt. In *Tabelle 4* sind die wesentlichsten Pro- und Contra-Argumente zusammengestellt. Dort findet sich u. a. das Argument, daß höhere Programmier-

Programmierplatz für Mikroprozessor-Software

Die Firma Siemens bietet jetzt einen kompletten Programmierplatz an, mit dem man die Software für Mikroprozessor-Systeme entweder direkt oder über Timesharing-Betrieb mit Hilfe eines Großcomputers erstellen kann. Um einen zentralen Mikrocomputer sind ein PROM-Programmiergerät, ein Floppy-Disk-System ein Modem zur telefonischen Datenübertragung und eine Datensichtstation angeordnet. Das System wird über eine Tastatur vom Benutzer bedient. Der Monitor, ein Texteditor und ein Assembler bilden die „lokalen" Programmierhilfen. Mit dem Programmierplatz kann man ein Programm zunächst ohne Großrechner über die Datensichtstation erstellen. Auch die nachfolgende Übersetzung ist mit der Anlage an Ort und Stelle möglich. Bei umfangreichen Programmen empfiehlt sich jedoch, auf die Möglichkeit zum Anschluß an die Siemens-Rechenanlagen 4004/151 und 7.750 zurückzugreifen und die schnellen Compiler des Betriebssystems BS 2000 zu benutzen. Das erstellte Programm wird nach seiner Prüfung unmittelbar auf PROMs abgespeichert. Eventuelle Fehler können mit dem Texteditor korrigiert werden. Mit Hilfe von entsprechend zugeschnittenen Cross-Assemblern und Simulatoren kann auf dem Programmierplatz die Software für alle Siemens-Mikroprozessor-Systeme erstellt werden.

☐ Hersteller: Siemens AG, ZVW-104, Postfach 103, 8000 München 1.

```
.IF     INDICATOR
 OUT    PORT 1
.ELSE
 OUT    PORT 2
.ENDIF
```

Bild 8. Beispiel einer optionellen Assemblierung

IF A>B, THEN C=A, ELSE C=B

```
TEST  SHL   B   : Lade Adresse von B
      LAM       : Lade B in Akkumulator
      SHL   A   : Lade Adresse von A
      CMPM      : Vergleiche B mit A
      JFC   L1  : Falls B  A springe nach L1
      LAM       : Lade A in Akkumulator
L1    SHL   C   : Lade Adresse von C
      LMA       : Speichere Akkumulatorinhalt nach C
      END
```

Bild 9. Oben: Problemformulierung in höherer Programmiersprache; unten: entsprechendes Assemblerprogramm

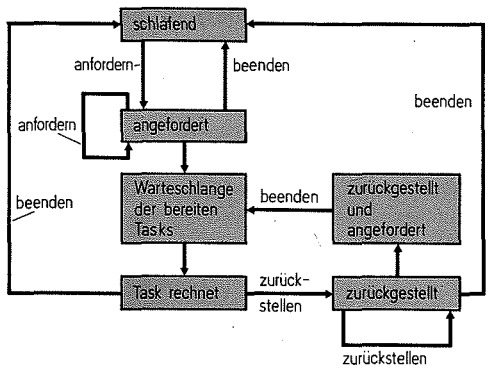


Bild 10. ▶ Der Graph gibt an, welche Aufgabe (*Task*) sich in welchem von fünf möglichen Zuständen zu befinden hat

sprachen eine selbstdokumentierende, problembezogene und schnell verständliche Programmformulierung gestatten. Als Beleg sei auf *Bild 9* verwiesen. Man beachte, wieviel Schreibaufwand nötig ist, das dem HPS-Programm äquivalente neunzeilige Assemblerprogramm so zu kommentieren, daß auch ein anderer als der Programmautor ohne viel Mühe nachvollziehen kann, was das Programm macht.

Alle bisher eingeführten HPS für Mikrorechner sind Derivate von PL/1. Die bekannteste Version ist die PL/M-Sprache, für die es in eigenständigen Entwicklungssystemen schon Compiler gibt. Neben den genannten Vorteilen von höheren Programmiersprachen hat PL/M weitere erwähnenswerte Züge:

- die starke Rücksichtnahme auf das 8- und 16-bit-Konzept durch entsprechende Datenstandardtypen;
- bei aller Maschinenunabhängigkeit ist es möglich, Hardwareeigenschaften auszunutzen, indem in relevanten Datenworten einzelne Bits manipuliert werden;
- aus PL/M übersetzte Programme sind gegebenenfalls ohne Betriebssystem ablauffähig. Die Sprache bietet sich somit als Systementwicklungssprache an.

PL/M eignet sich hervorragend zur Formulierung mathematisch-algebraischer Algorithmen. Die Sprache beinhaltet jedoch keine Elemente zur Behandlung dynamischer zeitlicher Anforderungen, wie sie eine Prozeßsteuerung verlangt.

An der „State University" von New York wird daher gegenwärtig ein System namens TOMAL (*Task Oriented Microprocessor Applications Language*) entwickelt, das die quasiparallele Aktivität mehrerer Programme, sogenannter Tasks, nach den Prinzipien der Echtzeitverarbeitung gestattet [9].

Die augenfälligsten Eigenschaften von TOMAL sind:

1. Es handelt sich um ein „Stand-alone"-Programmsystem, d. h., es ist kein gesondertes Betriebssystem vonnöten.
2. Um eine zuverlässige und übersichtliche Programmierung zu sichern, werden a priori gewisse „gefährliche" Sprachelemente der HPS ausgeschlossen, z. B. Referenzvariablen, Rekursion und variable Feldgrenzen.
3. Die verschiedenen quasi-parallel laufenden Aufgaben (*Tasks*) werden über einheitlich festgelegte Synchronisationsfunktionen koordiniert. Jede Aufgabe befindet sich eindeutig in einem von fünf möglichen Zuständen.
4. Die Ein-/Ausgabe ist per Software flexibel an diverse Schnittstellen anpaßbar. Datenwandlungsroutinen sind standardmäßig vorhanden.
5. Externe Unterbrechungen sind nur ausnahmsweise erlaubt. Normalfall ist die programmgesteuerte Abfrage der Peripheriegeräte (*Polling*).

Der Programmierer legt Priorität, Antwortzeit und Ablauffolge der Aufgaben fest. Die Entscheidung, welche Aufgabe

Tabelle 4. Pro und Contra der höheren Programmiersprachen (HPS)

Pro	Contra
50...80 % der Gesamtkosten entfallen auf Programmierung. HPS erlauben 5- bis 10mal schnellere Codierung als in Assembler	
Zusätzlicher Speicherbedarf billiger als speichersparende Programmierung	HPS-Programme liefern 30...200 % mehr Code als Assemblersprachen. Die Speicherkosten bestimmen aber immer noch die Gesamtkosten, d. h. an Speicher ist zu sparen
HPS-Programme sind für die meisten Anwendungen ausreichend schnell	HPS-Programme sind langsam, nicht optimiert benötigen sie bis zu 50 % mehr Laufzeit
Wegen Maschinenunabhängigkeit und damit Herstellerunabhängigkeit wird der Programmaustausch erleichtert	Wegen starker Anpassung an die (prozeß-) spezifischen Problemstellungen Programmaustauschbarkeit von geringem Wert
	Für eine Großzahl einfacher Probleme sind HPS zu universell
HPS fördern die Disziplin eines einheitlichen Programmierstils	Bei größeren Systemen ist sowieso eine detaillierte Programmplanung nötig
HPS-Programme sind verständlicher und leichter lernbar (selbstdokumentierend)	Ein Elektroingenieur beherrscht eher seine konkrete Hardware als abstrakte Strukturen
HPS erlauben Problemabstraktion. Der Entwerfer wird von technischen Details entlastet	Der Entwerfer verliert den Bezug zur Maschine bzw. zum gesamten System

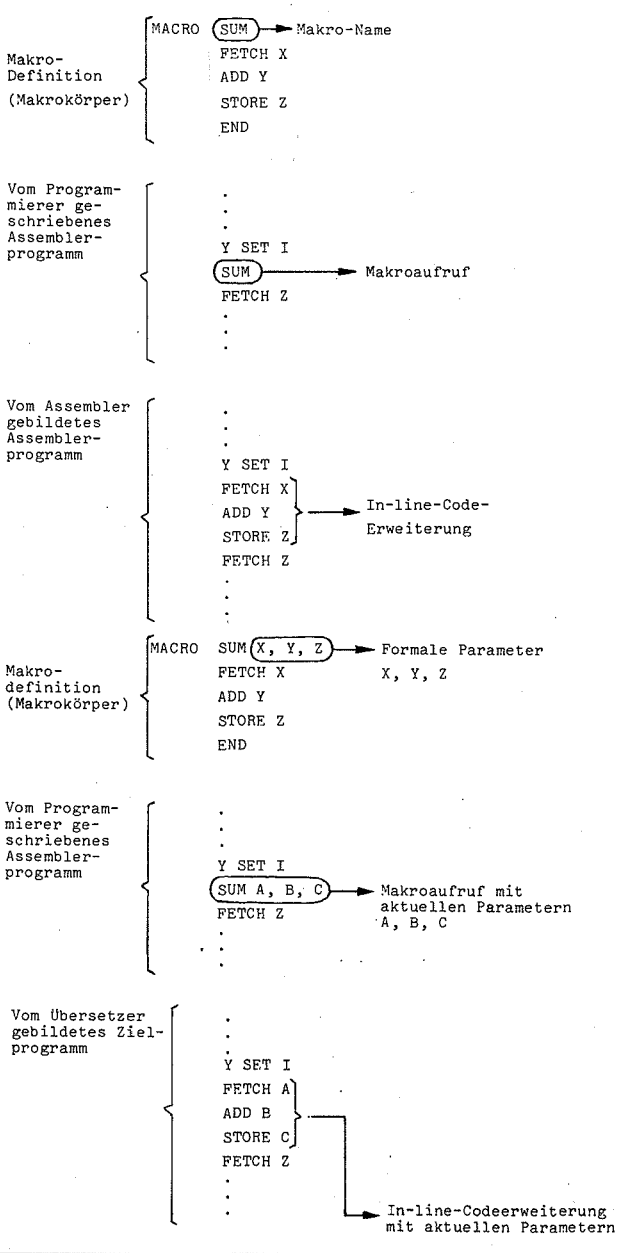
sich in welchem von fünf vorgesehenen Zuständen zu befinden hat, übernimmt ein kleines Zustandsverwaltungsprogramm, das nach Angaben des Graphen in *Bild 10* die korrekte Programmabfolge sicherstellt.

4 Mittelhohe Programmiersprachen

Der plötzliche Entwicklungssprung vom Assembler zur höheren Sprache ist Grund, irritiert zu sein. Das Sprachniveau dazwischen bleibt vernachlässigt. Aber gerade die mittelhohen Sprachen stellen eine Verbindung zwischen Hardware und Software her, da sie einerseits maschinennah genug sind, um effizient zu sein, andererseits hoch genug, um vom Systemprogrammierer akzeptiert zu werden. Diese Sprachen sollen hier – vielleicht nicht ganz zutreffend – unter dem Begriff *Makrosprachen* subsumiert werden.

Makrobefehle (*Makros*) sind Standardprogrammstücke, die sich als gleichbleibende Codesequenzen in ein Zielprogramm beliebig oft durch Quellprogrammaufrufe einfügen lassen. In *Bild 11* werden zwei Beispiele gezeigt. Zur besseren Verständlichkeit ist das erzeugte Zielprogramm in Assemblernotation wiedergegeben. Im zweiten Beispiel dieses Bildes ist ein parameterisierbarer Makro dargestellt. Er hat gegenüber dem im ersten Beispiel den Vorteil, daß die zur Datenübergabe verwendeten Register im Hauptprogramm nicht schematisch starr an die Festlegungen im Makrokörper gebunden sind. Dieser Makro wird zunächst mit formalen, platzhaltenden Namen für Datenquellen und -senken formuliert. Erst im Makroaufruf im Hauptprogramm ist spezifiziert, welches die aktuellen Namen der Register sind. Die funktionale Bedeutung von Makros kann gesteigert werden, wenn innerhalb eines Makros wieder Makroaufrufe möglich sind, wenn als Parameter auch Operationen übergeben und wenn die Parameterlisten beliebig lang oder optional gestaltet werden können.

Die Bildung einer Vielzahl von Makros läuft letztlich auf die Definition einer neuen Programmiersprache hinaus, die den Vorteil besitzt, einen sehr effizienten Code zu erzeugen. Um dies zu belegen, wird ein Programmbeispiel aus [3] angeführt (*Bild 12*). Inhalt des Programms ist ein Vertausch-Sortier-Algorithmus, dessen PL/M-Formulierung dem PL/M-Handbuch von Intel entnommen ist. Dieses Programm wird




```
DECLARE A (256) BYTE,
        (N,I,T1,T2, SWITCHED) BYTE;
SWITCHED=1;
DO WHILE SWITCHED;
  SWITCHED=0;
  DO I = 1 TO N-1;
    T1 = A(I); T2 = A (I+1);
    IF T1>T2 THEN
      DO;
       A (I+1) = T1;
       A (I) = T2;
       SWITCHED = 1;
      END;
    END;
  END;
  EOF
```

PL/M

```
      MVI   D,1
LAB1  MOV   A, D
      ADI   O
      JZ    LAB2
      MVI   D,O
      MVI   H,N.H
      MVI   L,N.L
      MOV   B,M
      MVI   H,ARRAY.H
      MVI   L,ARRAY.L
LAB3  DCR   B
      JZ    LAB1
      MOV   A,M
      INR   L
      CMP   M
      JP    LAB3
      MOV   C,M
      MOV   M,A
      DCR   L
      MOV   M,C
      INR   L
      MVI   D,1
      JMP   LAB3
LAB2  HLT
N     DB    O
ARRAY DS    256
```

Assembler

```
address.H denotes the high-order byte (bits 8-15) of "address"
address.L denotes the low-order  byte (bits O-7)  of "address"
The macro " H,L←address " expands to
  H - address.H
  L - address.L
DECLARE ARRAY (256) BYTE;
DECLARE N BYTE;       /# length of the data in ARRAY  /
D←1;     /# "Switched" flag  /
WHILE (A←D; A  A + o) NOT ZERO;  /# must add O to affect flags /
 DO;
  D←O;
  B←M (N) ;/# macro for H,L←N; B - M; /
  H, L←ARRAY;
  WHILE (B←B - 1) NOT ZERO;
   DO;
    A←M  (HL);
    L←L + 1;
    IF (A : : M (HL) SIGN THEN
     DO;
      C←M (HL);
      M (HL)←A;
      L←L  -  1;
      M(HL)←C;
      L←L +  1;
      D←1;     /# set the flag /
     END;
   END;
  END;
```

Makro

Bild 12. Das Vertausch-Sortier-Programm belegt nach der Übersetzung in PL/M 132 Byte, in Makrosprache 43 Byte und in Assemblersprache 40 Byte

◀ Bild 11. Beispiele für Makrobefehle: unten ein parameterisierbarer Makro, bei dem zunächst nur formale Namen für Datenquellen und -senken formuliert werden

außer in PL/M in einer Makrosprache und in Assemblersprache geschrieben, und der nach der Übersetzung benötigte Speicherplatz wird registriert. Dieser ist für die Makroversion 7 % größer als für die Assemblerversion. Das in PL/M geschriebene Programm benötigt dagegen über 200 % mehr Zielcode, als der Assembler erzeugt.

Über die genannten Vorzüge hinaus haben Makrosprachen noch weitere erwähnenswerte positive Eigenschaften.

1. Sie erlauben die Formulierung von Systemmoduln, die in einer plattenresidenten Programmbibliothek dem Benutzer verfügbar sind. Die wichtigsten Routinen sind die zur Unterstützung der Ein/Ausgabe, zur Datenwandlung und zur Fest- und Gleitpunktarithmetik.
2. Die modernen Steuerstrukturkonzepte der strukturierten Programmierung lassen sich leicht verwirklichen.

5 Assembler gegen höhere Programmiersprachen – eine Wirtschaftlichkeitsanalyse

Von Seiten der Hardwareingenieure werden vorwiegend zwei Argumente gegen höhere Programmiersprachen angeführt, die für den Programmierer gegenläufige Forderungen enthalten:

1. Die Programme müssen kurz, d. h. speicherplatzsparend sein (bei großen Stückzahlen bestimmen die immer noch teuren Speicher die Gesamtkosten).
2. Die Programme müssen schnell sein.

In der konventionellen Datenverarbeitung ist zumindest der erste Punkt längst gelöst. Man vergrößert den zu geringen Speicherplatz quasi unendlich durch dynamisch-organisatorische Maßnahmen wie überlagernde Programmsegmentierung und virtuelle Speichertechniken unter Einsatz einer Speicherhierarchie. Gerade in punkto Programmspeichertechnik unterscheiden sich die Mikrosysteme wesentlich von den „großen" Vorbildern. Wegen der meist gezielten Anwendungen zumindest beim Einsatz als „eingebaute Intelligenz" werden einmal ausgearbeitete Programme in nicht mehr änderbare Festwertspeicher „eingebrannt". Selbstverständlich verhindert dies die dynamische Mehrfachausnutzung von Speicherraum. Die Überschaubarkeit der Mikrosysteme erlaubt aber einen vernünftigen analytischen Ansatz der Optimierung bezüglich der Kosten bei Verwendung einer HPS gegenüber der Programmierung in Assembler [7].

Bild 13 gibt qualitativ den trivialen Tatbestand wieder, daß die Gesamtkosten für Speicher und Programmierung mit der Stückzahl linear wachsen. Bekanntlich machen sich bei großen Stückzahlen die kurzen Assemblerprogramme wegen des geringeren Speicherbedarfs bezahlt, während bei kleinen Produktzahlen die billigeren Programmerstellungskosten mit Compilern den Preis bestimmen. *Bild 14* veranschaulicht dies qualitativ. Der Punkt, der nun interessiert ist, wo die Stückzahlgrenze liegt, ab welcher der Vorteil der Assemblerprogrammierung überwiegt. Der einfache Ansatz zur Ermittlung des Schnittpunktes n_s lautet

$$n_s \cdot \underbrace{L_{ASS} \cdot B_{ASS}}_{1} \cdot S + \underbrace{L_{ASS} \cdot P}_{2} =$$

$$n_s \cdot \underbrace{L_{Comp} \cdot B_{Comp}}_{1'} \cdot S + \underbrace{L_{Comp} \cdot P}_{2'}$$

Es bedeuten: n = Stückzahl; L = Länge des Programms in Zeichen; B = durchschnittliche Anzahl Bits, die von einer Quellprogrammzeile erzeugt werden; S = Speicherpreis pro Bit; P = Programmierkosten pro Quellzeile; ASS: betrifft Assemblerprogrammierung; Comp: betrifft Compilerprogrammierung.

Der Term 1 gibt die Gesamtkosten für Speicher, der Term 2 die einmaligen Programmierkosten an.

Der Zusammenhang zwischen dem Umfang an Zielcode aus der Compilierung und dem aus der Assemblierung wird in einer Gleichung ausgedrückt, die gleichzeitig den Gütefaktor G des verwendeten Compilers definiert:

$L_{Comp} \cdot B_{Comp} = L_{ASS} \cdot B_{ASS} \cdot G.$

Unter Verwendung dieser Gleichung im oben angeführten Ansatz läßt sich ein endgültiger Ausdruck zur Bestimmung der Stückzahlgrenzen finden. Hierbei werden noch folgende realistischen Annahmen gemacht: Die Programmierkosten pro Quellzeile gleich welcher Art setzt man mit ungefähr 25 DM an. Aus einer Assemblerzeile resultieren 16 bit Speicherbedarf, also $B_{ASS} = 16$. Aus einer HPS-Zeile werden etwa 80 bit erzeugt, d. h. $B_{Comp} = 80$. Somit ergibt sich

$$n_s = \frac{25\left(\frac{1}{16} - \frac{G}{80}\right)}{(G\text{-}1) \cdot S} \text{ DM}$$

Tabelle 5 gibt eine Auswahl von Werten für n_s wieder, die sich durch eine vernünftige Variation der Parameter G und S ergeben. Nach einer sehr groben Daumenregel sollte man wie folgt verfahren: n_s liegt bei einer Verwendung der billigeren ROMs bei 1000, bei Einsatz teuerer PROMs bei 100 verkauften Systemen. In dieses Kalkül sollten zur Verfeinerung weitere Überlegungen einbezogen werden. So wächst z. B. die Speicherkostenkurve für ein System nicht linear mit zunehmendem Speicherbedarf, sondern bei Anbruch einer neuen typischen Speichergröße (z. B. 4 K) sprunghaft. Ebenso sind Mengenrabatte nicht berücksichtigt. Unabhängig von diesen Feinheiten zeigen Bild 14 und Tabelle 5 einen Sachverhalt auf, der für die Zukunft der HPS u. a. von Bedeutung ist: Mit abnehmenden Speicherkosten wandert die Stückzahlgrenze immer weiter nach oben. Es scheint für zeitunkritische Anwendungen nur eine Frage der Zeit zu

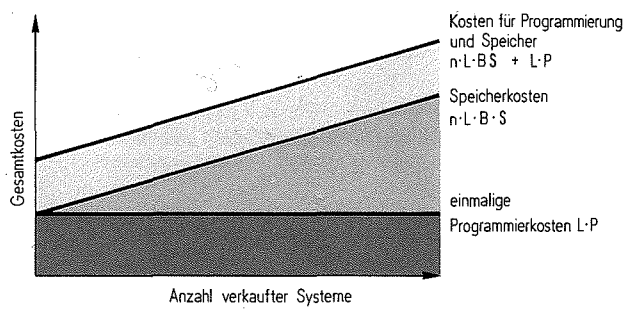


Bild 13. Programmlänge bei Interpretation und Compilation in Abhängigkeit von der Anzahl der Quellprogrammzeilen

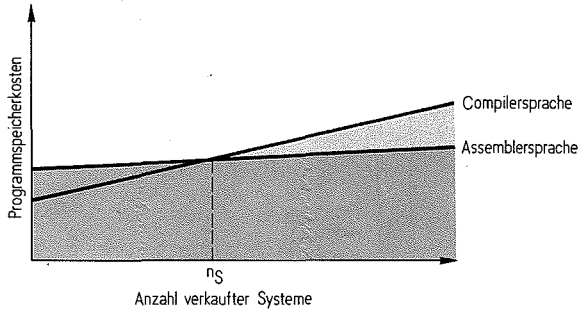


Bild 14. Vergleich der Programmspeicherkosten in Abhängigkeit von der Anzahl der verkauften Systeme

sein, wann HPS wie FORTRAN oder BASIC die Assemblersprachen verdrängt haben werden.

6 Interpretative Systeme

Es mag bisher der Eindruck entstanden sein, daß sich zur Übersetzung höherer Sprachen nur Compiler eignen, die ja bekanntlich die Abbildung in den gewünschten Zielcode in einem separaten sehr aufwendigen Arbeitsgang bewältigen. Moderne Compiler sind so strukturiert, daß sie leicht von einer Rechenmaschine auf die andere zu übertragen sind (*Portabilität*). Meist wird so verfahren, daß ein maschinenunabhängig formuliertes Compiler-Oberteil eine Zwischensprache erzeugt, aus der nun ein weiterer Übersetzer, ein sogenannter Codegenerator, den endgültigen Maschinencode produziert. Wird das Compiler-Oberteil selbst in der Zwischensprache formuliert, so benötigt man letztlich nur noch einen für jedes Zielsystem speziellen Codegenerator, um den gesamten Compiler zu implementieren. Man kann nun noch versuchen, auch den Codegenerator maschinenunabhängig zu gestalten, indem die Eigenart der Zielmaschine und ihr Maschinencode dem anpaßbaren Codeerzeuger per Parameter mitgeteilt werden [11].

Ein weiterer erwähnenswerter Vorteil von Compilern ist ihre Fähigkeit, den Zielcode beschränkt zu optimieren, womit sich eine Effizienzsteigerung bezüglich Speicherbedarf und/oder Programmlaufzeit erzielen läßt.

Als Nachteil ist zu vermerken, daß in den von Compilern erzeugten Zielprogrammen sehr viele Unterprogrammaufrufe zu entdecken sind. Die Erklärung ist einfach: Eine große Anzahl immer wieder benötigter Standardprogrammstücke werden sinnvollerweise als einmal abgefaßte Systemmoduln in einer residenten Programmbibliothek geführt und bei Bedarf zur Laufzeit herbeigeholt und aus dem Hauptprogramm heraus angesprungen. Die Handhabung von Unterprogrammen bringt zur Ausführungszeit einen erklecklichen Anteil an programmierter Verwaltung und somit an Zeitverlusten mit sich.

Neben dem Compiler existiert ein anderer Typ von Übersetzer: der Interpretierer. Er kennt keinen separaten Übersetzungslauf, sondern führt eine erkannte, d. h. interpretierbare Programmeinheit sofort aus, bevor er den weiteren Programmtext untersucht. Interpretierbare Sprachen müssen ganz bestimmten „grammatikalischen" Regeln genügen. Als Beispiel sei BASIC als die populärste interpretierbare Sprache erwähnt. Der Nachteil liegt auf der Hand: Interpretierer müssen bei jedem Programmdurchlauf aufs neue den Programmtext erkennen. Jedesmal wenn eine Adresse zu berechnen oder eine Sprungadresse zu ermitteln ist, wird dieser Vorgang wiederholt.

Nicht nur höhere, auch und gerade maschinennahe Sprachen sind interpretierbar. Bei Anwendung der schon früher erwähnten Mikroprogrammierung zur hardwareunmittelbaren Interpretation von Maschinencode kann so auf die Technik des Unterprogrammaufrufs verzichtet werden. Es läßt sich zeigen, daß bei zunehmendem Programmumfang der von Unterprogrammen befreite Code trotz aller bekannten Nachteile insgesamt weniger Aufwand an Speicher benötigt als ein mit Unterprogrammen gestalteter Zielcode (*Bild 15*).

Es ist nicht einzusehen, wieso ein mikroprogrammierter Interpretierer nicht auch auf höheren interpretierbaren Sprachen operieren können sollte. Tatsächlich wird dieses Konzept schon in einigen mikroprogrammierbaren Computern mit Erfolg praktiziert. Den bisher etwas stiefmütterlich behandelten „interpretierenden Übersetzern" darf eine erfolgreiche Zukunft vorhergesagt werden, da der Entwicklungsweg in der Mikroprozessortechnologie auf mikroprogrammierbare Systeme mit extrem schnellen Zykluszeiten ausgerichtet ist. Die Technologie wird also helfen, die Schnittstelle zwischen Hardware und Software nach „oben" zu den höheren interpretierbaren Sprachen hin zu verschieben.

Tabelle 5. Stückzahlgrenzen (n_s) bei verschiedenen Werten für G und S

Speicherkosten pro Bit (in Pf.) \ Gütefaktor des Compilers		1,1	1,25	1,5	2,0
↑	0,125	9750	3750	1750	750
ROMs	0,25	4875	1875	875	375
	1,25	975	375	175	75
	2,50	488	188	88	38
PROMs	3,75	325	125	58	25
↓	5,00	244	94	44	19

7 Zusammenfassung

Das zuletzt angetippte Thema ist zu weit und zu ergiebig, um hier als „die" Perspektive der Zukunft ausführlich dargestellt zu werden. Die Tendenz ist jedoch aufgezeigt. In der anhaltenden Phase des „Nachvollziehens" schon entwickelter Softwaretechniken darf man in diesen Tagen von den Herstellern und Softwarehäusern folgende Produkte erwarten:

- „mittelhohe" Systemprogrammiersprachen,
- Programmbibliotheken auf Floppy Disk oder Platten,
- Standard-Systemmoduln,
- Betriebssysteme, insbesondere Echtzeitbetriebssysteme,
- Echtzeitprogrammiersprachen.

Derartiger Komfort verlangt eine Kopplung zwischen Mikro- und Gastrechner oder Entwicklungssystem, die aus Mikrorechner, Floppy-Disk, PROM-Programmierer, alphanumerischer Tastatur, Drucker und/oder Bildschirm bestehen. Der Preisrahmen hierfür wird bei 40 000...60 000 DM liegen. Ein Ergebnis dieses Beitrags sollte die Empfehlung sein, eher eine solche Investition zu tätigen, als mit der langfristig wesentlich teureren „Primitivprogrammierung" zu arbeiten. Höhere Programmiersprachen werden sich trotz der Vorbehalte seitens der Hardwarefachleute durchsetzen, und zwar um so mehr, je „effizienter" sich aus ihnen Code erzeu-

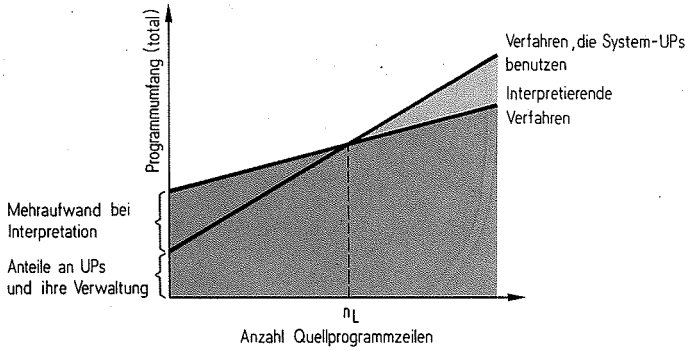


Bild 15. Programmumfang in Abhängigkeit von der Anzahl der Quellenprogrammzeilen

gen läßt. Die zukünftigen Techniken vereint mit der Eigenschaft der Mikroprogrammierbarkeit werden diesen Trend unterstützen.

Literatur

[1] *Gößler, R.* und *Schwerte, J.*: Auf dem Weg zur Mikroprozessor-Praxis. ELEKTRONIK 1976, H. 3, S. 74 ff; ELEKTRONIK-Report.

[2] *Watson, I. M.*: Comparison of Commercially Available Software Tools for Microprocessor Programming. Procedure of the IEEE, Vol. 64, Juni (1976), H. 6, S. 910...920.

[3] *Popper, Ch.*: SMAL – A Structured Macro-Assembly Language for a Microprocessor. COMPCOM Digest of Papers (1975), S. 147...151.

[4] *D'Agapeyeff, A.*: Micro Software – A Re-Incarnation of Problems and Potential. Small Systems Software, Vol. 1 (1976), H. 2, S. 2...5.

[5] *Bass, Ch., Brown, D.*: A Perspective on Micro-Computer Software Proceedings of the IEEE, Vol. 64, Juni (1976), H. 6, S. 905...908.

[6] *Cassell, D. A., Mayhew, W. F.*: The Current State of the Art in Microprocessor Software. CACM 10 (1975), S. 119...121.

[7] *Gibbons, J.*: When to use hiher – level languages in microcomputer systems. Electronics 7, August (1975), S. 107...111.

[8] *Holt, Pokoski, Cordell*: A Software Development System for Micro-Computers. IEEE Transactions on Industrial Electronics..., Vol. IECI-22, Nr. 3, August (1975), H. 3, S. 279...282.

[9] *Hennessy, Kieburtz, Smith*: TOMAL: A Task Oriented Applications Language. IEEE Transactions on Industrial Electronics..., Vol. IECI-22, August (1975), H. 3, S. 283...289.

[10] *Sohrabji, N.*: Macro Processor Simplifies Microcomputer Programming. Computer Design, August (1976), S. 108...197.

[11] *Pelz, K.*: Software-Situation der Mikroprozessoren. Unveröffentlichte Notiz des Physikalischen Instituts der Universität Erlangen–Nürnberg.

[12] *Kidall, G. A.*: Microcomputer Software Design – A checkpoint. National Computer Conference, Mai (1975), S. 99...106.

[13] *Eymeren, M. v., Heinzel, W.*: Implementierung einer arithmetischen UP-Bibliothek am Beispiel des Mikrorechnersystems INTELLEC 80. Regelungstechnische Praxis (1976), H. 4, S. 101 ff.

[14] *Sidline, A.*: Operating System Software-Interfaces for a Microprocessor. COMPCON FALL 1975, Sept. (1975), S. 166...169.

[15] *Weatherhead, J.*: Programming Microprocessors. Software World Vol. 6, H. 9, S. 7...11.

[16] Rubrik MICROCOSM in Small Systems Software. Vol. 1 (1976), H. 3, S. 15...22.

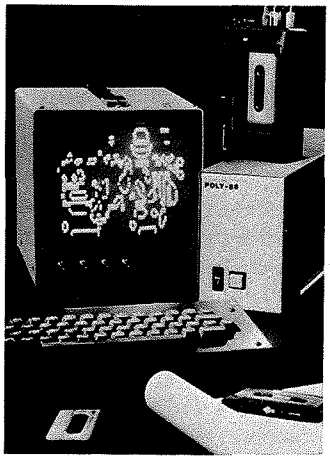
[17] *Martinez, R.*: A Look at Trends in Microprocessor/Minicomputer Software Systems. Computer Design, Juni (1976), S. 51...57.

Dipl.-Inform. Günter R. Koch, geboren in Freiburg/Br., studierte Elektrotechnik an der TH Karlsruhe bis zum abgeschlossenen Vordiplom. Danach wechselte er zur „Informatik", die in Karlsruhe als erste Fakultät der Bundesrepublik eingerichtet wurde. Seit 1975 ist er wissenschaftlicher Mitarbeiter am ersten Informatik-Anwendungslehrstuhl (Planungs- und Programmiertechniken von Prozeßrechnern) in Karlsruhe und ist dort zuständig für anwenderorientierte Echtzeitprogrammierung und Programmiersprachen.
Hobbys: Alte Uhren, Feinschmeckerküche, Wohnungseinrichtung
Telefon: 07 21/6 08–36 56
Gelegentlicher ELEKTRONIK-Leser seit 1974

Preiswertes Entwicklungssystem für den Mikroprozessor 8080

Unter der Bezeichnung POLY 88 bietet die Firma Digitronic einen kompletten Mikrocomputer mit Peripheriegeräten an. In dieser Konfiguration ist die Gerätezusammenstellung bereits als arbeitsfähiges Entwicklungssystem verwendbar, das auf der Basis des Mikroprozessors 8080 aufgebaut ist. Auf der Platine der Zentraleinheit ist ein Arbeitsspeicher mit 512 Worten untergebracht, und es besteht außerdem die Möglichkeit, 3 KBytes eines EPROMs einzusetzen. Die Dateneingabe erfolgt über eine voll ausgebaute Tastatur; als externer Massenspeicher fungiert ein Kassetten-Tonbandgerät, und zur Ausgabe ist ein Datensichtgerät vorgesehen. Der Preis für das gesamte System liegt unter DM 5000.–.

□ Hersteller: Digitronic Computersysteme GmbH, Bei der Doppeleiche 3–5, 2000 Wedel, Tel. (0 41 03) 73 93.



Testhilfsspeicher für Mikrocomputer

Der Testhilfsspeicher der Fa. Schmitt & Drösel ist ein einfaches, preiswertes und leistungsfähiges Gerät für die Entwicklung und Erprobung von anwendungsspezifischen Mikrorechnern. Er läßt sich an alle Mikroprozessoren anschließen, deren Signale die auf dem Datenblatt gezeigten Bedingungen erfüllen. Er vertritt während der Erprobungsphase den späteren Einsatzfestwertspeicher des Mikrorechners, ist selbst jedoch im wesentlichen ein Schreiblesespeicher und damit dem Teststadium, in dem Programme noch häufig geändert werden, besser angepaßt. Die entscheidende Eigenschaft des Testhilfsspeichers ist es, daß sein Schreiblesespeicher während der Testphase genau den gleichen Adreßraum einnimmt wie der spätere Einsatzfestwertspeicher und somit ein adressengetreues Testen der späteren Festwertspeicherprogramme gestattet. Zum Laden von Programmen und für den sonstigen in der Testphase notwendigen Dialogbetrieb besitzt er Einrichtungen für den Anschluß einer Teletype-Schreibmaschine sowie einen programmierbaren Festwertspeicher zur Bereitstellung des Ladeprogramms und anderer Hilfsprogramme.
Die Grundstufe des Testhilfsspeichers umfaßt 2 KByte statischen Schreiblesespeicher, 512 Byte reversibel programmierbaren Festwertspeicher und den Datenpuffer für eine Teletype-Schreibmaschine. Die Grundstufe ist auf 2 Doppel-Europakarten untergebracht. Der Schreiblesespeicher kann in Stufen zu je 2 KByte bis auf 16 KByte erweitert werden, der Festwertspeicher in Stufen zu je 512 Byte bis auf 2048 Byte. Für den Kartenanschluß sind Kontaktstekker vorgesehen, auf Wunsch wird der Testhilfsspeicher in einem besonderen Gehäuse geliefert.

Mit der Bestellung werden folgende Informationen benötigt:

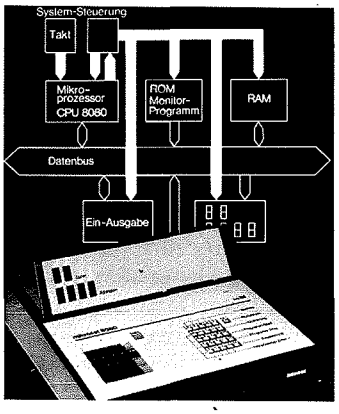
a) Adreßraum für den Schreiblesespeicher während des Testens.
b) Adresse für den Teletype-Datenpuffer als Binärzahl auf den Adreßleitungen A7...A2. Die gewünschte Binäradresse darf nicht mehr als 3 Nullbits enthalten, vorgesehene Standardadresse ist A7...A2 = 100110.
c) Verfügbarer Wert der Taktfrequenz F (auf 3 Prozent genau).

□ Hersteller: Schmitt & Drösel, Birkenauer Talstr. 10, 6940 Weinheim, Tel. (0 62 01) 6 42 49.

Mikrocomputer-Lernsystem von Siemens

Weniger als DM 2000.– kostet der Mikrocomputer „Mikroset 8080", den die Firma Siemens speziell für ambitionierte Hobby-Elektroniker anbietet. Es handelt sich dabei um ein Entwicklungssystem, das auf der Basis des Mikroprozessors SAB 8080 aufgebaut ist. In einem ROM mit einer Kapazität von 1 KByte ist ein Systemprogramm zum Testen von eigenen Programmen festverdrahtet. Ferner ist ein RAM mit 512 8-bit-Worten enthalten. Außer der Eingabetastatur, den Ziffernanzeigen und dem Anschluß für einen Kassettenrekorder enthält das Gerät noch die komplette Stromversorgung. Es sind Erweiterungsmöglichkeiten vorgesehen, die die Ausstattung mit Ein-/Ausgabe-Kanälen und mit einer Programmiereinrichtung für EPROMs ermöglichen. Weiterhin sind ein Druckeranschluß und ein Interface für Datensichtgeräte geplant.

□ Hersteller: Siemens AG, ZVW-104, Postfach 103, 8000 München 1.



Wir haben etwas gegen Ihre Probleme: Individuelle Lösungen.*

*(Das Ei des Kolumbus, wenn Sie es wollen)

Wir fertigen und liefern Hardware

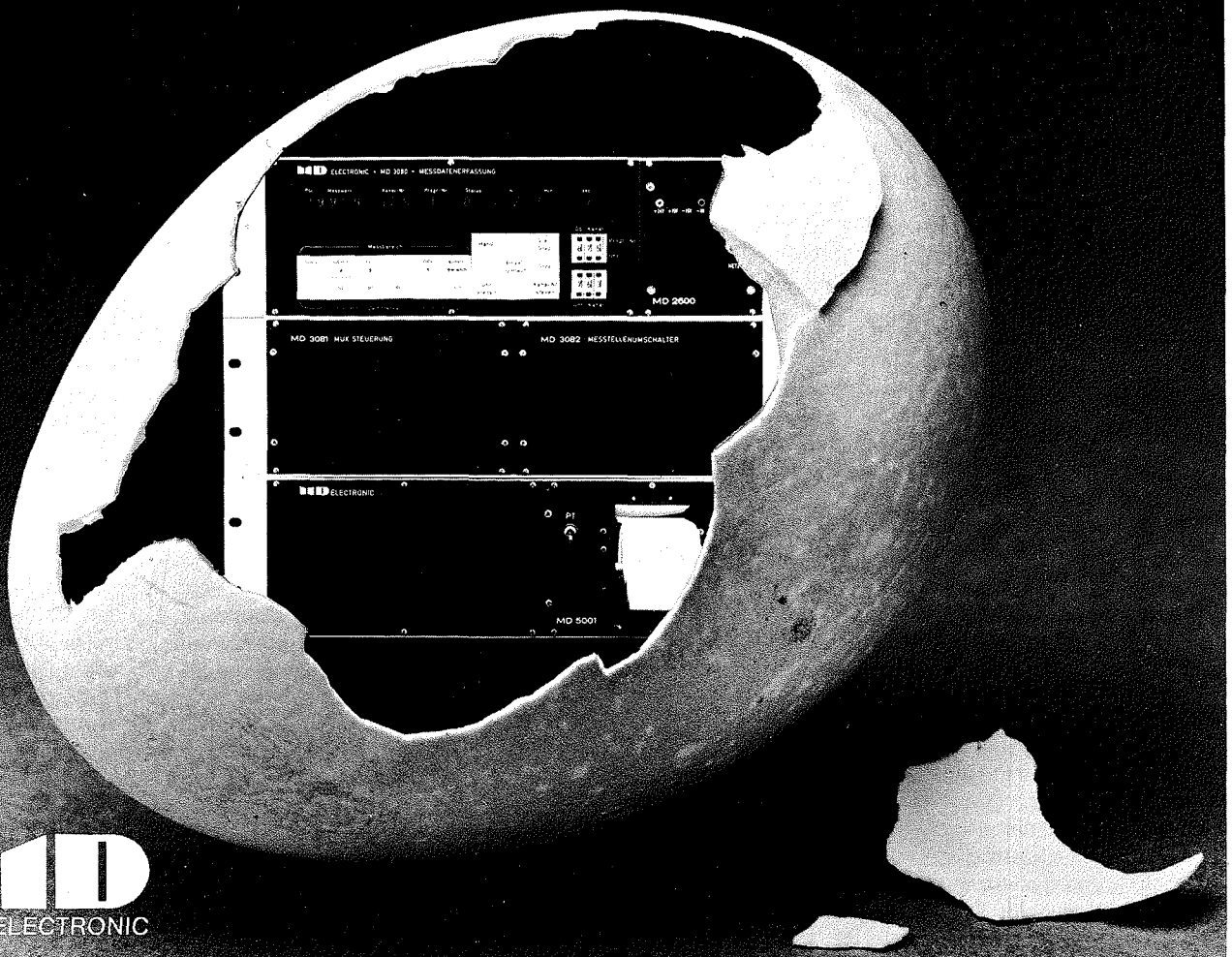
Z 80 modulares Mikrocomputersystem **PROC/80 Z** auf Einfach-Europakarte.
F 8 Single Board Mikrocomputer **PROC/80F** Einfach-Europakarte und **PROC 80 LSI** auf Großplatine.
Zu beiden Mikrocomputern liefern wir das Kartensystem MD 2000: Analog / Digital-Wandler, Digital / Analog-Wandler. Meßstellenumschalter, Meßverstärker, Optogetrennte Ein- / Ausgabekarten.

Wir entwickeln Software

Für folgende Computer: PDP 11, LSI 11, Z 80, F 8, 8080, 8X 300.

Wir bauen Anlagen und Automatisierungssysteme

Zur Meßdatenerfassung und Prozeßsteuerung in Industrie und Forschung.
Wir entwickeln Ihre Geräte für den Markt von morgen mit dem Bauelement MICROCOMPUTER, das unsere Welt verändern wird. Dabei zahlt sich unsere 5jährige Erfahrung für Sie aus.
Sprechen Sie mit MD Electronic – dem Mikrocomputersystem-Haus.



MD ELECTRONIC

Münzer + Diehl Electronic GmbH

Schloßstraße 2, 5060 Bergisch Gladbach 1
Telefon 02204 / 51061, Telex 8878471
Wir stellen aus: INTERKAMA 77, Halle 10, Stand 10 A 34

Dipl.-Ing. Frank Heubach

Strukturierte Programmierung auch bei Mikrocomputern

Nach den Großrechnern und den Minicomputern kam mit den Mikrocomputern eine dritte Rechnerkategorie, die nach der Meinung zahlreicher Fachleute zum großen Teil dieselben Entwicklungsphasen durchmachen wird wie ihre beiden Vorgänger. Das gilt vor allem für die Software, und es ist deshalb naheliegend, Methoden der Groß-EDV auch für Mikrocomputer in Betracht zu ziehen – z. B. die strukturierte Programmierung. An einfachen Beispielen soll gezeigt werden, womit sich Mikrocomputer-Programmierer in Zukunft auseinanderzusetzen haben.

Während der Kostenanteil der Software an den Gesamtkosten eines Rechensystems 1955 weniger als 20 % und 1965 etwa 45 % betrug, wird er 1985 auf 85...90 % angestiegen sein.

Die mittlere Programmierleistung (in Maschinenbefehlen pro Mannmonat) lag 1955 (Programme in Maschinensprache) bei etwa 200, 1970 (Fortran u. ä.) bei 650 und wird 1985 (bei Anwendung der Methoden der strukturierten Programmierung) bei 2400 liegen (dabei ist ein Streubereich nach oben und nach unten jeweils um den Faktor 5 angenommen).

Bei Rechenzentren entfallen heute etwa 40 % der gesamten Softwarekosten auf die Wartung von Programmen. In dieser Wartung ist Fehlerbeseitigung in laufenden Programmen, Übernahme von Fremdsoftware und Anpassen von Programmen an neue Randbedingungen (z. B. veränderte Aufgabenstellung, neue Betriebssystem-Software, andere Rechner) enthalten.

Die Veränderung der Kostenanteile wird sowohl durch die gegenläufige Kostenentwicklung (komplexere Hardware wird kostengünstiger, Programmierleistung immer teuerer) als auch durch die ständig zunehmende Komplexität der Programme hervorgerufen. Bei der Steigerung der Programmierleistung sind die jeweils verwendeten Sprachebenen und Entwurfshilfsmittel ausschlaggebend. Die Zahlen [1, 7] machen deutlich, daß die Frage, wie man gute Software ökonomisch herstellen kann, ein wesentliches Problem aller Zweige der Datenverarbeitung darstellt.

1 Qualitätsmerkmale der Software

Nach heutigen Vorstellungen [6, 7] ist gute Software

- benutzerfreundlich,
  der Benutzer kann das Programm mit wenigen Eingaben zu den gewünschten Aktionen veranlassen,
- fehlerfrei,
  die Software erfüllt (auch in Ausnahmesituationen) ihre Aufgabe,
- leicht auf Fehlerfreiheit zu überprüfen,
  die Richtigkeit des Software-Entwurfs kann schon anhand der Planungsunterlagen und des Programmtextes nachgewiesen werden.
- einfach zu testen,
  bei der Planung der Software schon mit entworfene Tests sind einfach durchzuführen und erlauben sichere Aussagen über die Funktionstüchtigkeit der implementierten Software,
- einfach zu warten und leicht an geänderte Problemstellungen und Hardwarekonfigurationen anzupassen,
  durch übersichtlichen Entwurf können zu modifizierende Stellen schnell gefunden werden, und die Änderungen führen zu keinen unerwarteten Nebeneffekten,
- gut dokumentiert,
  die Dokumentation wird nicht nach Abschluß der Softwareentwicklung in Angriff genommen, sondern sie hält Schritt mit dem Fortgang der Entwicklung.

Der heute im Bereich der Mikroprozessoren noch manchmal beschrittene Weg, die Software „erst mal zum Laufen zu bringen" und sie dann mit Hilfe von Tests und Korrekturen in Richtung der ursprünglichen Spezifikationen zu „trimmen", ist weit von den eben genannten Vorstellungen entfernt. Bei großen Programmen kann mit solchen Tests oft nur ein verschwindend kleiner Teil der möglichen Abläufe untersucht werden. Da sich durch Testen nur die Anwesenheit, nicht aber die Abwesenheit von Fehlern nachweisen läßt (Dijkstra), sollte sich die Überprüfung der Richtigkeit eines Programms schon anhand der Entwurfsunterlagen, auch anhand des Programmtextes selbst, durchführen lassen. Programmtexte und die üblicherweise zur Programmplanung verwendeten Programmablaufpläne sind jedoch nicht oder nur sehr schlecht für diese Überprüfung brauchbar. Das liegt daran, daß einmal der dynamische Ablauf des Programms aus ihnen nicht zu erkennen ist. Es kann z. B. sein, daß in einem Programm, das aus vielen Verzweigungen besteht, im wesentlichen eine Schleife mehrfach durchlaufen wird und die vielen Verzweigungen nur der Behandlung von selten auftretenden Sonderfällen dienen. Zum anderen treten in ihnen zwei Dinge miteinander vermischt auf: die prinzipielle Lösung des Problems (der Algorithmus) und die Umsetzung dieser Problemlösung in ein Programm unter Benutzung bestimmter Sprachelemente (die Implementierung).

Die hier skizzierten Schwierigkeiten werden im wesentlichen durch ein Element unserer Programmiersprachen und des Programmablaufplans hervorgerufen: durch den Sprung, besonders durch den bedingten Sprung, die Verzweigung. Im Programmablaufplan kann man eine Programmschleife nur über eine Verzweigung verlassen. Für

Fallunterscheidungen, für die Behandlung von Sonderfällen oder zur Fehlerbehandlung wird der gleiche Mechanismus verwendet. In den bisher genannten Fällen hatte die Verzweigung eine Funktion in der Problemlösung, im Algorithmus. Die gleiche Verzweigung kann aber an einer anderen Stelle lediglich zu Zwecken der Implementierung verwendet werden, z. B. um eine Mehrfachausnutzung von Programmteilen zu ermöglichen. Diese unterschiedlichen Funktionen der Verzweigung sind aus dem Programmtext und dem Programmablaufplan nicht zu ersehen [4].

1.1 Modular aufgebaute Programme

Schon Mitte der sechziger Jahre galt die Aufteilung eines Programms in kleine Moduln, die jeweils eine überschaubare Teilaufgabe erledigten, als ein wesentliches Kennzeichen guten Programmierstils. Schon damals wurde verlangt, daß ein solcher Programm-Modul nicht länger als eine oder zwei Druckerseiten (etwa 50 bis 100 Zeilen) sein sollte. Ein scheinbar ziemlich einfaches, aus sechs Moduln aufgebautes, Programmstück zeigt *Bild 1*. Bei genauerer Betrachtung stellt man fest, daß 34 verschiedene Wege vom Eingangsmodul E zum Ausgangsmodul A führen, falls die äußere Rückführungsschleife nicht durchlaufen wird. Bei nur zweifachem Durchlaufen der äußeren Schleife ergeben sich bereits mehr als 39 000 verschiedene Wege. Trotz des modularen Aufbaus wird hier durch die starke Vermaschung der Moduln das Testen und das Einarbeiten in die Wirkungsweise dieses Programmstücks sehr erschwert.

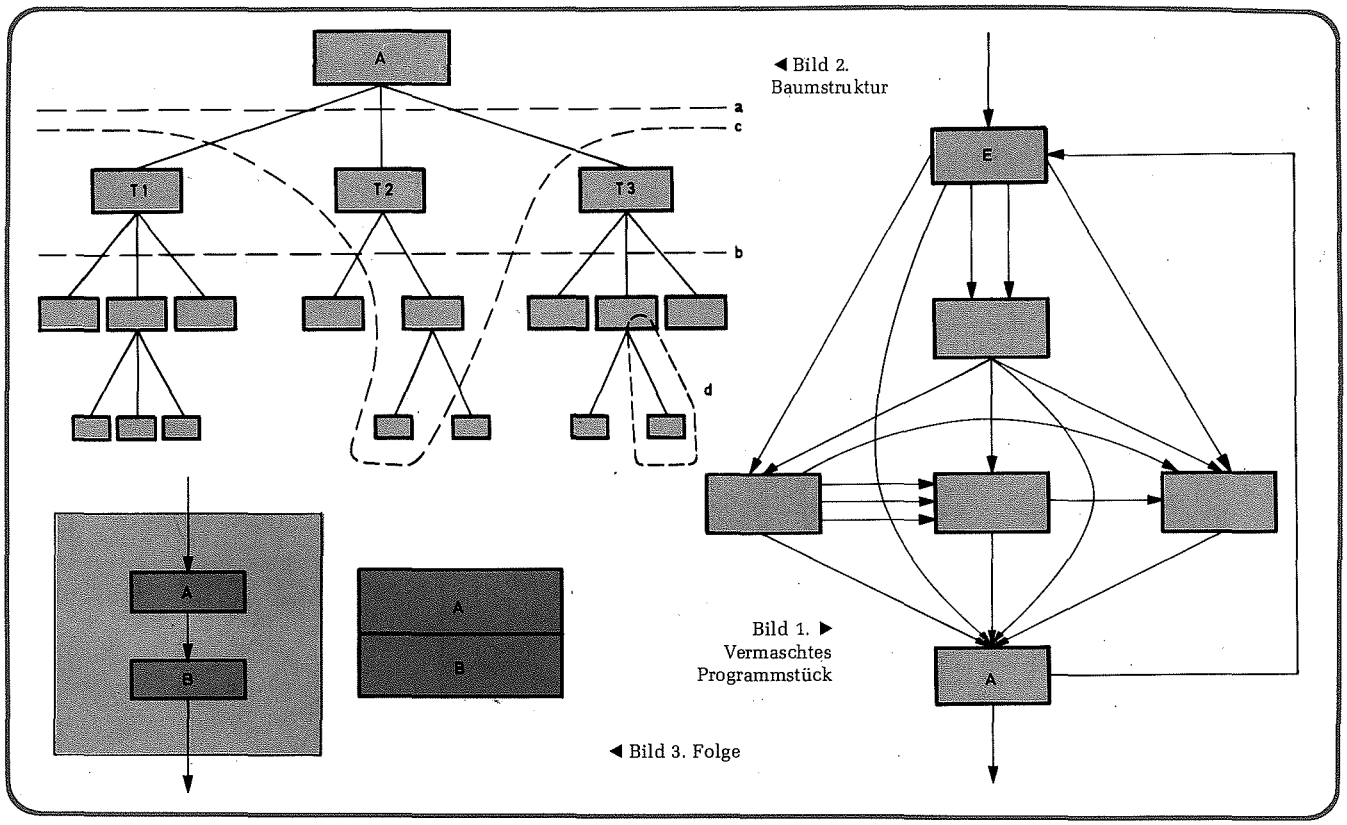
Man erkannte ebenfalls bei dem Versuch, die Richtigkeit von Programmen mit formalen Methoden zu beweisen, daß die Schwierigkeit eines formalen Beweises nicht nur von der Aufgabe, sondern auch von der Komplexität der Struktur des betrachteten Programms abhängt. Besondere Schwierigkeiten ergaben sich auch hier für umfangreiche Programme und stark vermaschte Programme, d. h. für modular aufgebaute Programme mit sehr vielen Hin- und Hersprüngen zwischen den einzelnen Programm-Moduln. Für die praktische Programmierarbeit bedeutet das, daß man nur dann die Richtigkeit eines Programms leicht nachweisen kann, wenn es folgende Eigenschaften aufweist:

- das Programm besteht aus vielen kleinen, überschaubaren Moduln,
- diese Moduln sind sehr einfach strukturiert,
- die Zahl der Verbindungen zwischen den Moduln ist sehr gering.

2 Methode der strukturierten Programmierung

Diese Erkenntnisse führten zur Methode der strukturierten Programmierung [3]. Durch „schrittweise Verfeinerung" (*top-down-design*) wird dabei eine Aufgabe mit Hilfe von nur drei Strukturelementen in immer kleinere Teilaufgaben unterteilt. Diese Zerlegung wird so lange fortgesetzt, bis sich Teilaufgaben überschaubarer Größe ergeben, die mit der verwendeten Programmiersprache einfach programmiert werden können. *Bild 2* zeigt ein Beispiel für die Vorgehensweise bei der „schrittweisen Verfeinerung". Aufgabe A wird in drei Teilaufgaben T1, T2 und T3 unterteilt. Diese Teilaufgaben werden wieder unterteilt und so fort. Das auf diese Weise entstehende Programm weist eine Baumstruktur auf. Jedes Rechteck stellt einen Programm-Modul für die jeweilige (Teil-)Aufgabe dar.

Die Baumstruktur ist die Struktur mit der minimal möglichen Zahl von Verbindungen zwischen den Programm-Moduln. Eine Baumstruktur für ein Programm aus n Moduln weist nur n-1 Verbindungen auf, während z. B. für eine vollständig vermaschte Struktur (jeder der n Moduln ist mit den übrigen n-1 Moduln direkt verbunden) $\frac{n}{2}$ (n-1) Verbindungen erforderlich sind. Bei der Baumstruktur gibt es jeweils nur eine einzige Verbindung zwischen einem Modul und dem ihm übergeordneten (auf der nächsthöheren Ebene) und einem ihm untergeordneten Modul (auf der nächsttieferen Ebene). Somit gibt es auch nur einen einzigen Pfad (eine Folge von Verbindungen) zwischen dem Modul A auf der obersten und einem Modul auf der untersten Ebene. Jeder



◀ Bild 2. Baumstruktur

Bild 1. ▶ Vermaschtes Programmstück

◀ Bild 3. Folge

Modul steuert die Zusammenarbeit aller ihm auf der nächsttieferen Ebene zugeordneten Untermoduln. Die Untermoduln selbst besitzen keine Verbindungen untereinander. Der Modul auf der obersten Ebene wird wegen seiner steuernden Funktion für das gesamte Programm auch Steuermodul genannt.

Ein Vorteil der schrittweisen Verfeinerung und der daraus entstehenden Baumstrukturen liegt darin, daß die zu lösende Aufgabe (oder auch die fertige Lösung der Aufgabe) in allen Stadien der Entwicklung und auf allen Ebenen der Detaillierung vollständig beschrieben ist. Die gestrichelten Linien a,b,c stellen drei Beispiele für die Betrachtung eines derartigen Systems dar: Entlang der Linie a ergibt sich ein globaler Überblick über die zur Lösung der Aufgabe A erforderlichen Teilaufgaben, entlang der Linie b wird auch die Struktur dieser Teilaufgaben deutlich. Linie c gibt z. B. an, welche Informationen benötigt werden, um eine Änderung an einem der Programmteile auf der untersten Ebene vorzunehmen. Neben der globalen Kenntnis der Aufgaben aller übrigen Programmzweige sind nur Detailinformationen über den Pfad erforderlich, auf dem der interessierende Programmteil vom Steuermodul A aus erreicht wird. Die geschlossene Linie d schließlich gibt an, wo sich Änderungen in einem solchen System unter der Voraussetzung auswirken, daß derartige Änderungen bereits beim Entwurf der Programmstruktur berücksichtigt wurden.

2.1 Die Strukturelemente

Zur Durchführung der schrittweisen Verfeinerung und zum Aufbau der Baumstruktur des Programms werden bei der strukturierten Programmierung lediglich drei einfache Strukturelemente (sogenannte Strukturblöcke) verwendet, dies sind
- Folge
- Schleife,
- Verzweigung.

Schon 1966 hatten Böhm und Jacopini [2] gezeigt, daß sich mit diesen Strukturelementen jeder Algorithmus darstellen läßt. Die drei Strukturblöcke werden im folgenden in zwei Darstellungsweisen nebeneinandergestellt, auf der linken Seite mit den Symbolen des Programmablaufplans und auf der rechten Seite als Struktogramm oder Nassi-Shneidermann-Diagramm (nach den Autoren [5], auf die diese Darstellungsweise zurückgeht). Während die Reihenfolge einzelner Elemente des Programmablaufplans durch die Richtung der Flußlinien festgelegt wird, werden Struktogramme immer von oben nach unten durchlaufen. Der zu einem Strukurblock gehörige Ablaufplan ist durch ein Rechteck eingerahmt. Jedes in den Strukturblöcken mit Großbuchstaben gekennzeichnete Rechteck steht für einen weiteren Strukturblock. Auf diese Weise entsteht eine Schachtelung von Strukturblöcken. Wie aus der Darstellung als Programmablaufplan hervorgeht, besitzt jeder dieser drei Strukturblöcke nur einen Eingang und einen Ausgang.

1. Folge (*Bild 3*)

Ein Strukturblock dient zur Beschreibung einer Aktion. Mehrere Strukturblöcke können zu einer Folge von nacheinander zu durchlaufenden Blöcken aneinandergesetzt werden. Diese Folge stellt selbst einen Strukturblock dar.

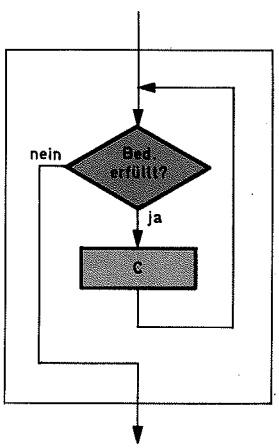
2. Schleife (*Bild 4*)

Bei der Schleife wird ein Strukturblock so lange wiederholt ausgeführt, wie die Wiederholungsbedingung erfüllt ist.

3. Verzweigung

a) Auswahl zwischen zwei Möglichkeiten (*Bild 5*).

b) Auswahl zwischen mehreren Möglichkeiten (Fallunterscheidung, *Bild 6*).



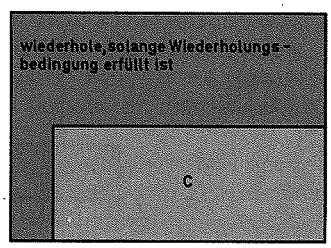
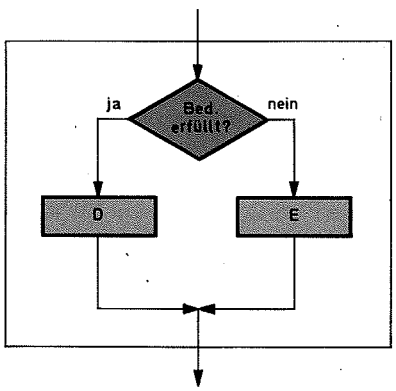


Bild 4. Schleife



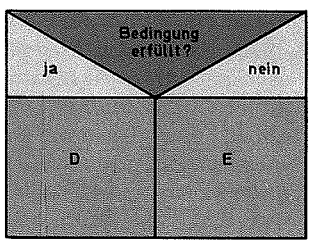
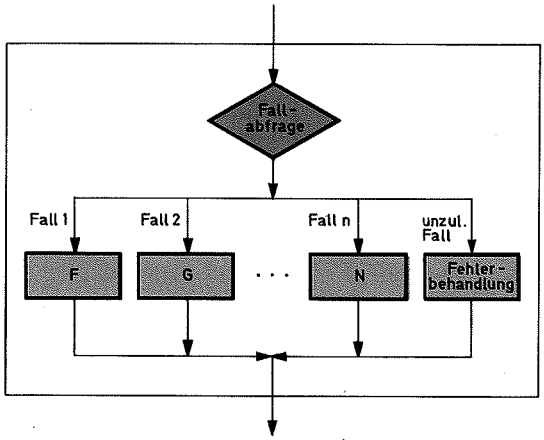


Bild 5. Verzweigung



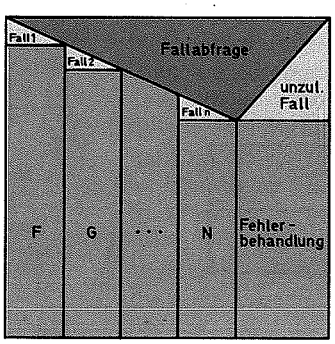
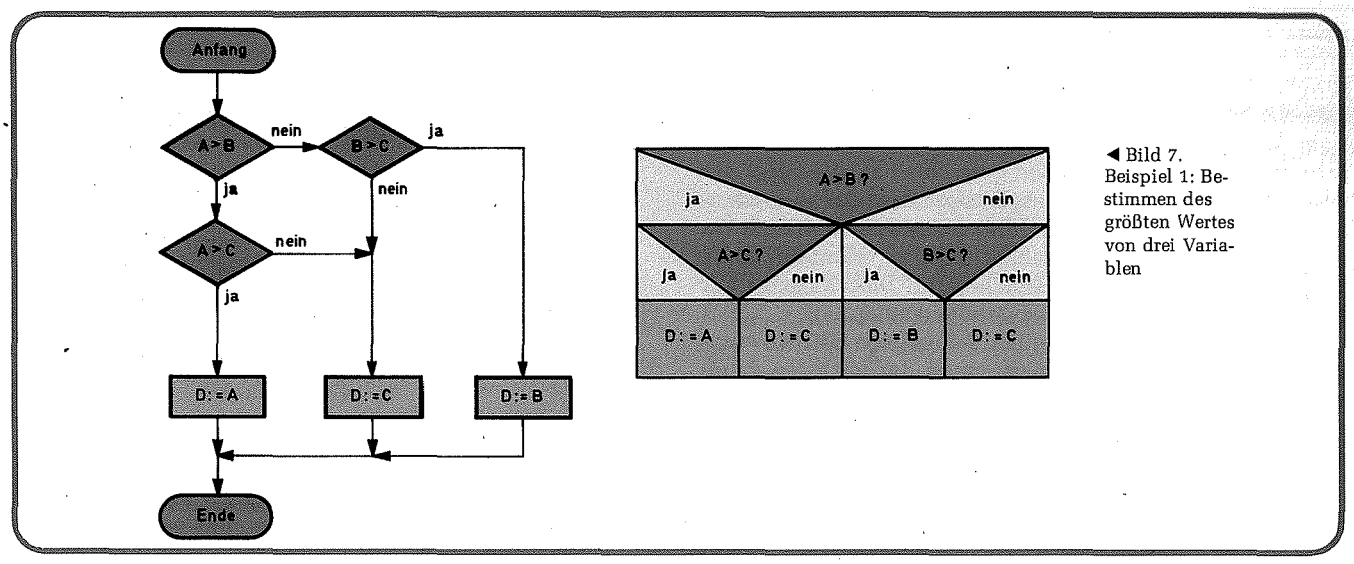


Bild 6. Fallunterscheidung



◀ Bild 7. Beispiel 1: Bestimmen des größten Wertes von drei Variablen

2.2 Realisierung der Strukturblöcke

Ein Strukturblock kann bei der Umsetzung des Struktogramms in ein Programm je nach den zur Verfügung stehenden Mitteln z. B. als Block einer blockorientierten Sprache (PLUS, PL/M), als Unterprogramm oder als einfaches Programmstück mit Verzweigungen etwa in der Art realisiert werden, wie es bei der Vorstellung der elementaren Strukturblöcke im Programmablaufplan gezeigt wurde. Von der Sprache abhängig ist auch die Art der Festlegung der Schnittstelle zwischen dem übergeordneten und dem ihm untergeordneten Strukturblock, speziell hinsichtlich der Vermeidung von Nebenwirkungen durch gezielte Auswahl der an der Schnittstelle zu übergebenden Daten.

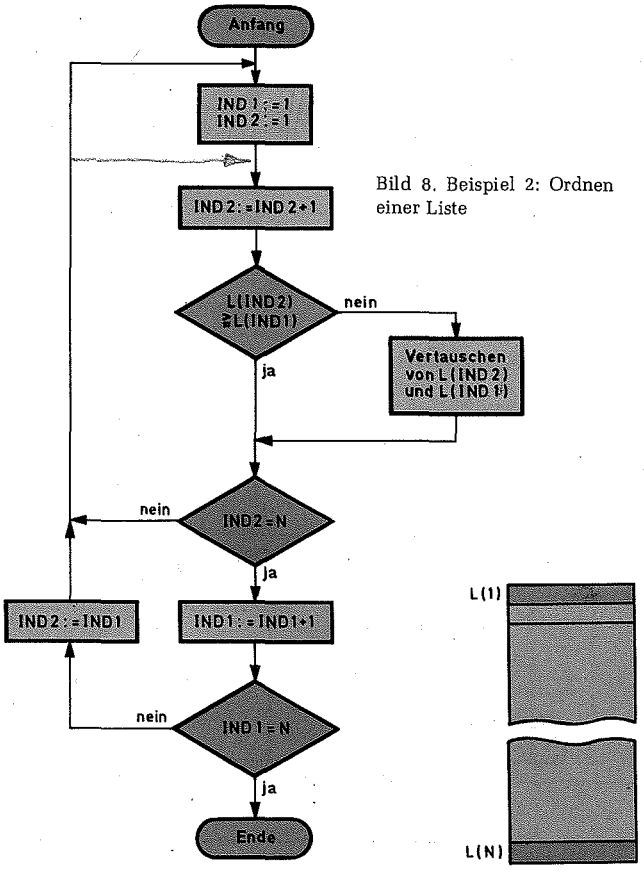


Bild 8. Beispiel 2: Ordnen einer Liste

2.3 Beispiele zur Anwendung der Struktogramme

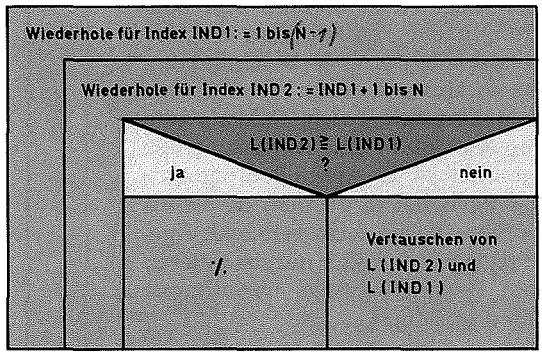
Zwei einfache Beispiele sollen die Anwendung dieser Strukturblöcke veranschaulichen. Es sind wieder die Darstellungen als Programmablaufplan und als Struktogramm gegenübergestellt. Beide Beispiele wurden mit etwas anderen Bezeichnungen in [10] vorgestellt.

Beispiel 1 (Bild 7)

Bestimmen des größten Wertes dreier Variabler A,B,C. Es soll festgestellt werden, welche der Variablen A,B,C den größten Wert aufweist. Der Wert dieser Variablen soll der Variablen D zugewiesen werden.

Beispiel 2 (Bild 8)

Die Eintragungen der Liste L mit der Länge N sollen der Größe nach geordnet werden, das erste Listenelement L(1) soll den niedrigsten und das letzte Listenelement L(N) den höchsten Wert enthalten. Man kann im Struktogramm erkennen, daß in diesem Programmbeispiel nur eine wirkliche Abfrage (Entscheidung) vorhanden ist, nämlich der Größenvergleich der Listenelemente L(IND1) und L(IND2). Die



beiden übrigen Abfragen des Programmablaufplans haben nur die Aufgabe, je eine Wiederholungsschleife aufzubauen. Aus dem Struktogramm geht sehr klar hervor, wie die beiden Schleifen zusammenhängen.

2.4 Beispiel zur Methode der schrittweisen Verfeinerung

Als Beispiel für die schrittweise Verfeinerung soll ein Programm zur Ausgabe der ersten M Primzahlen entworfen werden. Zunächst wird der äußerste Block mit der Gesamtaufgabe als Überschrift gezeichnet (*Bild 9*) Danach wird die Entscheidung getroffen, daß eine Tabelle PRIMZ mit der

(einzugebenden) Länge M aufgebaut werden soll. Dadurch ergeben sich die Teilaufgaben „Eingabe und Initialisierung", „Füllen der Primzahlentabelle PRIMZ" und „Ausgabe der Primzahlentabelle PRIMZ". Um die Funktion wichtiger Strukturblöcke besser erläutern zu können, wird der jeweilige Strukturblock größer gezeichnet als die Folge der Blöcke seiner Teilaufgaben. Außerdem wird der so entstandene freie Raum dazu benutzt, die erstmals verwendeten Variablen (außer den Laufvariablen) kurz zu erläutern. Diese Erläuterung kann z. B. in Sprachen mit Blockstruktur zur Deklaration der Variablen verwendet werden.

Bei der Zerlegung der zweiten Teilaufgabe fällt eine umfangreichere Teilaufgabe „Bestimmen der nächsten Primzahl" an, die der besseren Übersichtlichkeit wegen auf einer neuen Seite (*Bild 10*) untergebracht und dort weiter unterteilt wird. Der abgerundete Rahmen um den Namen der Teilaufgabe verweist auf die Fortsetzungsseite. Zur Erläuterung der Vorgehensweise bei der schrittweisen Verfeinerung genügt die hier erreichte Detaillierung.

Im Strukturblock „Bestimmen der nächsten Primzahl" werden zur Erläuterung des weiteren Ablaufs zwei Formulierungen („FAKMAX: = $\sqrt{\text{ZAHL}}$" und „Ist PRIMZ(K) ein Teiler von ZAHL?") verwendet, die für die Realisierung in ASSEMBLER-Sprache noch wesentlich mehr ins Detail gehend aufbereitet werden müssen.

Schon anhand der beiden ersten Beispiele war deutlich geworden, daß die Struktogramme eine wesentlich kompaktere Darstellung von Verarbeitungsabläufen ermöglich. Dies liegt daran, daß sie über problemnähere Strukturelemente verfügen als die üblichen Programmablaufpläne und daß sie keine Flußlinien zur Festlegung der Reihenfolge der Strukturblöcke benötigen. Bei dem letzten Beispiel wurde ein weiterer Vorteil der Struktogramme sichtbar: Sobald die Detaillierung auf einer Seite so weit fortgeschritten ist, daß für weitere Unterteilungen kein Platz mehr vorhanden ist, werden Teilaufgaben auf neue Seiten „ausgelagert".

Die Tests der mit Struktogrammen erzeugten Programme sind einfach durchzuführen, weil sie im wesentlichen aus den Einzeltests der leicht zu überschauenden Moduln bestehen. Die Nachprüfung der Fehlerfreiheit des Gesamt-Programms wird durch die Beschränkung auf die Baumstruktur und die zu ihrem Aufbau erforderlichen Elemente Folge, Schleife und Verzweigung erleichtert. Eine Anpassung derartiger Programme an geänderte Bedingungen kann leicht vorgenommen werden, da die erforderlichen Eingriffe bei entsprechender Auslegung der Programm-Struktur nur die unteren Ebenen eines Programmzweigs beeinflussen (z. B. Bereich d in Bild 2).

3 Vor- und Nachteile

Auch bei Verwendung der strukturierten Programmierung wird sich nichts daran ändern, daß der erste Programmentwurf für den Papierkorb und erst der zweite für den Rechner gemacht wird. Programme, die nach der Methode der strukturierten Programmierung geschrieben werden, benötigen gegenüber Programmen mit beliebiger Struktur (und allen denkbaren Tricks auf Maschinencode-Ebene) mehr Laufzeit und Speicherplatz, besitzen aber wesentlich mehr Flexibilität, werden schneller geschrieben,

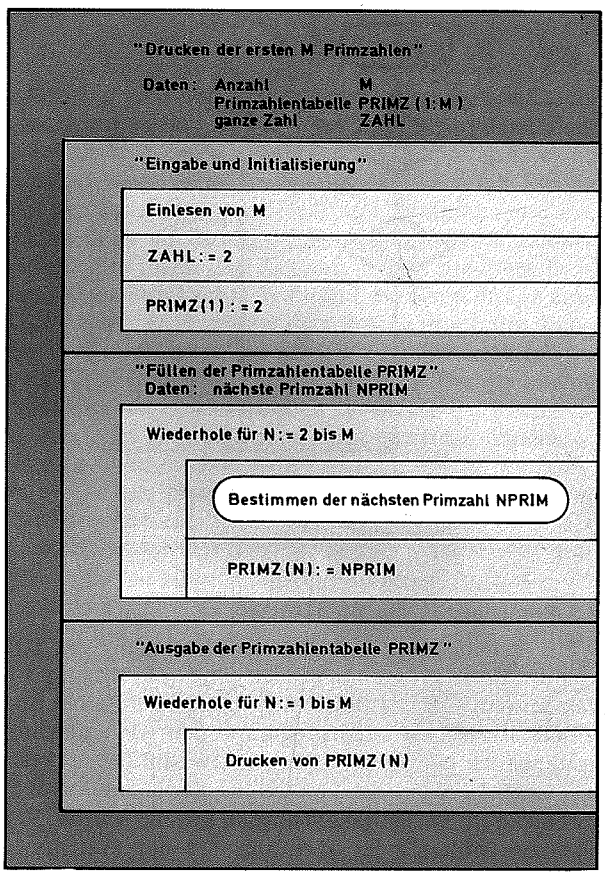


Bild 9. Steuermodul für das Programm zum Drucken der ersten M Primzahlen

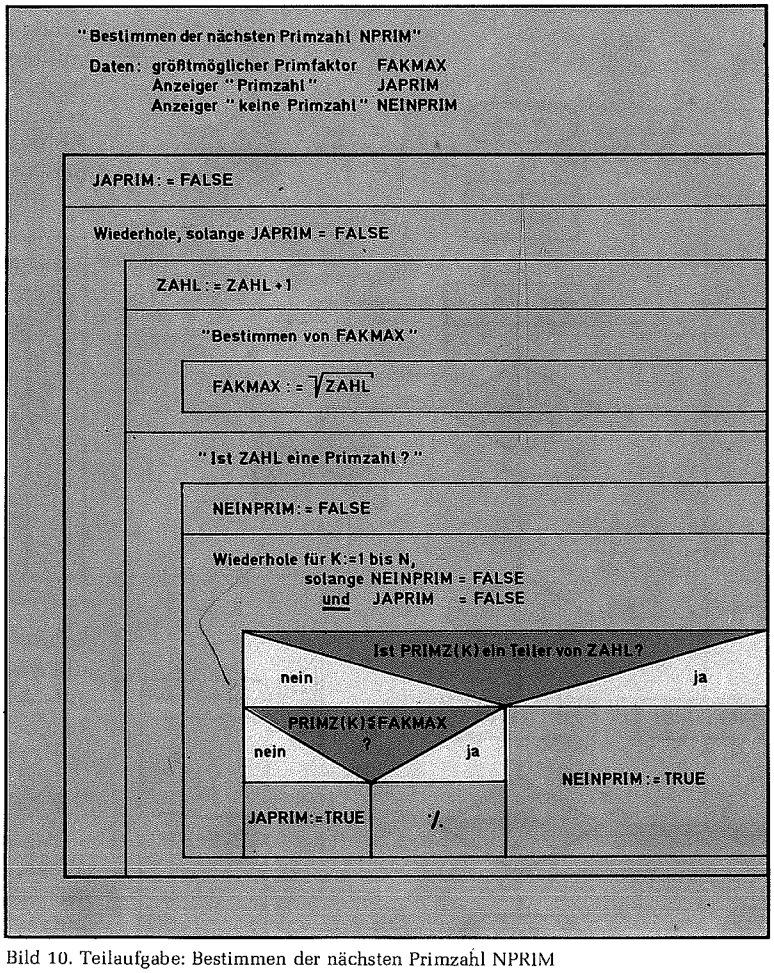


Bild 10. Teilaufgabe: Bestimmen der nächsten Primzahl NPRIM

verstanden und geändert und verursachen dadurch weniger Kosten. Mikroprozessor-Systeme für komplexere Aufgaben brauchen umfangreichere Programme. In solchen Fällen sind i. a. die Stückzahlen nicht sehr groß. Gerade aber bei niedrigen Stückzahlen gehen (schon bei einfachen Programmen) die Softwarekosten stark in die Gesamtkosten ein. Je umfangreicher die Software ist, desto häufiger muß sie außerdem erfahrungsgemäß an neue Gegebenheiten angepaßt werden.

Alle diese Aussagen sprechen dafür, zumindest bei softwarelastigen Projekten mit kleinen und mittleren Stückzahlen zu Hilfsmitteln wie der strukturierten Programmierung zu greifen, die Entwurf, Realisierung und Wartung der Software beschleunigen (und dadurch verbilligen) und die darüber hinaus auch das Projektmanagement und den Qualitätsnachweis für das Produkt erleichtern. Daß die strukturierte Programmierung schon im Begriff ist, im Bereich der Mikroprozessor-Systeme Fuß zu fassen, zeigen die Publikationen [8, 9].

Literatur

[1] *Böhm, B.W.*: Software and its impact: a quantitative assessment. Datamation 19, Mai 1973, S. 48...59

[2] *Böhm, C., Jacopini, G.*: Flow diagrams, Turing machines, and languages with only two formation rules. Communications of the ACM 9 (1966), S. 366...371.

[3] *Dahl, O.-J., Dijkstra, E. W., Hoare, C.A.R.*: Structured Programming. Academic Press, New York 1972.

[4] *Dijkstra, E. W.*: GOTO statement considered harmful. Communications of the ACM 11 (1968), S. 147...148.

[5] *Nassi, I., Shneidermann, B.*: Flow chart techniques for structured programming. SIGPLAN Notices 8 (1973) 8, S. 12...26.

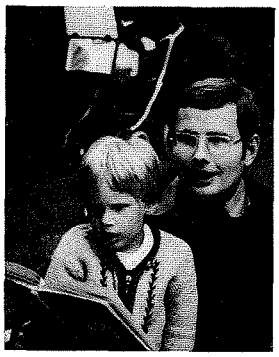
[6] *Schnupp, P.*: Struktogramme – eine neue Methode der Systemplanung. online 12 (1974), S. 736...743.

[7] *Schnupp, P., Floyd, Ch.*: Software, Programmentwicklung und Projektorganisation. Walter de Gruyter, Berlin, New York 1976.

[8] Kursunterlagen „2650 intensive workshop", Beispiel „intelligent typewriter", Philips.

[9] Programmbibliothek Band 1, Grundrechenarten 1. Mikrocomputer-System SAB 8080, Siemens.

[10] Feger, O: Einführung in die Mikrocomputer-Programmierung. ELEKTRONIK 1977, H. 4, S. 79...84.



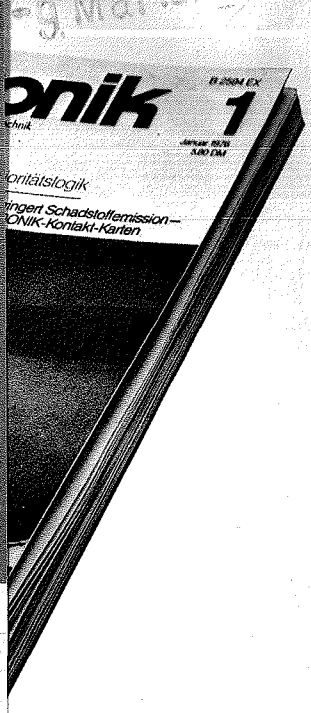
Dipl.-Ing. Frank Heubach stammt aus Königsee/Thüringen. Er studierte Nachrichtentechnik mit Schwerpunkt Informationsverarbeitung an der Technischen Universität Berlin. Von 1969 an war er bei der Philips GmbH, Forschungslabor Hamburg, mit Software-Entwicklung auf den Gebieten grafische Datenverarbeitung, Parallelrechner und Simulation beschäftigt. Seit Anfang 1977 ist er bei Valvo für Kundenunterstützung durch Dokumentation und Seminare auf dem Gebiet Mikroprozessoren tätig.
Hobbys: Hausmusik, Sprachen, Brettspiele, Familie (2 Kinder)
Privattelefon: (0 40) 57 65 54
ELEKTRONIK-Leser seit 1969

Mikroprozessor-Hersteller und Vertriebsadressen

Hersteller	Vertrieb
Advanced Micro Devices (AMD)	Advanced Micro Devices Mikro Elektronik GmbH, Herzog-Heinrich-Str. 3, 8000 München, Tel. (0 89) 53 95 88
AEG-Telefunken	AEG-Telefunken, FB-Halbleiter, Theresienstr. 2, 7100 Heilbronn 2. Tel. (0 71 31) 8 82-1
American Micro Systems Inc. (AMI)	AMI Microsystems GmbH, Rosenheimer Str. 30, Suite 237, 8000 München 80, Tel. (0 89) 48 30 81
ASEA HAFO	HEK GmbH, Postfach 18 10, 2400 Lübeck, Tel. (04 51) 5 29 91
Computer Automation	Geveke, Gutenbergring 40, 2000 Norderstedt, Tel. (0 40) 5 23 50 61
Digital Equipment	Digital Equipment, Wallenstein-platz 2, 8000 München 40, Tel. (0 89) 3 50 31
Electronic Arrays	Electronic Arrays GmbH, Hofmann-str. 20, 8000 München 70, Tel. (0 89) 7 85 31 68
Electronic Memories and Magnetics Corp.	EMM GmbH, Ferdinandspl. 14, 6380 Bad Homburg, Tel. (0 61 72) 66 29
Fairchild	Fairchild Halbleiter GmbH, Daimlerstr. 15, 8046 Garching-Hochbrück, Tel. (0 89) 32 00 31
Ferranti	Ferranti GmbH, Widenmayerstr. 5, 8000 München 22, Tel. (0 89) 29 73 53
General Automation	General Automation, Heider-Hof-Weg 23, 5100 Aachen, Tel. (0 24 05) 6 41
General Instrument (GI)	General Instrument GmbH, Nordendstr. 1a, 8000 München 40, Tel. (0 89) 28 40 31
Harris Semiconductor	Kontron Elektronik GmbH, Oskar-von-Miller-Str. 1, 8051 Eching, Tel. (0 81 65) 7 73 39
Hitachi Ltd.	Hitachi Components, Immer-mannstr. 15, 4000 Düsseldorf, Tel. (02 11) 35 30 73
Intel Corporation	Intel Semiconductor GmbH, Seidlstr. 27, 8000 München 2, Tel. (0 89) 55 81 41 Alfred Neye-Enatechnik GmbH, Schillerstr. 14, 2085 Quickborn, Tel. (0 41 06) 61 21
Intersil Corporation	Spezial Electronic KG, Ortlerstr. 8, 8000 München 70, Tel. (0 89) 7 60 00 31
Monolithic Memories Inc. (MMI)	MMI GmbH, Mauerkircherstr. 4, 8000 München 80, Tel. (0 89) 98 26 01
MOS-Technology	Neumüller GmbH, Eschenstr. 2, 8021 Taufkirchen, Tel. (0 89) 61 18-1

Hersteller	Vertrieb
Mostek	Mostek GmbH, Talstr. 172, 7024 Filderlinden-Bernhausen, Tel. (07 11) 70 10 96
Motorola	Motorola GmbH, GB Halbleiter, Münchner Str. 18, 8043 Unterföhring, Tel. (0 89) 95 10 41
National Semiconductor Corp. (NS)	National Semiconductor GmbH, Industriestr. 10, 8080 Fürsten-feldbruck, Tel. (0 81 41) 10 31
Nippon Electric Comp. (NEC)	NEC Electronic (Europe) GmbH, Immermannstr. 22, 4000 Düssel-dorf, Tel. (02 11) 35 70 88
Periphere Computer Systeme (PCS)	PCS GmbH, Dompfaffweg 10, 8000 München 82, Tel. (0 89) 46 40 35
Plessey	Plessey (Deutschland) GmbH, Motorstr. 56, 8000 München 40, Tel. (0 89) 3 51 60 21
Prolog	Spezial Electronic
Raytheon Corp.	Raytheon Halbleiter GmbH, Thal-kirchner Str. 74, 8000 München 2, Tel. (0 89) 53 96 93
RCA	RCA GmbH, Bereich Halbleiter, Ju-stus-v.-Liebig-Ring, 2085 Quickborn, Tel. (0 41 06) 20 01
Rockwell International	Rockwell International GmbH, Fraunhoferstr. 11, 8033 Martins-ried, Tel. (0 89) 8 59 95 75
SGS-Ates	SGS-Ates Deutschland GmbH Haidling 17, 8018 Grafing, Tel. (0 80 92) 50 71
Siemens	Siemens AG, ZVW-104, Postfach 103, 8000 München 1
Solid State Scientific	Neutron GmbH, Postfach 124, 6050 Offenbach, Tel. (06 11) 81 39 30
Synertek	Astronic, Winzererstr. 47 d, 8000 München 40, Tel. (0 89) 30 40 11
Valvo/Signetics	Valvo, Unternehmensbereich Bau-elemente der Philips GmbH, Postfach 10 63 23, 2000 Hamburg 1
Texas Instruments Inc.	Texas Instruments Deutschland GmbH, Haggertystr. 1, 8050 Freising, Tel. (0 81 61) 8 02 78
Thomson-CSF	Thomson-CSF GmbH, Fallstr. 42, 8000 München 70, Tel. (0 89) 76 75-1
Toshiba	Toshiba Europa, Emmastr. 24, 4000 Düsseldorf, Tel. (02 11) 72 30 91
Western Digital	Spezial Electronic
Zilog	Kontron

Hier können Sie die Elektronik bestellen



Bestellkarte für ein Elektronik-Abonnement

Bitte übersenden Sie mir bis auf Widerruf monatlich ein Elektronik-Heft – also 12 Hefte im Jahr. Kündigung ist bei jährlicher Zahlweise 8 Wochen zum Kalenderjahresende möglich, bei 1/4jährlicher Zahlweise 8 Wochen zum Quartalsende.

ab Monat _____

Name/Vorname

Straße/Hs.-Nr.

PLZ Ort

Beruf

Die Abonnementsgebühr zahle ich nach Erhalt der Rechnung.*

Datum

Unterschrift

* Falls Sie Abbuchung von Ihrem Konto wünschen, bitten wir Sie, die Rückseite zu beachten.

Ich wünsche folgende Zahlungsweise (Zutreffendes habe ich angekreuzt):

☐ 1 x jährlich DM 65.– Die Jahresabonnements-Gebühr ist im In- und Ausland gleich.

☐ 4 x jährlich DM 18.– = insgesamt DM 72.– pro Jahr (im Ausland nicht möglich).

Alle Nebenkosten, einschließlich Porto, sind in diesen Preisen enthalten. MIK 2/78

Bestellkarte für ein Elektronik-Abonnement

Bitte übersenden Sie mir bis auf Widerruf monatlich ein Elektronik-Heft – also 12 Hefte im Jahr. Kündigung ist bei jährlicher Zahlweise 8 Wochen zum Kalenderjahresende möglich, bei 1/4jährlicher Zahlweise 8 Wochen zum Quartalsende.

ab Monat _____

Name/Vorname

Straße/Hs.-Nr.

PLZ Ort

Beruf

Die Abonnementsgebühr zahle ich nach Erhalt der Rechnung.*

Datum

Unterschrift

* Falls Sie Abbuchung von Ihrem Konto wünschen, bitten wir Sie, die Rückseite zu beachten.

Ich wünsche folgende Zahlungsweise (Zutreffendes habe ich angekreuzt):

☐ 1 x jährlich DM 65.– Die Jahresabonnements-Gebühr ist im In- und Ausland gleich.

☐ 4 x jährlich DM 18.– = insgesamt DM 72.– pro Jahr (im Ausland nicht möglich).

Alle Nebenkosten, einschließlich Porto, sind in diesen Preisen enthalten. MIK 2/78

nen, ohne das
er nicht mehr
ht ein ELEK-
t auf individu-

nt für 12 Hefte

len Sie hier!

0. Sep. 1982

, Software" kostet

DM 19.– (zuzüglich

kt beim FRANZIS-

gegen Vorauszah-
l plus 2.– DM Porto)
tscheckkonto Mün-
NIK-Sonderheft 2,
Ihre volle Anschrift

ünstigten Kombibe-

89/51 17-2 40

eich beim Fachbuch

8 6 148

Elektronik

die große Fachzeitschrift für angewandte Elektronik und Datentechnik

Die ELEKTRONIK behandelt alles Wissenswerte der professionellen Elektronik, der Datentechnik im industriellen Bereich sowie die Anwendung der Elektronik in der Fertigungs-, Meß-, Steuer- und Regelungstechnik, der Automation und Energieumsetzung, des Verkehrswesens, der Medizin und der Forschung. ELEKTRONIK-Beiträge sind praxisnah geschrieben und damit bei den täglich anfallenden Problemen direkt verwertbar; sie vermitteln die oft teuer bezahlte Erfahrung anerkannter Experten und bringen Grundlagen des Ingenieurwesens in Form der „Elektronik-Arbeitsblätter" und marktgerechte Neuheitenberichte im „Elektronik-Markt" sowie aktuelle Branchen-Nachrichten im „Elektronik-Express".

Umfangreiche Reports stellen von Zeit zu Zeit das Wichtigste über bestimmte Themen zusammen und entwirren das internationale Angebot durch übersichtliche Tabellen.

Vier „Lexikon-Karten" pro Heft bereichern laufend den technischen Wortschatz des Lesers und enträtseln schwer zu merkende Abkürzungen. Eine umfangreiche Literaturschau gibt Einblick in die aktuellsten Zeitschriften-, Buch- und Firmenpublikationen des In- und Auslands.

Mikroprozessor und Mikrocomputer werden in der ELEKTRONIK besonders intensiv betreut, um auch den Lesern aus anderen Branchen, z.B. aus dem Maschinenbau, den Einstieg in die umwälzend neue Ära der „intelligenten" Maschinen und der „verteilten Intelligenz" in der Datentechnik zu erleichtern.

Und wer über wenig Zeit verfügt, kann den Rubriken „Fortschritte der Technik" und „ELEKTRONIK-Notizen" mit einem Blick das Neueste aus Forschung und Anwendungen der Elektronik entnehmen.

Das lebenslange Lernen, ohne das heute der Elektroniker nicht mehr auskommt, ermöglicht ein ELEKTRONIK-Abonnement auf individuelle Weise.

Das Jahresabonnement für 12 Hefte kostet DM 58,–.

Eine Bestellkarte finden Sie hier!

30. Sep. 1982

Das Ihnen vorliegende Sonderheft 2 der ELEKTRONIK zum Thema **„Mikroprozessoren, Software"** kostet DM 16.– (zuzüglich DM 2.– Porto), Inhalt 120 Seiten.

Das Sonderheft 1 der ELEKTRONIK beinhaltet die **„Mikroprozessor-Hardware".** Preis DM 19.– (zuzüglich DM 2,.– Porto), Inhalt 144 Seiten.

Bestellung dieser Sonderhefte:

Sie erhalten die Hefte bei allen Bahnhofsbuchhandlungen, guten Buchhandlungen oder direkt beim FRANZIS-Verlag.

Bitte haben Sie Verständnis, daß der Verlag Einzelhefte aus organisatorischen Gründen nur gegen Vorauszahlung liefern kann. Wir bitten Sie in diesem Fall als Bestellung den Betrag von 21.– DM (19.– DM plus 2.– DM Porto) für Sonderheft 1, bzw. 18.– DM (16.– DM plus 2.– DM Porto) für Sonderheft 2 auf unser Postscheckkonto München Nr. 5758-807 mit Hinweis „ELEKTRONIK-Sonderheft 1, Hardware" bzw. „ELEKTRONIK-Sonderheft 2, Software" zu überweisen. Bitte vergessen Sie nicht, auf dem Zahlungsbeleg in Druckschrift Ihre volle Anschrift anzugeben. Sofort nach Eingang der Zahlung senden wir Ihnen das Heft zu.

Wenn Sie beide Hefte zusammen direkt beim Verlag bestellen, bitten wir Sie, den etwas vergünstigten Kombibetrag von 34.– DM (32.– DM plus 2.– DM Porto) zu überweisen.

FRANZIS-VERLAG, KARLSTRASSE 37, 8000 MÜNCHEN 2, TELEFON 0 89/51 17-2 40

Das Heft erhalten Sie in der Schweiz auch beim Verlag Thali AG, 6285 Hitzkirch und in Österreich beim Fachbuch Center Erb, 1061 Wien, Amerlingstraße 1

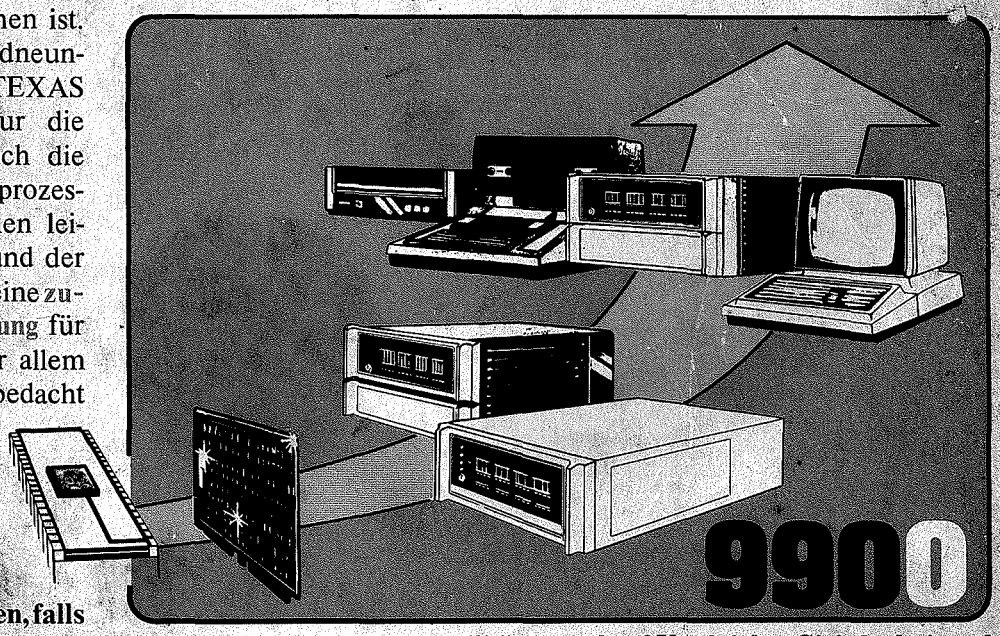
1978 b 148

Das Konzept

Wir von TEXAS INSTRUMENTS waren die ersten, die für Mikroprozessoren, Mikrocomputer und Minicomputer ein klares und zukunftssicheres Konzept aufgestellt haben, das Hand und Fuß hat: das 'Neunundneunzighundert-Konzept'.
Wir meinen, das sollte einmal ganz klar gesagt werden!

Erst durch die Software-Kompatibilität von Mikros und Minis kann man von „Konzept" sprechen, denn was heute noch Mini ist, ist vielleicht morgen schon Mikro. Deshalb sind alle Mitglieder des 9900-Konzeptes software-kompatibel – das heißt, sie verstehen und sprechen alle die gleiche Sprache. In die Software-Entwicklung wird jeder Anwender in den kommenden Jahren große Investitionen an Zeit und Geld für Ausbildung, Prototypensysteme und Programmentwicklungen vornehmen müssen und trägt das große Risiko, daß der gewählte Mikroprozessor durch die rapide Steigerung der Halbleiter-Komplexität eventuell schon in wenigen Jahren als technisch überholt anzusehen ist.

Deshalb bauen im 'Neunundneunzighundert-Konzept' von TEXAS INSTRUMENTS nicht nur die Minicomputer, sondern auch die Mikrocomputer und Mikroprozessoren alle auf dem gleichen leistungsfähigen Befehlssatz und der Architektur auf. Nur so ist eine zukunftssichere Standardisierung für den Anwender möglich, vor allem wenn man langfristig darauf bedacht ist, die jeweils optimalen Hardware-Kosten erzielen zu können, ohne daß bis dahin erstellte Programme wertlos werden.



Ob andere das Gleiche meinen, falls sie im Zusammenhang mit Mikroprozessoren von „Software-Kompatibilität" sprechen? Vergleichen Sie's doch einmal.

TEXAS INSTRUMENTS
Deutschland GmbH

Haggertystraße 1 · 8050 Freising Telefon: 08161/80-1 · Telex: 05-26 529